বাংলা ভাষায় রচিত প্রথম কমপ্লিট C রেফারেন্স এর বই

C ল্যাংগুয়েজ

কমপ্লিট রেফারেন্স

- অসংখ্য উদাহরণ প্রোগ্রাম
- ফ্লপি ডিস্ক প্রোগ্রাম
- হার্ড ডিস্ক প্রোগ্রাম
- মাউস প্রোগ্রাম
- ভাইরাস প্রোগ্রাম
- গেইমস প্রোগ্রাম
- দু'টি সংখ্যা বিশিষ্ট তৈরী
- C++ সম্পর্কে প্রোজেক্ট ধারণা

সরদার তারেক হাবিব

বাংলা ভাষায় রচিত প্রথম কমপ্লিট C রেফারেন্স এর বই

C ল্যাংগুয়েজ

কমপ্লিট রেফারেন্স

- অসংখ্য উদাহরণ প্রোগ্রাম
- ফ্লপি ডিস্ক প্রোগ্রাম
- হার্ড ডিস্ক প্রোগ্রাম
- মাউস প্রোগ্রাম
- ভাইরাস প্রোগ্রাম
- গেইম প্রোগ্রাম
- দু'টি সংখ্যা বিশিষ্ট স্ক্রীণ বক্স তৈরী
- C++ সম্পর্কে প্রোজেক্ট ধারণা

সরদার তারেক হাবিব

C ল্যাংগুয়েজ (কমপ্লিট রেফারেন্স)

(B.Sc (Hons), Diploma, Short Course-এ অধ্যয়নরত শিক্ষার্থীসহ
যে কোন C ল্যাংগুয়েজ শিক্ষার্থীদের জন্য প্রযোজ্য)

সরদার তারেক হাবিব

জ্ঞানকোষ প্রকাশনী
৩৮/২-ক, বাংলাবাজার, ঢাকা-১১০০

C ল্যাংগুয়েজ (কমপ্লিট রেফারেন্স)

প্রকাশক : শরীফ হাসান তরফদার
জ্ঞানকোষ প্রকাশনী
৩৮/২-ক, বাংলাবাজার, ঢাকা ১১০০।

প্রকাশকাল : প্রথম প্রকাশ : অক্টোবর, ২০০০ইং
দ্বিতীয় প্রকাশ : অক্টোবর, ২০০১ইং
তৃতীয় প্রকাশ : আগষ্ট, ২০০৩ইং
চতুর্থ প্রকাশ : জানুয়ারী ২০০৮ ইং

প্রচ্ছদ পরিকল্পনায় : গ্রাফিক স্ক্যান
পল্টন, ঢাকা।

কম্পোজ : ফয়েজ স্ক্যান পয়েন্ট
৩৮, বাংলাবাজার, ঢাকা-১১০০।
ফোন : ৭১১৫০৭৪

মুদ্রণ : নোভা প্রেস এন্ড পাবলিকেশন্স
১৫/বি, মিরপুর রোড, ঢাকা ১২০৫
ফোন ঃ ৯৬৬৭৯১৯

মূল্য : ৩৬০.০০ টাকা মাত্র।

C Language (Complete Reference)

BY : SARDAR TAREQ HABIB

Gyankosh Prokashani

38/2-KA, Banglabazar, Dhaka
Phone : 7118443, 8112441, 8623251

উৎসর্গ

পিতা ঃ মোঃ হাব্বির রহমান সরদার।

মাতা ঃ সৈয়দা নূরুননাহার।

ভাই ঃ সরদার ওমর হাবিব।

ভূমিকা

প্রথমেই স্মরণ করছি আল্লাহ্‌কে যিনি আমাকে এই বইটি লেখার ক্ষমতা দান করেছেন।

বর্তমান বিশ্বে C ল্যাংগুয়েজ একটি অতি পরিচিত প্রোগ্রামিং ল্যাংগুয়েজ। এর মাধ্যমে অনেক বড় বড় সফট্‌ওয়্যার তৈরি করা হয়েছে। C-ল্যাংগুয়েজ তৈরির পর থেকে ১৯৭৮ খ্রিস্টাব্দ পর্যন্ত Kernigham ৪' Ritchie-এর C Reference Manual-কে C-এর standard হিসেবে ধরা হতো। পরবর্তিতে American National Standard Institute (ANSI) C-ল্যাংগুয়েজের একটি আদর্শ মান নির্ধারণ করেন। Turbo C, Borlond C, Microsoft C হল বিভিন্ন প্রতিষ্ঠানের তৈরি C ল্যাংগুয়েজ যারা এই ANSI C-এর আদর্শ মান অনুসরণ করেছেন। যদিও এই বইটির সকল প্রোগ্রাম Turbo C (Tc3)-তে লেখা, তবে এই বইয়ের সকল প্রোগ্রাম ছোটখাট পরিবর্তন ছাড়াই Borland C কিংবা Microsoft C-তে Run করবে।

এই বইটিতে প্রথমে C ল্যাংগুয়েজ সম্বন্ধে বিস্তারিত ধারণা দেয়া হয়েছে এবং পরবর্তিতে C ল্যাংগুয়েজ ব্যবহার করে প্রোগ্রাম কিভাবে করতে হয় তা বর্ণনা করা হয়েছে। নবীন কিংবা অভিজ্ঞ প্রোগ্রামারগণ এই বইটি পাঠ করে সুফল পাবেন বলে আমি মনে করি। এই বই-এ বিভিন্ন স্থানে ইংরেজি শব্দ ব্যবহার করা হয়েছে এজন্য যে কিছু কিছু ইংরেজি শব্দের বাংলা করা সম্ভব হয়নি। কারণ এগুলো হল Programming jargon। পাঠকদের এটুকু ইংরেজি বোঝার ক্ষমতা থাকতে হবে।

যেসকল পাঠক কম্পিউটার বিজ্ঞানে পড়ছেন তাদের সুবিধার জন্য বইয়ের শেষের দিকে কয়েকটি project বর্ণনা করা হয়েছে। এছাড়াও বইয়ের শেষের দিতে ROM-BIOSএবং DOS service-এর একটি লিস্ট দেয়া আছে যা অভিজ্ঞ প্রোগ্রামারদের বিশেষ সহায়তা করবে বলে আমি মনে করি।

এই বইয়ে Mouse, Floppy disk, Harddisk এবং graphics-এর উপর প্রোগ্রামিং নিয়ে বিস্তারিত আলোচনা করা হয়েছে।

আমার এই বই পাঠ করে পাঠকেরা C ল্যাংগুয়েজ সম্বন্ধে স্বচ্ছ ধারণা পাবেন বলে আমি আশা করি।

SARDAR TAREQ HABIB

20.10.2000

কৃতজ্ঞতা জ্ঞাপন

এই বই লেখার সময় যারা প্রত্যক্ষ বা পরোক্ষভাবে সহায়তা করেছেন তাদের সকলকেই আমার কৃতজ্ঞতা জ্ঞাপন করলাম। যাদের উৎসাহ এবং পরামর্শে বইটি লেখা সম্ভব হয়েছে তাদের মধ্যে পপুলার বুকস্ এর আনিস, সুমন, মিল্টন, ইমরান, এমদাদ, আরিফ, সাজ্জাদ, পাভেল, রঞ্জন, তারক, বাশার, মানস প্রমূখের নাম উল্লেখযোগ্য। এছাড়াও এই মুহূর্তে কৃতজ্ঞতা জ্ঞাপন করছি জ্ঞানকোষ প্রকাশনীর সত্বাধিকারী জনাব শাহীদ হাসান কে। পরিশেষে সকলকে ধন্যবাদ জ্ঞাপন করলাম।

সরদার তারেক হাবিব।

২০/১০/২০০০

পাঠকদের জ্ঞাতার্থে

এই বইটি নবীন এবং অভিজ্ঞ উভয় প্রোগ্রামারদের কথা স্মরণ রেখেই লেখা হয়েছে। বইটির প্রথমদিকের অধ্যায়গুলো অতি সহজ এবং বিস্তারিতভাবে লেখা হয়েছে যাতে করে নবীন শীক্ষার্থীরা সহজেই C ল্যাংগুয়েজ প্রোগ্রাম শিখতে পারেন। শেষোক্ত অধ্যায়গুলো অভিজ্ঞ প্রোগ্রামারদের বিশেষভাবে সহায়তা করবে। C ল্যাংগুয়েজ-এ প্রোগ্রাম শিখতে হলে শিক্ষার্থীদের যথেষ্ট ধৈর্যশীল হতে হবে বলে আমি মনে করি।

সরদার তারেক হাবিব।

বিশেষ সাবধানতা

C++ অধ্যায় ব্যাতীত অন্যান্য সব অধ্যায়-এর প্রোগ্রাম C extension-এ save করে Run করাতে হবে। যেমন— কোন প্রোগ্রাম ফাইলের নাম যদি test দিতে চাই তবে test.c লিখতে হবে।

ফাইলের নামের শেষে .c না লিখলে প্রোগ্রাম ফাইল .cpp extension-এ save হবে।

আমি আবারো বলছি যে, এই বইয়ের প্রোগ্রামগুলো .c extension-এ save করে Run না করালে, অনেক প্রোগ্রাম error দেখা যাবে।

Graphics প্রোগ্রাম করার সময় initgraph ( ) ফাংশনের মধ্যে "c:\\tc3\\bgi" address ব্যবহার করেছি। পাঠকের কম্পিউটারে TC3 প্রোগ্রামের নাম যদি Tc হয় তবে এখানে "c:\\tc\\bgi" লিখতে হবে। কিংবা Tc3 সফট্ওয়্যারটি যদি c:\-drive-এ না থেকে D:\drive-এ থাকে তবে "D:\\Tc3\\bgi" লিখতে হবে।

পরিবেশক / প্রাপ্তিস্থান

❑ **ঢাকা ঃ বাংলাবাজার**
১। সিসটেক পাবলিকেশনস্

❑ **ঢাকা ঃ নিউ মার্কেট**
১। বুক ভিউ।
২। বুক ওয়ার্ল্ড
৩। ঢাকা বুক কর্পোরেশন।
৪। আলীগড় লাইব্রেরী।
৫। বুক মার্ট।

❑ **ঢাকা ঃ স্টেডিয়াম মার্কেট**
১। অনুপম জ্ঞান ভান্ডার।

❑ **ঢাকা ঃ নীলক্ষেত**
১। ডলফিন।
২। ফরিদা করপোরেশন।
৩। পপুলার বুকস্
৪। মডার্ন বুক সেন্টার।
৫। খন্দকার বুক হাউস।
৬। স্টুডেন্টস্ লাইব্রেরী।
৭। বুক সেন্টার।

❑ **ঢাকা ঃ ফার্মগেট**
১। তোফাজ্জল বুক হাউস।
২। এম. জামান এন্টারপ্রাইজ।

❑ **ঢাকা ঃ মিরপুর**
১। বুক প্যালেস।
২। আইডিয়েল লাইব্রেরী।

❑ **ঢাকা ঃ কচুক্ষেত (ক্যান্টনমেন্ট)**
১। লাকীবুকস এন্ড ষ্টেশনারী।

❑ **কুমিল্লা ঃ**
১। শাহিন লাইব্রেরী।

❑ **খুলনা ঃ**
১। সোহাগ বুক হাউস।
২। খুলনা বুক সেন্টার।
২। নূর লাইব্রেরী।
৩। সালেহিয়া লাইব্রেরী।
৪। আজিজিয়া লাইব্রেরী।

❑ **যশোর ঃ**
১। জনতা লাইব্রেরী, দড়াটানা রোড।

❑ **চট্টগ্রাম ঃ**
১। আযাদ বুকস্, আন্দরকিল্লা, চট্টগ্রাম।
২। গুলিস্তান লাইব্রেরী, বিপনী বিতান, চট্টগ্রাম।
৩। বুক ম্যান, বিপনী বিতান, চট্টগ্রাম।
৪। কারেন্ট বুক সেন্টার, জলসা সিনেমা, চট্টগ্রাম।
৫। জেনুইন লাইব্রেরী, আন্দরকিল্লা, চট্টগ্রাম।
৬। সোনলী বুক হাউস, আন্দরকিল্লা, চট্টগ্রাম।
৭। গ্রন্থ বিতান, আন্দরকিল্লা, চট্টগ্রাম।
৮। বই ঘর, লাকী প্লাজা, চট্টগ্রাম।
৯। বুক ল্যাণ্ড, সেন্ট্রাল প্লাজা, চট্টগ্রাম।
১০। বুক চয়েজ, চিটাগাং শপিং সেন্টার, চট্টগ্রাম।

❑ **রাজশাহী ঃ**
১। সবুজ লাইব্রেরী।
২। আলীগড় লাইব্রেরী।
৩। কই পত্র।
৪। সংলাপ।
৫। তানজীব লাইব্রেরী।
৬। সবুজ লাইব্রেরী।
৭। বই বিচিত্রা।
৮। বীনাপানি বুক ডিপো।

❑ **কুষ্টিয়া ঃ**
১। বইমেলা, এন. এস. রোড।

❑ **চুয়াডাঙ্গা ঃ**
১। পুথিঘর লাইব্রেরী, বড় বাজার।

❑ **সিলেট ঃ**
১। পপি লাইব্রেরী।
২। মালঞ্চ বুক সেন্টার।
২। নিউনেশন লাইব্রেরী।
৩। অক্সফোর্ড লাইব্রেরী।
৫। ইসলামিয়া লাইব্রেরী।
৬। প্যারাগন লাইব্রেরী।
৭। আদর্শ লাইব্রেরী।
৮। সেন্ট্রাল লাইব্রেরী।
৯। জনতা লাইব্রেরী।
১০। সামী লাইব্রেরী।

❑ **রংপুর ঃ**
১। বিপণী বিচিত্রা।
২। সাহিত্য ভান্ডার।
৩। টাউন ষ্টোর লাইব্রেরী।

উত্তর বঙ্গের একমাত্র পরিবেশক কোরান হাদিস মঞ্জিল, বগুড়া।

এছাড়াও দেশের বিভাগীয় এবং জেলা সদরের সকল অভিজাত বইয়ের দোকানসমূহে।

সূচিপত্র (Contents)

সংক্ষিপ্ত সূচিপত্র

বিস্তারিত সূচীপত্র

সাধারণ আলোচনা

Some Fundamentals
(কিছু মৌলিক বিষয়ে আলোচনা)

- প্রোগ্রামিং ল্যাংগুয়েজ
- হার্ডওয়্যার এবং সফটওয়্যার
- মাইক্রোপ্রোসেসর এবং রেজিস্টার
- মেমোরী বা স্মৃতিস্থান
- **Bit** এবং **Byte**
- **ASCII** ক্যারেক্টার কোড
- নাম্বার সিস্টেম

এই অধ্যায়ে কম্পিউটার এবং প্রোগ্রামিং সম্পর্কে সাধারণ আলোচনা করা হয়েছে। যারা কম্পিউটার ব্যবহারে এবং প্রোগ্রামিং-এ একেবারেই নতুন তাদের জন্য এই অধ্যায়টি বিশেষ উপকারে আসবে। কম্পিউটারের মেমোরী এবং number system সম্পর্কেও এখানে আলোচনা করা হয়েছে।

প্রোগ্রামিং ল্যাংগুয়েজ ঃ

প্রোগ্রামিং ল্যাংগুয়েজের মাধ্যমে কম্পিউটারকে বিভিন্ন কাজের নির্দেশ দেয়া হয়। আমরা বাংলা কিংবা ইংরেজি ভাষা বুঝি। কম্পিউটারকে বাংলা কিংবা ইংরেজি লিখে কোন কাজের নির্দেশ দেয়া হলে কম্পিউটার তা বুঝতে পারে না। প্রোগ্রামিং ল্যাংগুয়েজ এমন একটি ভাষা যা কম্পিউটার বুঝতে পারে। তাই কোন ব্যক্তি প্রোগ্রামিং ল্যাংগুয়েজের মাধ্যমে কম্পিউটারকে কোন নির্দেশ দিলে, কম্পিউটার তা পালন করে।

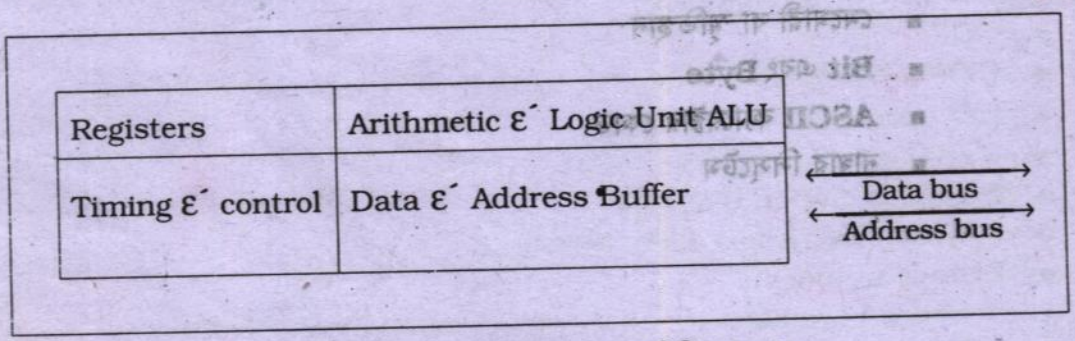
হার্ডওয়্যার এবং সফট্ওয়্যার ঃ

কম্পিউটারের বিভিন্ন যন্ত্রাংশকে হার্ডওয়্যার বলা হয়। যেমন ঃ mouse, keyboard, microprocessor ইত্যাদি। হার্ডওয়্যার পরিচালনার জন্য সফওয়্যার-এর প্রয়োজন হয়। সফট্ওয়্যার হল কিছু প্রোগ্রামসামগ্রী যা বৈদ্যুতিক তরঙ্গের মাধ্যমে কম্পিউটারকে নির্দিষ্ট কোন কাজ করতে বাধ্য করে।

মাইক্রোপ্রোসেসর (Microprocessor) :

মাইক্রোপ্রোসেসরকে কম্পিউটারের ব্রেইন এবং হৃদপিণ্ড বলা যেতে পারে। এর মধ্যে বিভিন্ন রকম বৈদ্যুতিক বর্তনী থাকে যার মাধ্যমে মাইক্রোপ্রোসেসর নিম্নলিখিত কাজ করে থাকে ঃ

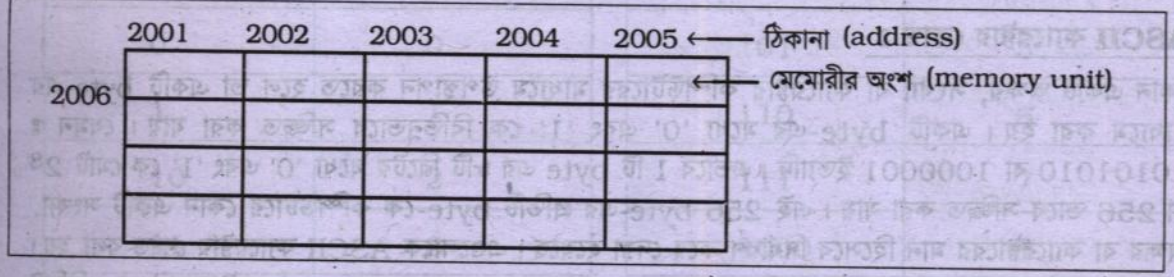
- তথ্য প্রক্রিয়াকরণ।
- গাণিতিক ও যৌক্তিক কাজ।
- প্রোগ্রামের নির্দেশ পালন।
- বিভিন্ন যন্ত্রাংশের সমন্বয় সাধন।
- সময় গণনা এবং বিভিন্ন সংকেত প্রদান।



চিত্র ঃ Microprocessor-এর বিভিন্ন অংশ।

মেমোরী (Memory) :

মেমোরী হল কম্পিউটারের স্মৃতিস্থান। বিভিন্ন তথ্য সংরক্ষণের জন্য মেমোরী ব্যবহৃত হয়। কম্পিউটারে মেমোরী ক্ষুদ্র ক্ষুদ্র অংশে বিভক্ত। এই প্রতিটি অংশের নির্দিষ্ট address বা ঠিকানা থাকে। মেমোরীতে রক্ষিত তথ্য যেমন গঠন করা সম্ভব তেমনই তা মুছে নতুন তথ্য সংরক্ষণ করাও সম্ভব। তবে মেমোরীর address পরিবর্তন করা সম্ভব নয়।



চিত্র ঃ কম্পিউটারের স্মৃতিস্থানের রূপক চিত্র।

Registers :

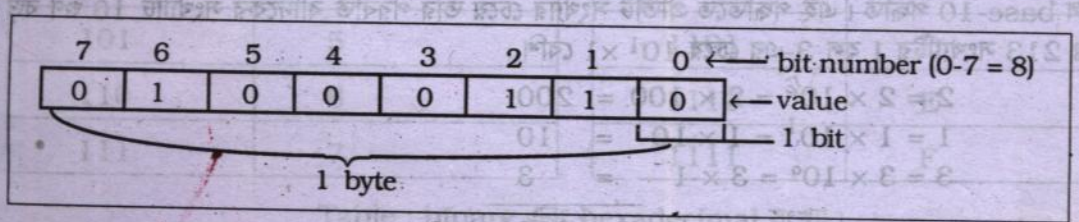
রেজিস্টার মাইক্রোপ্রোসেসরের অভ্যন্তরে ইলেকট্রনিক বর্তনীর সমন্বয়ে গঠিত ক্ষুদ্র স্মৃতিস্থান। মাইক্রোপ্রোসেসর যখন তথ্য প্রক্রিয়াকরণ করে তখন স্বল্প সময়ের জন্য তথ্য সংরক্ষণের উদ্দেশ্যে রেজিস্টার ব্যবহৃত হয়। বিভিন্ন ধরনের রেজিস্টার বিভিন্ন কাজে ব্যবহৃত হয়। রেজিস্টার দুই প্রকারের–

—General Purpose Register
— Special Purpose Register

Bit এবং Byte :

কম্পিউটার সমস্ত কাজ করে ইলেক্ট্রনিক বর্তনীর মাধ্যমে। কোন ইলেক্ট্রনিক বর্তনী 'on' বা 'off' এই দু'টি অবস্থায় থাকতে পারে। এখানে 'on' এবং 'off'-কে যথাক্রমে 1 এবং 0 ধরা হয়। আমরা কম্পিউটারে যে সব তথ্য সংরক্ষণ করি তা মেমোরীতে 1 এবং 0-এর সমন্বয়ে সংরক্ষিত হয়। এই 1 এবং 0 কে bit বা binary সংখ্যা বলা হয়। অর্থাৎ একটি bit-এর দুটি অবস্থার যে কোন একটি থাকতে পারে— '0' কিংবা '1'। যে কোন সংখ্যা বা অক্ষর বা ক্যারেক্টার এই 0 বা 1 এর সমন্বয়ে কম্পিউটারে সংরক্ষিত হয়।

একটি একক bit কোন সংখ্যা, অক্ষর বা ক্যারেক্টার সংরক্ষণের জন্য যথেষ্ট নয়। তাই ৮টি bit-কে একত্রিত করে byte গঠন করা হয়। এবং এই byte-এর মধ্যে একটি একক সংখ্যা, অক্ষর বা ক্যারেক্টার সংরক্ষণ করা হয়।



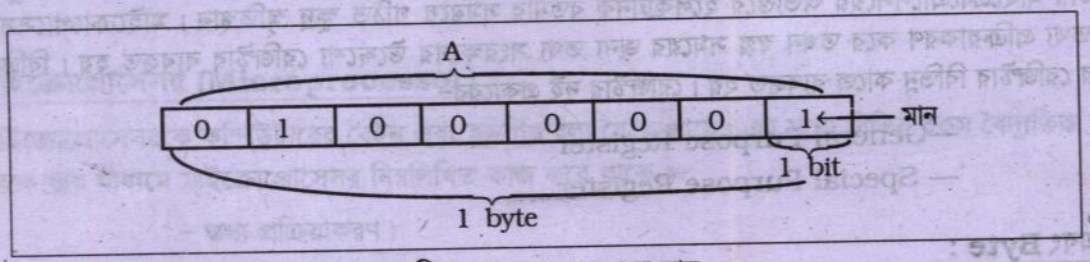
চিত্র ঃ bit এবং byte

1 byte = 8 bits
1 kilo byte = 1024 bytes
1 Mega byte = 1024 kilo bytes
1 Giga byte = 1024 Maga bytes
1 Tera byte = 1024 Giga bytes

ASCII ক্যারেক্টার কোড ঃ

কোন একটি অক্ষর, সংখ্যা বা ক্যারেক্টার কম্পিউটারের মাধ্যমে উপস্থাপন করতে হলে তা একটি byte-এর মাধ্যমে করা হয়। একটি byte-এর মধ্যে '0' এবং '1' কে বিভিন্নভাবে সজ্জিত করা যায়। যেমন ঃ 10101010 বা 1000001 ইত্যাদি। এভাবে 1 টি byte এর ৮টি বিটের মধ্যে '0' এবং '1' কে মোট 2^8 বা 256 ভাবে সজ্জিত করা যায়। এই 256 byte-এর প্রতিটি byte-কে কম্পিউটারে কোন একটি সংখ্যা, অক্ষর বা ক্যারেক্টারের মান হিসেবে নির্ধারণ করে দেয়া হয়েছে। এগুলোকে ASCII ক্যারেক্টার কোড বলা হয়। (ASCII = American Standard Code for Information Interchnge)। বইয়ের শেষে পরিশিষ্ট অংশে ASCII ক্যারেক্টার চার্ট দেয়া আছে।

কম্পিউটারের keyboard থেকে 'A' প্রেস করলে কম্পিউটার তাকে 01000001 হিসেবে সনাক্ত করে।



চিত্র ঃ 'A'-এর ASCII মান

Number System

আমরা জানি কম্পিউটার তার ভ্যন্তরীণ কাজের জন্য binary সংখ্যা (0, বা 1) ব্যবহার করে। তবে কিছু কিছু ক্ষেত্রে Hexadecimal সংখ্যাও ব্যবহৃত হয়। এখানে বিভিন্ন নাম্বার পদ্ধতি সম্পর্কে আলোচনা করা হয়েছে।

Decimal Number System

0-9 পর্যন্ত সংখ্যা নিয়ে এই পদ্ধতি গঠিত।

এটা হল base-10 পদ্ধতি। এই পদ্ধতিতে প্রতিটি সংখ্যার চেয়ে তার পরবর্তি বাদিকের সংখ্যাটি 10 গুণ বড়। যেমন ঃ 213 সংখ্যাটির 1 হল 3-এর চেয়ে $10^1 \times 1$ বেশি।

$2 = 2 \times 10^2 = 2 \times 100 = 200$
$1 = 1 \times 10^1 = 1 \times 10 = 10$
$3 = 3 \times 10^0 = 3 \times 1 = 3$
213

Binary System :

এটা হল base-2 পদ্ধতি। Binary পদ্ধতিতে কেবলমাত্র দুটি সংখ্যা ব্যবহৃত হয়, '0' এবং '1'। নিম্নের table-এ binary সংখ্যার সমমানের decimal সংখ্যার উদাহরণ দেয়া হল ঃ

Binary সংখ্যা	Decimal সংখ্যা
0	0
1	1
10	2
11	3
100	4

Binary সংখ্যা	Decimal সংখ্যা
101	5
110	6
111	7
1000	8
1001	9

Table : binary ও decimal সংখ্যা

Hexadecimal System :

এটা হল base-16 পদ্ধতি। এখানে 0-15 অর্থাৎ মোট 16টি সংখ্যা ব্যবহৃত হয়। এই পদ্ধতিতে 10 থেকে 15 পর্যন্ত সংখ্যাগুলো A থেকে F অক্ষরগুলো দ্বারা উপস্থাপিত হয়। Hexadecimal সংখ্যা কম্পিউটারের ক্ষেত্রে উপযোগী কারণ একটি hexa সংখ্যা লিখতে সর্বোচ্চ 4 bit এর প্রয়োজন এবং দু'টি hexa সংখ্যাকে 1 byte এর মধ্যে সংরক্ষণ করা যায়। এখানে binary সংখ্যার সমমানের Hexa সংখ্যার উদাহরণ দেয়া হল ঃ

Binary সংখ্যা	Hexadecimal সংখ্যা
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7

Binary সংখ্যা	Hexadecimal সংখ্যা
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Table : binary এবং hexadecimal সংখ্যা।

Exercise

1. bit এবং byte কি?
2. Word কি?
3. Memory কি?
4. প্রোগ্রামিং ল্যাংগুয়েজ কি কাজে ব্যবহৃত হয়?
5. ASCII কোড কি?
6. Binary Number এবং Hexa Number-এর পার্থক্য কি?
7. Gigabyte কি?
8. Register কি?
9. মাইক্রোপ্রোসেসরের কাজ কি?
10. 'C' এর ASCII মান কত?

1

Introducing C Language
(C ল্যাংগুয়েজ পরিচিতি)

- C-এর সাধারণ আলোচনা
- প্রোগ্রাম কোথায় লিখতে হয়
- কিভাবে প্রোগ্রাম লিখতে হয়?
- কম্পাইলার এর সাহায্য

C একটি কম্পিউটার প্রোগ্রামিং ল্যাংগুয়েজ। Dennis Ritchie নামক একজন প্রতিভাধর Programmar 1972 সালে যুক্তরাষ্ট্রের AT & T-এ Bell Laboratory'তে এটি তৈরি (Develop) করেন।

C ল্যাংগুয়েজ DEC PDP—11 মেশিনে UNIX অপারেটিং সিসটেম-এ তৈরি করা হয়। পরবর্তিকালে C একটি অতি জনপ্রিয় Programming Language-এ পরিণত হয় এবং বর্তমানে সারা বিশ্বের অনেক প্রয়োজনীয় Software C দ্বারা তৈরি করা হচ্ছে।

মূলত C একটি 'Standard' প্রোগ্রামিং ল্যাংগুয়েজ এবং বর্তমান বিশ্বের 90%-এর অধিক Operating System C দ্বারা তৈরি।

C-এর সংক্ষিপ্ত ইতিহাস (C History)

1960-এর দশকে বেশ কিছু Computer Prgramming Language তৈরি করা হয়েছিল। তখন এক একটি Programming Language-এক এক কাজে ব্যবহার করা হত। যেমন–

COBOL (Common Ordinary Business-Oriented Language) ব্যবহার করা হত ব্যবসারিক Software তৈরির উদ্দেশ্যে। FORTRAN (Formula Translator) ব্যবহার করা হত বৈজ্ঞানিক এবং Ergineering software তৈরির উদ্দেশ্যে।

তখন থেকে বিজ্ঞানীরা এমন একটি Programming Language-এর কথা ভাবতে থাকেন' যার দ্বারা সব ধরনের Software তৈরি করা সম্ভব হয়। এরই ফলশ্রুতিতে বিজ্ঞানীরা তৈরি করেন ALGOL 60 (Algorithmic Language) এবং এরপর Combined Programming Language (CPL). কিন্তু, CPL শেখা এবং ব্যবহার করা ছিল বেশ কঠিন তাই এটা তখন জনপ্রিয়তা পায়নি।

Combrige University-এর Martin Richards CPL-কে ভিত্তি করে 1967 সালে তৈরি করেন Basic Combined Pragramming Language (BCPL)। কিন্তু এটি ছিল মূলত Less Poworful too specific এবং typeless ল্যাংগুয়েজ।

এসময়েই যুক্তরাষ্ট্রের AT & T's Bell laboratory'র Ken Thompson তৈরি করেন B নামক Programming Language এবং এটি ছিল পূর্বের CPL-এর উন্নত সংস্করণ। কিন্তু, এটিও তেমন Powerful Language ছিল না। মূলত BCPL এবং B ছিল 'Typeless' ল্যাংগুয়েজ। Dennis Ritchie পরবর্তিতে B এবং BCPL অনুসরণ করেন এবং নিজে থেকে আরো কিছু Advanced Techninque ব্যবহার করে তৈরি করেন C। মূলত B এর সীমাবদ্ধতাগুলো দূর করার উদ্দেশ্যেই C-এর উৎপত্তি।

ANSI C

C-এর বহুমুখী ক্ষমতার কারণে এটি দ্রুত জনপ্রিয়তা পায় এবং বিভিন্ন প্রতিষ্ঠান তাদের নিজস্ব C ল্যাংগুয়েজ (Own version of C) তৈরি করতে থাকে। বিভিন্ন প্রতিষ্ঠানের তৈরিকৃত C Language-এ কিছু পার্থক্য থাকার কারণে Programmar রা নানারকম অসুবিধার সম্মুখীন হন।

এরই ফলশ্রুতিতে American National Standards Institute (ANSI) 1983 সালে একটি কমিটি গঠন করেন (ANSI Committee X 3J[11])। এ কমিটি C এর একটি আদর্শ মান নির্ধারণ করেন। সম্পূর্ণ ANSI Standard তৈরি শেষ হয় 1988 সালে।

বর্তমানে বিভিন্ন ধরনের C কম্পাইলার পাওয়া যায়। যেমন- Turbo C, Microsoft C, Borland C ইত্যাদি।

কিছু পার্থক্য ছাড়া সব আধুনিক C কম্পাইলার ANSI C standard অনুযায়ী তৈরি। এই বই এর Program গুলো, TC_3-তে লেখা এবং compile করা।

C কিভাবে উৎপত্তি হল তা ছক ১.১-এ দেখানো হল ঃ

বৎসর	প্রোগ্রামিং ল্যাংগুয়েজ	**Developer** (তৈরি) করেন	বর্ণনা
1960	ALGOL	আন্তর্জাতিক কমিটি	সাধারণ এবং কম দক্ষ
1963	CPL	ক্যামব্রিজ ইউনিভর্সিটি	এটা শেখা ছিল যেমন কঠিন তখনই বাস্তবায়ন রাখাও ছিল কঠিন।
1967	BCPL	মার্টিন রিচার্ড	এই প্রোগ্রামিং ল্যাংগুয়েজের মাধ্যমে কিছু নির্দিষ্ট সমস্যার সমাধান করা যেত।
1970	B	AT & T-এর কেন থমপসন	সমস্যা সমাধানে এর যথেষ্ট সীমাবদ্ধতা ছিল।
1972	C	AT & T-এর ডেনিস রিচি	পূর্বের B এবং BCPL এর সীমাবদ্ধতাগুলো এখানে কাটিয়ে ওঠা সম্ভব হয়।

চিত্র ঃ ১.১

C-এর কিছু উল্লেখযোগ্য বিশিষ্ট (Characteristics of C)

STRUCTURED LANGUAGE : C একটি structured Language কারণ এখানে বড় একটি Program কে ছোট ছোট অংশে (module or function). ভাগ করে লেখা যায়। প্রত্যেকটি ছোট ছোট অংশকে বলা হয় মডিউল (Module) বা ফাংশন (Function). এ ফাংশনগুলো নিয়েই একটি পূর্ণাঙ্গ Program গঠন করা যায়।

কোন নির্দিষ্ট Function একাধিকবার ব্যবহার করে এর ব্যবহার যোগ্যতা (Code Reusability) বাড়ানো যায়।

Structured Language যেমন C-এ Program সঠিকভাবে যাতে কার্য সম্পন্ন করতে পারে, সেজন্য Loop ব্যবহার করা যায়। যেমন– While, do-While, for ইত্যাদি।

Structured Language-এ goto ব্যবহার করাকে নিরুৎসাহিত করা হয় কারণ এতে Program তার সহজ গাঠনিক বৈশিষ্ট্য হারিয়ে ফেলে।

চিত্র ঃ ১-২ এ কিছু, Structured এবং nonstructured ল্যাংগুয়েজের উদাহরণ দেয়া হল–

Structured Language	Nonstructured Language
Pascal	Basic
C	COBOL
Ada	FORTRAN

চিত্র ঃ ১-১

C block-structured ল্যাংগুয়েজ নয় ঃ C ল্যাংগুয়েজে কোন function-এর ভিতর অন্য কোন ফাংশন লিখা (Define) যায় না। এ কারণে C block-struetured ল্যাংগুয়েজ নয়।

কমপার্টমেন্টালাইজেশন (Compartmentalisation) ঃ C ল্যাংগুয়েজে কোন Program- কে ছোট ছোট ভাগে ভাগ করে এর প্রত্যেক ভাগের মধ্যে data (local variable) এবং ইনস্ট্রাকশন আলাদাভাবে তৈরি করা যায়। এর ফলে Program-এর প্রত্যেকটা অংশ (function) স্বাধীন হয় এবং এক অংশের প্রভাব অন্য অংশে প্রতিফলিত হয় না। অর্থাৎ প্রোগ্রামের Side effect থাকে না।

General Purpose ল্যাংগুয়েজ ঃ C এমন একটি Programming Language যা দিয়ে যে কোন ধরনের সমস্যার সমাধান করা যায়।

COBOL দিয়ে ব্যবসায়ীক সমস্যা, FORTRAN দিয়ে বৈজ্ঞানিক সমস্যার সমাধান করা হতো। কিন্তু C দিয়ে যে কোন ধরনের software তৈরি করা যায়।

এখানে উল্লেখ্য যে, C প্রোগ্রামিং-এর মধ্যে Assembly program করা যায় এবং এতে করে Program-এর কর্মক্ষমতা অনেক বৃদ্ধি পায়।

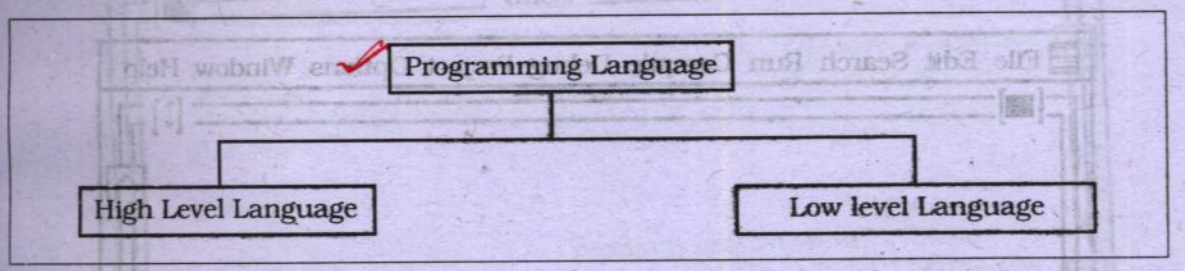
Portable ল্যাংগুয়েজ ঃ C একটি Portable ল্যাগুয়েজ কারণ এক Computer-এ (যেমন IBM PC) লিখিত C Program অন্য কোম্পানির তৈরি Computer-এ (যেমন EDC VAX) কম্পাইল এবং Run করা যায়।

C হতে C++ এবং Java ; C হল C++ এবং Java এর মূল ভিত্তি। অর্থাৎ C-এর অনেক গুণ এবং বৈশিষ্ট্য C++ এবং Java- তে ব্যবহার করা হয়েছে।

তাই বলা যায়, C শিখলে C++ এবং Java-এর অনেক কিছু শেখা হয়ে যায়।

C একটি Middle Level ল্যাংগুয়েজ

সমস্ত প্রোগ্রামিং ল্যাংগুয়েজগুলো দুই বিভক্ত। নিচে তা দেখানো হল–



চিত্র ঃ ১-২

High Level ল্যাংগুয়েজে ঃ এ প্রোগ্রামিং ল্যাংগুয়েজগুলো English like ল্যাংগুয়েজ। এতে প্রোগ্রামিং করা সহজ। যেমন— BASIC, PASCAL, COBOL ইত্যাদি। এদের Programming efficiency বেশি।

Low Level ল্যাংগুয়েজ ঃ এ ধরনের ল্যাংগুয়েজগুলোর machine efficiency বেশি। যেমন— Assembly, machine Language। এখানে মেশিনের ভাষায় প্রোগ্রামিং করতে হয়।

C এমনভাবে Design করা হয় যাতে করে এর Programming efficiency এবং machine efficiency উভয়ই অটুট থাকে। তাই C-কে Middle Level প্রোগ্রামিং ল্যাংগুয়েজও বলা হয়।

C Program কোথায় লিখতে হয়?

এই বইয়ের প্রতিটি Program Turbo C (ভার্সন TC_3) তে লেখা। TC_3 কম্পাইলার এর সাথে একটি editer আছে। এই editor-এ Program লিখতে হয় key board-এর মাধ্যমে এবং এখান থেকেই প্রোগ্রাম সংশোধন, এবং সংরক্ষণ (Save) করা যায়।

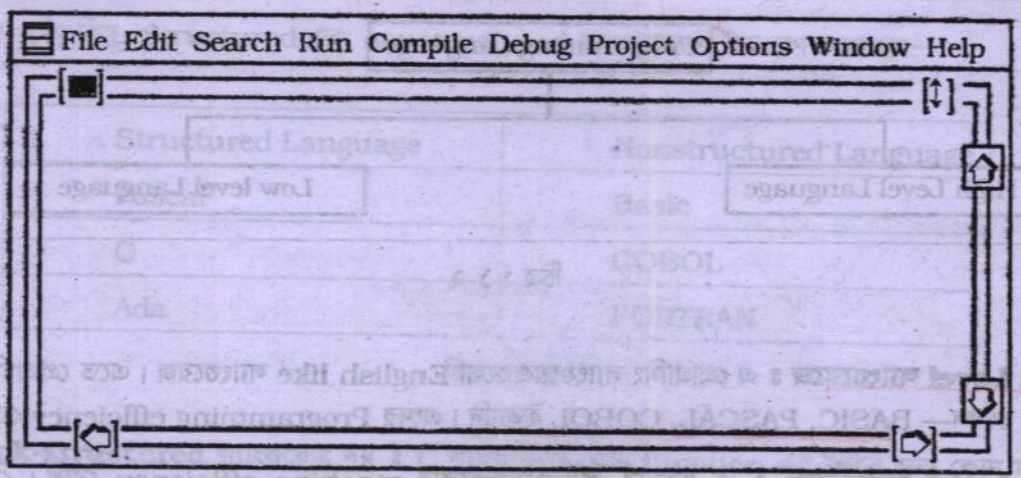
Dos command line-এ TC_3 editor খুলতে হলে নিম্নোক্ত command লিখতে হবে। (ধরা হল TC_3 আছে C:\ drive-এ)

C:\> cd TC3 (Keyboard থেকে Enter প্রেস করুন)

C:\TC3>cd bin (Key board থেকে Entar প্রেস করুন)

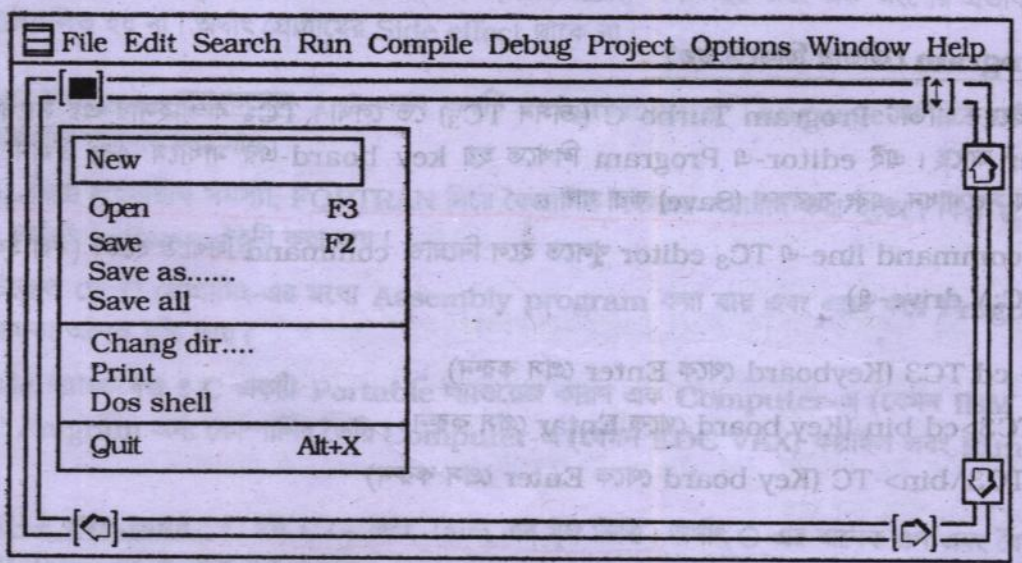
C:\ TC3\bin>TC (Key board থেকে Enter প্রেস করুন)

এখন মনিটরে নিম্নোক্ত চিত্রের মত window দেখা যাবে। এটা হল Turbo C-এর IDE (Integrated Development Environment)



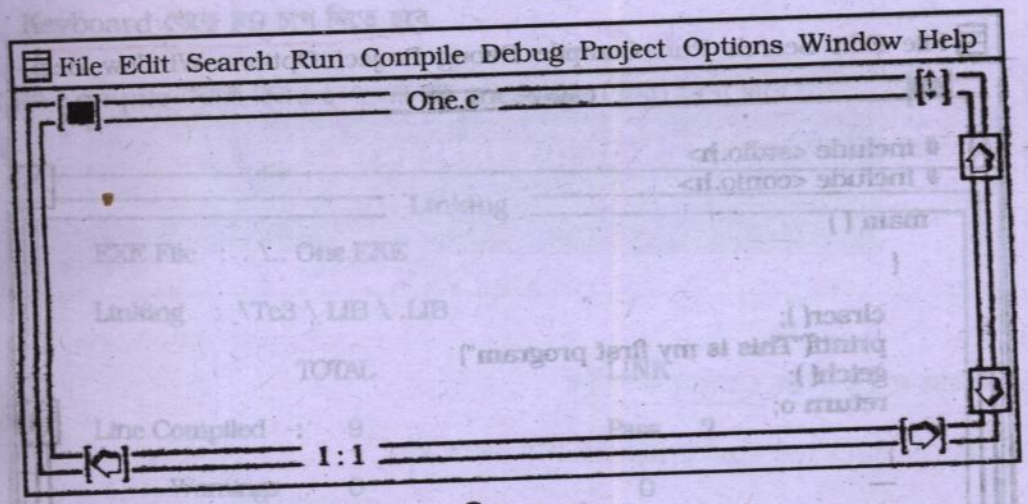
চিত্র ঃ ১-৩

এবার File মেনু থেকে New সিলেক্ট করতে হবে। (চিত্র ১-৪)



চিত্র ঃ ১-৪

এতে করে চিত্র ১-৫ এর মত Window open হবে। এখানেই program লিখতে হবে।



চিত্র ঃ ১-৫

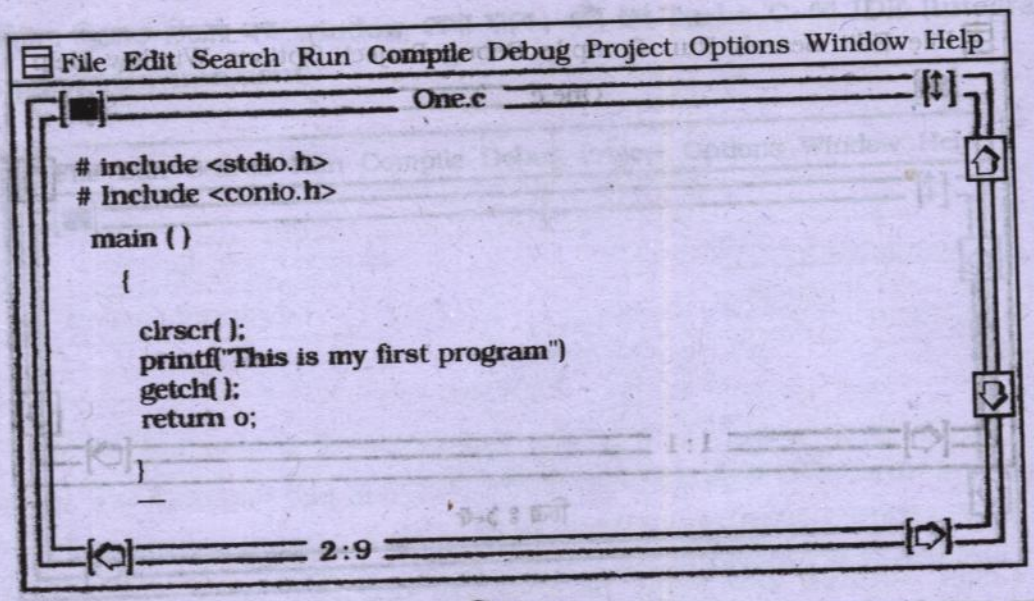
প্রোগ্রাম লেখা (write a program)

এবার নিম্নোক্ত প্রোগ্রামটি এই window-তে টাইপ পরতে হবে।

Program–1-1 **One.c**

```
# include<stdio.h>
# include <conio.h>
main ( )
{
clrscr ( );
printf ("This is my first program");
getch ( );
return o;
}
```

এই প্রোগ্রামটি লেখার পর window টি চিত্র ১-৬ এর অনুরূপ হবে।



চিত্র ঃ ১-৬

Program সমন্ধে বিশ্লেষণ পরবর্তি অধ্যায় থেকে করা হবে। এখানে আমরা জানবো Program কোথায়, কিভাবে লেখা হয়, Save (সংরক্ষণ) করা হয় এবং Compile ও Run করা হয়।

প্রোগ্রাম সংরক্ষণ (Save the progaom)

প্রোগ্রাম Type করার পর তা disk-এ সংরক্ষণ করার জন্য File→ save সিলেক্ট করতে হবে। (F2 প্রেস করলেও হবে) এখন Save File As উইন্ডো open হবে। এখানে একটি অস্থায়ী নাম (NONAMEO.O CPP) দেখা যাবে। আমরা One.c নামে Program টি C:/TC3/Bin ড্রাইভে save করবো।

Program Compile করা (Compile the Program)

Edit window-তে লেখা Program টি হল Source code বা Source File. এই Source code কে compile করলে, Program-এ কোন ভুল-থাকলে তা ধরা পরে এবং Error message দেখা যায়।

Program-এর code ঠিক থাকলে Compilation এর ফলে object File তৈরি হয়।

Turbo C-তে Program দুইভাবে Compile করা যায়।

1. menu থেকে ঃ

   Compile→ Complie

2. Hotkey ঃ

Keyboard থেকে F9 চাপ দিতে হবে

Pragram Compile করলে চিত্র ঃ ১-৭ এর মত message (তথ্য) দেখা যাবে।

```
                    Linking
EXE File  : ..\.. One.EXE
Linking   : \Tc3 \ LIB \ .LIB

                TOTAL          LINK
Line Compiled  :  9            Pass  2
Warnings  :  0                    0
Errors    :  0                    0
Available memory  :  30k
Success                :  Press any key
```

চিত্র ঃ ১-৭ এর মত

Program Run করা (Run the program)

কম্পিউটার-কে কোন ইনস্ট্রাকশন দিতে হলে তা মেশিন ল্যাংগুয়েজের মাধ্যমে দিতে হয়। Program Run করলে প্রোগ্রামের executable ফাইল তৈরি হয় এবং প্রোগ্রামের out put দেখা যায়। এই executable ফাইলটি মেশিন ল্যাংগুয়েজ-এর সমন্বয়ে তৈরি হয়। Turbo C তে Pragram দুইভাবে Run করা যায়–

1. menu থেকে ঃ

Run→ Run

2. Hotkey ঃ

Ctrl→ F9 (Control এবং F9 একসাথে চাপ দিতে হবে।)

One. C প্রোগ্রামটি Run করলে নিম্নোক্ত Output দেখা যাবে–

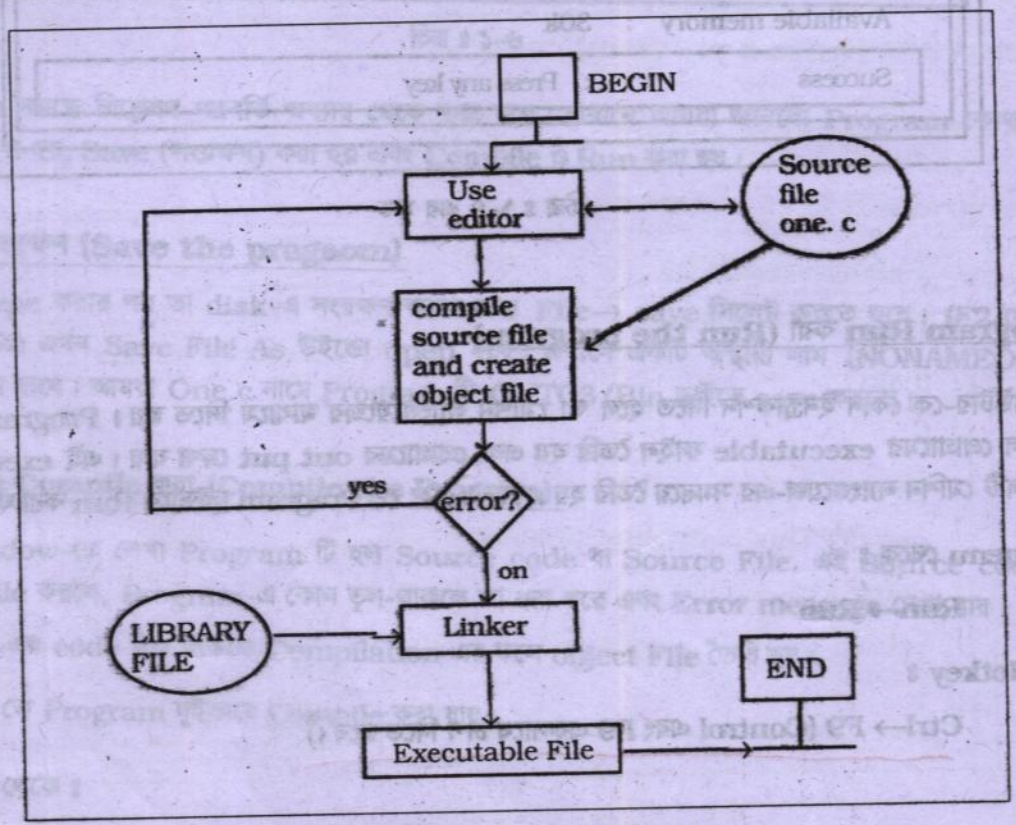
```
This is my first program
```

চিত্র ঃ ১-৮

Dos কমাণ্ড লাইন থেকে প্রোগ্রাম কম্পাইল

C:\> TC3> bin> tcc One.c [Enter প্রেস করতে হবে]

Executable File তৈরির ফ্লোচার্ট



চিত্র ঃ Executable File তৈরির প্রক্রিয়া।

প্রোগ্রাম বোঝা ঃ

এখানে C প্রোগ্রামিং ল্যাংগুয়েজ সম্বন্ধে সাধারণ ধারণা দেয়া হয়েছে। এখানে বেশ কয়েকটি Sample প্রোগ্রাম Line by Line বিশ্লেষণ করে দেয়া হয়েছে। আশা করা যায় এত শিক্ষার্থীরা উপকৃত হবেন। নতুন শিক্ষার্থীরা প্রোগ্রামগুলো মনোযোগ দিয়ে পড়লে পরবর্তি অধ্যায়গুলো বুঝতে সুবিধা হবে বলে আমি মনে করি।

নিচের message. C প্রোগ্রামটি কম্পিউটার এর পর্দায় একটি Sentence লিখবে।

Program–2-1 **message. c**

```
# include <stdio.h>
main ( )
  {
  print f ("My First program");
  }
```

Output My First Program

NOTE

এ Program-এর Output দেখতে হলে F9 দিয়ে Compile করে Ctrl–F9 দিয়ে রান করিয়ে Att–F5 চাপ দিতে হবে।

message.C প্রোগ্রাম বিশ্লেষণ (Dissection of message.C)

- # include <stdio.h>

এটা হল Preprocessor directive। এগুলো C compiler-এর সাথে যুক্ত থাকে। Preprocessor directive নাম্বার চিহ্ন # হতে শুরু হয়।

Stdio.h হল Standard Input/output header File. printf ( ) ফাংশনটির বর্ণনা এই ফাইলে লেখা থাকে। কোন প্রোগ্রামে printf ( ) লিখলে # include <stdio.h> ও লিখতে হবে।

inclide লিখে < >-এর ভিতর যে header File-এর নাম লেখা হবে, compiler সেই হেডার ফাইল Copy করে বর্তমান প্রোগ্রামে নিয়ে আসবে। Header File C system-এর সাথে যুক্ত থাকে।

include <stdio.h> এই line টি না লিখলে compiler error দেখাবে এবং printf ( ) ফাংশনটি কাজ করবে না।

■ **main ( )**

যে অংশটি প্রতিটি C প্রোগ্রামে অত্যবশ্যকীয় তা হল main ( ) ফাংশন। main-এর পরে ( )-কে বলে Parentheses. Parentheses অবশ্যই দিতে হবে। প্রোগ্রাম execution এই main ( ) ফাংশন থেকেই শুরু হয়। একটি C প্রোগ্রামে অনেক ফাংশন থাকতে পারে। তবে প্রতিটি ফাংশনই এই main ( ) ফাংশন এর মধ্যে ব্যবহার (call) করতে হবে।

■ **{**

এটাকে বলা হয় left brace। এটা দ্বারা প্রোগ্রামের শুরু (Begin) বোঝায়। একাধিক প্রোগ্রাম Line একত্রে সমন্বিত করার উদ্দেশ্যেও brace ব্যবহার করা হয়।

■ **printf ( )**

C system-এর standard Library-তে অনেক ফাংশন থাকে। printf ( ) হল stdio.h হেডার ফাইল এর এমনই একটি ফাংশন যা Computer-এর পর্দায় কিছু print করার জন্য ব্যবহৃত হয়।

■ **printf ("My First Program");**

(")-কে বলা হয় double quotes, printf-এর (" ..... ") এর মধ্যে যা লেখা হবে printf তাই Computer-এর পর্দায় print করবে।

C হল case sensative ল্যাংগুয়েজ। এখানে A ও a হল দুটি ভিন্ন character তাই printf-কে Printf বা main-কে Main লেখা যাবে না।

■ **;**

এটা হল semicolon. প্রতিটি instruction এর শেষে semicolon দিতে হবে। Semicolon সহ প্রতিটি instruction কে বলে statement।

■ **}**

এটা হল Right brace। কোন Program-এ প্রতিটি left brace-এর জন্য একটি করে Right brace অবশ্যই থাকতে হবে। এই program-এ } দ্বারা প্রোগ্রামের শেষ (end) বোঝানো হয়েছে।

নিচের newline. C প্রোগ্রামটি এবং এর output শিক্ষার্থীরা লক্ষ্য করুন।

Program–2-2 **newline. c**

```
# include <stdio.h>
# include < conio.h>
main ( )
 {
  clrscr ( );
  printf (" 1st Line \n 2nd line \n 3rd line");
  getch ( ) ;
 }
```

new line character

Output

1st Line
2 nd Line
3 rd Line

প্রোগ্রাম বিশ্লেষণ (Dissection of Program)

- **# include <conio.h>**

conio.h একটি headr File। clrscr ( ) এবং getch ( ) এর মত Screen handling ফাংশনগুলো conio. h-এর তৈরি (define) করা থাকে।

- **clrscr ( );**

এটি একটি ফাংশন যা conio.h হেডার ফাইল এ তৈরিকৃত (Define) থাকে। কম্পিউটারের Screen (পর্দা) clear করাই হল এর কাজ।

■ **\n**

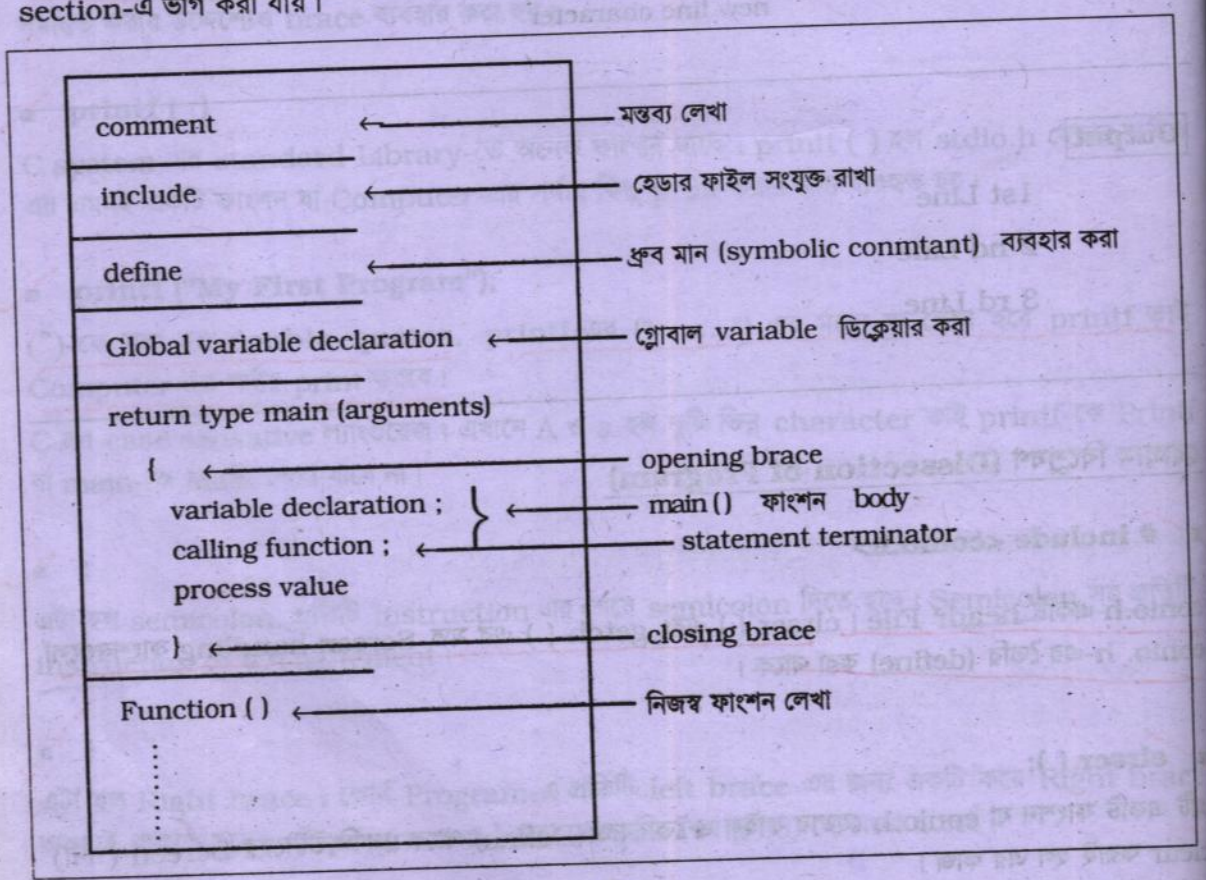
প্রোগ্রাম-এর printf ফাংশনের মধ্যে \n দুইবার ব্যবহার করা হয়েছে। এটা হল Newline character। \n-এর পর যা লেখা থাকবে তা পরবর্তি Line (Newline) থেকে Print হবে। শিক্ষার্থীরা program-এর output লক্ষ্য করুন। 1st line লেখার পর 2nd Line দ্বিতীয় লাইনে এবং 3rd Line তৃতীয় লাইনে লেখা হয়েছে।

■ **getch ( )**

getch ফাংশনটি পাওয়া যায় conio. h হেডার ফাইলে। এই ফাংশন Keyboard থেকে input নেয় তা screen- এ তা প্রদর্শিত করে না। এই প্রোগ্রাম-এ getch ( ) না থাকলে প্রোগ্রামের output দেখার জন্য Alt-F5 Key প্রেস করতে হবে।

একটি C প্রোগ্রামের মৌলিক গঠন (STRUCTURE OF A C PROGRAM)

একটি C প্রোগ্রাম-এ main ( ) ফাংশনসহ এক বা একাধিক ফাংশন থাকতে পারে। প্রতিটি ফাংশন কোন নির্দিষ্ট কাজ (specific task) করার উদ্দেশ্যে লেখা হয়। এই ফাংশন সহকারে C প্রগ্রামকে অনেকগুলো section-এ ভাগ করা যায়।



চিত্র ঃ

প্রোগ্রামের ব্যকরণগত ভুল (Syntax error)

আমরা জানি প্রতিটি ভাষার নিজস্ব ব্যকরণ আছে। যেমন–

ইংরেজি ভাষা লিখতে গেলে নির্দিষ্ট Grammar অনুসরণ করতে হয়।

তেমনই প্রতিটি প্রোগ্রামিং ল্যাংগুয়েজ লেখার নিজস্ব নিয়ম-কানুন রয়েছে। একে বলা হয়–

Program syntax। প্রোগ্রাম Syntax- এ কোন ভুল থাকলে তাকে বলা হয়। Syntax error.

কম্পাইলেশন ইরর (compilation error)

প্রোগ্রাম লিখে compile করলে, প্রোগ্রামে যদি কোন ভুল থাকে তবে compiler error দেখায়। একে বলে compilation error বা compile time error. কোন প্রোগ্রামে syntax error থাকলে program compilation fail (ব্যর্থ) হয়।

Welcome. C প্রোগ্রামটিতে syntax error আছে। তাই এই প্রোগ্রামটি compile করলে compiler error দেখায়–

	Program	Welcome. c
Line 1 :	# include < stdio.h>	
Line 2 :	# include <conio.h>	
Line 3 :	main ( )	
Line 4 :	{	
Line 5 :	printf ("Welcome to programming world);	
Line 6 :	getch ( )	
Line 7 :	}	

প্রোগ্রামটি compile করলে compiler নিম্নোক্ত message দেখায়–

Compiling WELCOME. C

Error Welcome. c 5 : Unterminated string or character constant

Error Welcome.c 6 : Function call missing

উপরের প্রথম error message-টির তিনটি অংশ আছে। এগুলো হল–

Welcome.c	যে প্রোগ্রামে ভুল হয়েছে সেই প্রোগ্রাম ফাইল এর নাম
5	প্রোগ্রামের যে লাইনে ভুল হয়েছে তার Line mumber
Unterminated string or character constant	error (ভুল) এর বর্ণনা

Line 5-এ আমরা printf ( ) ফাংশন এর মধ্যে opening double quote দিয়েছি কিন্তু closing double quote দেইনি ঃ

closing double quote

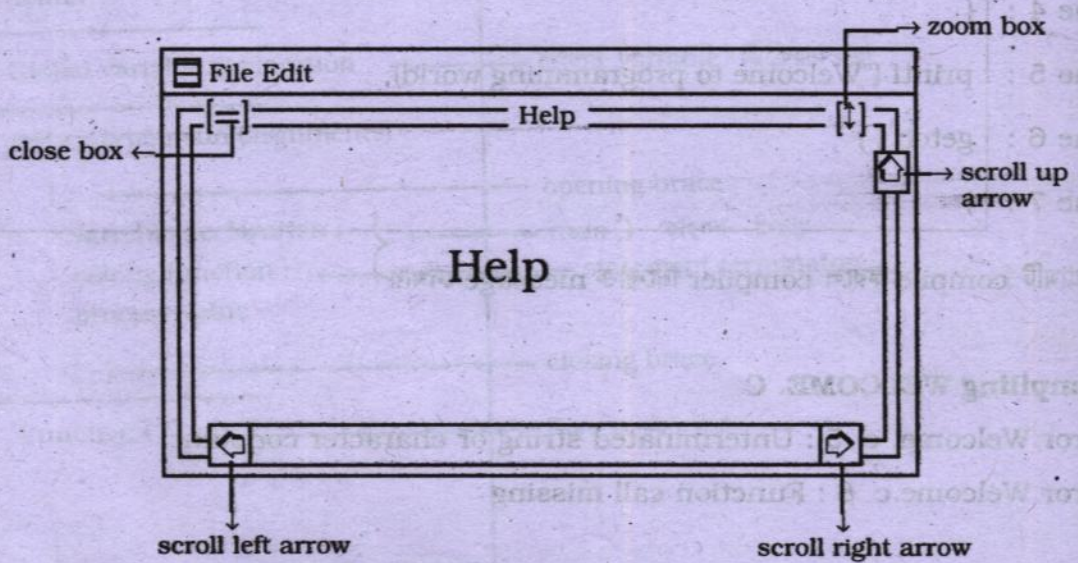
printf (" Welcome to programming world closing dauble quote");

Line 6-এ কোন ভুল হয়নি। এটা Line 5-এর কারণে দেখানো হচ্ছে। Line 5 ঠিক করলে এটাও ঠিক হবে।

HELP

Turbo C-এর কোন Library function বা Key word সম্পর্কে জানতে হলে সেই Function বা Kayword-এর শুরুতে cursor নিয়ে ctrl-F1 একসাথে প্রেস করতে হবে।

যেমন– printf সম্পর্কে জানতে হলে- p এর নিচে কারসর (cursor) এনে ctrl–F1 প্রেস করতে হবে। ফলে কম্পিউটার sereen-এ printf ( ) সম্পর্কিত তথ্য দেখা যাবে। নিচে printf ( ) এর help screen-এর গঠন দেখানো হল–



Example 1

centre. c প্রোগ্রামটি Run করলে কম্পিউটার Screen-এর ঠিক মাঝখানে centre লেখাটা দেখা যাবে।

Program–2-3 **centre. c**

```
# include <stdio.h>
# include < conio.h>
main ( )
   {
   gotoxy (39,12);
   printf ( "centre");
   getch ( );
   }
```

Example 2

beep. c প্রোগ্রামটি Run করলে কম্পিউটারের internal speaker-এ beep শব্দ হবে।

Program–2-4 **beep. c**

```
# include < stdio.h>
# include < conio.h>
main ( )
   {
      printf (" \a");
      getch ( );
   }
```

Example 3

নিচের প্রোগ্রামটি রান করলে screen-এ শুধু একটি character (A) দেখা যাবে। একাধিক character-এর সমন্বয়ে গঠিত একটি word-কে Screen-এ show করতে হলে printf-এর মধ্যে duble quote (" ") ব্যবহার করতে হয়। কোন একটি character (যেমন, A, B, & ইত্যাদি) screen-এ show করতে হলে printf-এর মধ্যে single quote (' ') ব্যবহার করলেও হয়।

Program **char. c**

```
# include <stdio.h>
# include < conio.h>
main ( )
   {
   printf ( 'A');
   getch ( );
   }
```

Q&A :

1. একই Compiler দিয়ে c ও c++ প্রোগ্রাম Compile ও Link করা যায় কি?

উত্তর ঃ হ্যাঁ, একই Compiler দিয়ে c ও c++ প্রোগ্রাম compile ও Link করা যায়। কোন c প্রোগ্রাম ফাইলের extension যদি .c হয় তবে compiler একে একটি c প্রোগ্রাম হিসেবে গণ্য করে। কিন্তু extonsion যদি .cpp হয় তবে compiler তা c++ প্রোগ্রাম হিসেবে Compile ও Link করে।

2. Object ফাইল কখন তৈরি হয় এবং এর extension কি?

উত্তর ঃ প্রোগ্রাম Compile করলে object file তৈরি হয়। Object ফাইলের extension হল .OBJ।

3. প্রোগ্রাম Run করলে EXE ফাইল কোথায় তৈরি হয়?

উত্তর ঃ EXE ফাইল কোথায় তৈরি হয় তা বুঝতে হলে প্রথমে মেনু থেকে Option-এ যেয়ে Directories সিলেক্ট করতে হবে এবং এখানে Output Directory- তে যদি আমরা c:\TC3/Bin লিখি তবে EXE ফাইল Bin ডিরেক্টরিতে তৈরি হয়ে থাকবে।

4. নিচের প্রোগ্রামের ভুল কোথায়?

```
# include <stdio.h>;
   main ( )
   { }
```

উত্তর ঃ # include <.....>-এর পর Semicolon (;) দেয়া যায় না।

5. নিচের প্রোগ্রামটি ঠিক কিংবা বেঠিক?

```
# include <stdio.h>
main ( )
   {
      printf ("Calling main");
      main ( ),
   }
```

উত্তর ঃ main ( ) ফাংশন-এর মধ্যে main ( ) লেখা যায় না।

6. True or False নির্ণয় ঃ

(a) প্রতিটি c প্রোগ্রামে main ( ) ফাংশন থাকতে হবে।

(b) প্রোগ্রাম execution শুরু হয় main ( ) থেকে।

(c) C ল্যাংগুয়েজে A ও a দুটি ভিন্ন character।

(d) \n হল new line character

(e) compile error হলে প্রোগ্রাম Run করা যায়।

উত্তর ঃ (a) True (b) True (c) True (d) True (e) False

7. Dos Prompt থেকে সরাসরি কিভাবে C প্রোগ্রাম লেখার editor Open করা যায়?

উত্তর ঃ c :\> cd TC3 (Enter প্রেস করুন)

c:\TC3>cd bin (Enter প্রেস করুন)

c:\TC3\bin> TC Example.c (Enter প্রেস করুন)

উপরোক্ত commnd-এর মাধ্যমে Dos থেকে সরাসরি প্রোগ্রাম লেখার editor খোলা যায়।

Exercise

1. Header File কি?
2. Compiler -এর কাজ কি?
3. Machine ল্যাংগুয়েজ কি?
4. C-কে কেন Middle Level ল্যাংগুয়েজ বলা হয়?

2

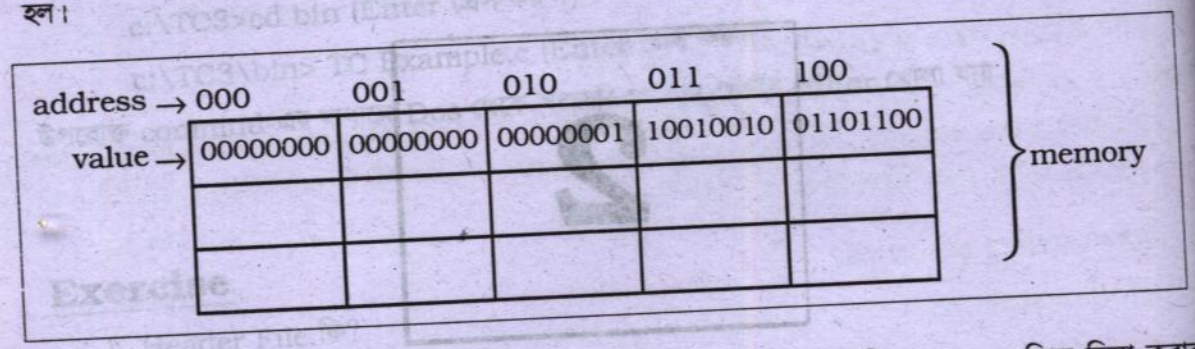
Variable And Data type
(ভেরিয়েবল এবং ডাটা টাইপ)

- ভেরিয়েবল
- ডাটা টাইপ
- ইনপুট / আউটপুট

ভেরিয়েবল (Variable)

ভেরিয়েবল হল কম্পিউটার এর মেমোরীতে সংরক্ষিত স্থান যেখানে তথ্য (data) রাখা হয় এবং মেমোরীর এই স্থানের নির্দিষ্ট নাম দেয়া হয়।

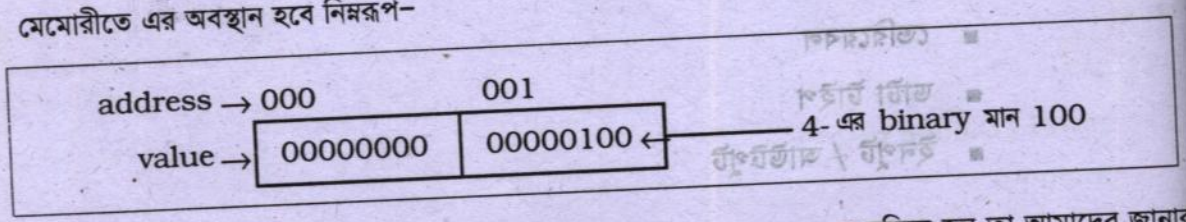
কম্পিউটারের মেমোরীতে প্রতিটি byte ক্রমানুসারে সজ্জিত থাকে এবং প্রতিটি byte-এর নির্দিষ্ট address থাকে। এই addrss 0 থেকে শুরু হয় এবং ক্রমানুসারে প্রতি বাইটের addrss বাড়তে থাকে। এই address দ্বারা প্রতিটি byte আলাদাভাবে সনাক্ত করা যায়। নিচের চিত্রে মেমোরী এবং address দেখানো হল।



C প্রোগ্রামিং-এর মাধ্যমে মেমোরীতে কোন তথ্য রাখতে হলে আমাদের মেমোরী address নিয়ে চিন্তা করার প্রয়োজন নেই। ভেরিয়েবল ব্যবহার করলে C কম্পাইলার সিদ্ধান্ত নিবে কোন address-এ কোন data (তথ্য) রাখতে হবে। মেমোরীর একই স্থানে বিভিন্ন সময় ভিন্ন ভিন্ন তথ্য (data) রাখার জন্য ভেরিয়েবল ব্যবহার করা হয়। যেমন, ধরি x একটি ভেরিয়েবল এবং এর মান দেয়া হল 4.

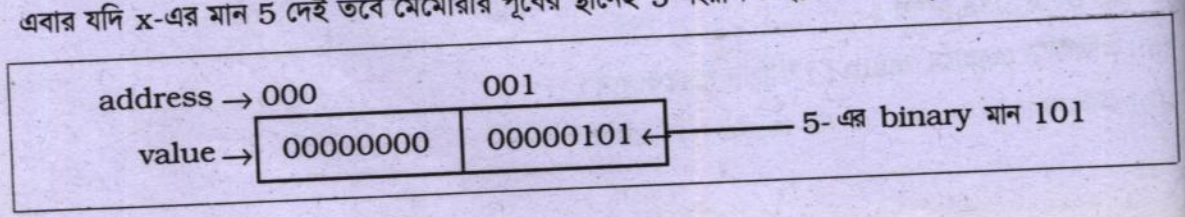
```
x = 4
```

মেমোরীতে এর অবস্থান হবে নিম্নরূপ–



যেহেতু আমরা ভেরিয়েবল ব্যবহার করেছি, 4 মেমোরীর কোন address-এ সংরক্ষিত হল তা আমাদের জানার দরকার নেই। Compiler এটা পরিচালিত (Manage) করবে।

এবার যদি x-এর মান 5 দেই তবে মেমোরীর পূর্বের স্থানেই 5 সংরক্ষিত হবে এবং 4 মুছে যাবে।



ভেরিয়েবল গঠনের নিয়ম ঃ ভেরিয়েবল এর নাম প্রোগ্রামাররা ইচ্ছে মত দিতে পারেন। তবে ভেরিয়েবল এর নাম দেবার কিছু নিয়ম (Rule) আছে।

C প্রোগ্রামে ভেরিয়েবল ব্যবহার করতে হলে এই নিয়মগুলো জানতে হবে ঃ

- ভেরিয়েবল এর নাম অক্ষর (A....Z, a.....z), সংখ্যা (0.....9) বা Underscore (_)-এর সমন্বয়ে গঠিত হতে পারে।
- ভেরিয়েবল এর নামের প্রথমেই অক্ষর (a....z, A...Z) ব্যবহার করতে হবে। কোন কোন Compiler প্রথমে underscore (_) ব্যবহারকে support করে।
- ANSI আদর্শ অনুসারে ভেরিয়েবল এর নাম সর্বোচ্চ 31 ক্যারেক্টার হতে হবে। তবে Compiler ভেদে এর ভিন্নতা দেখা যায়।
- ভেরিয়েবল এর নাম অবশ্যই C keyword-এর অনুরূপ হবে না।
- C তে Uppercase (A....Z) এবং Lowercase letter ভিন্ন (a....z) অক্ষর হিসেবে ধরা হয়। তাই sum এবং sum এখানে দুটি ভিন্ন ভেরিয়েবল
- ভেরিয়েবল এর নাম লেখার সময় এর মধ্যে কোন খালি স্থান (White space) রাখা যাবে না।

সঠিক (Valid) ভেরিয়েবল

Tareq

B1

EMDAD

S_A_Z_Z_A_D

ভুল (Invalid) ভেরিয়েবল

im ran

012

[ABC]

(Suman)

%T.

1st

milton

নিম্নে কারণসহ কিছু গ্রহণযোগ্য ও অগ্রহণযোগ্য ভেরিয়েবল এর উদাহরণ দেয়া হল ঃ

ভেরিয়েবল	ঠিক/বেঠিক	বর্ণনা
my_age	√	
distance	√	
_2000_	√	ঠিক তবে ব্যবহার না করলেই ভাল
Address #	×	অগ্রহণযোগ্য ক্যারেক্টার #
int	×	C keyword
ari f	×	ফাকা স্থান গ্রহণযোগ্য নয়।

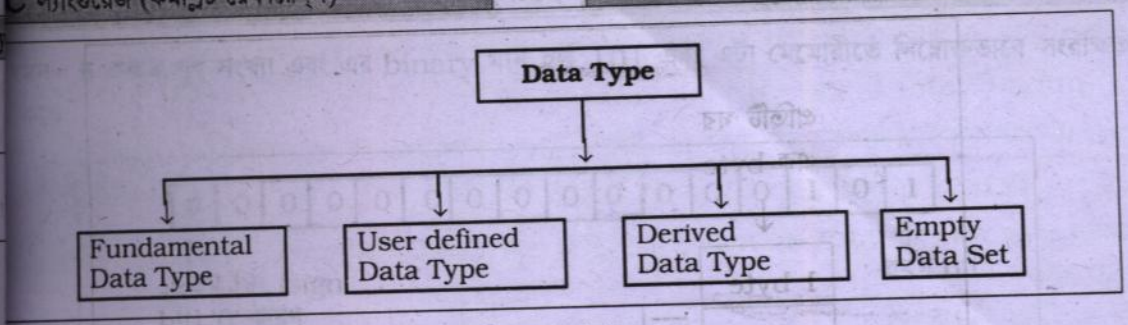
NOTE

- কিছু character আছে যা ভেরিয়েবল এর নামে ব্যবহার করা যায় না। যেমন– #, (, {, [ইত্যাদি।
- যে উদ্দেশ্যে ভেরিয়েবল ব্যবহার করা হবে, ভেরিয়েবল এর নাম তার সাথে সামঞ্জস্যপূর্ণ হলে ভাল হয়। যেমন– প্রোগ্রামে বয়স (age) নিয়ে কাজ করা হলে ভেরিয়েবল এর নাম age দিলে সকলের বুঝতে সুবিধে হবে।
- ভেরিয়েবল এর প্রথম অক্ষর Uppercase দেয়া প্রোগ্রামের একটি style। একে বলে camel notation. যেমন– phone_no না দিয়ে Phone_no দেয়া যেতে পারে।

ডাটা টাইপ (Data Type in C)

C প্রোগ্রামিং ল্যাংগুয়েজে অনেক ধরনের ডাটা টাইপ আছে। এই ধরনের ডাটা টাইপগুলোর মাধ্যমে কম্পিউটারের মেমোরীতে বিভিন্ন ধরনের ডাটা (10,10.10, A, ABC, ইত্যাদি) রাখা ও ব্যবহার করা যায়।

ANSI C চার ধরনের ডাটা টাইপ-কে স্বীকৃতি দেয়–

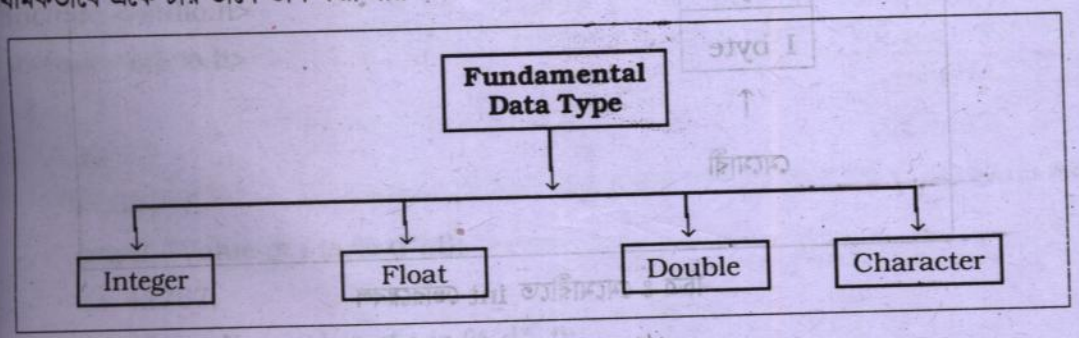


চিত্র ঃ ডাটা টাইপ

এই অধ্যায়ে Fundamental Data Type নিয়ে আলোচনা করা হয়েছে।

মৌলিক ডাটা টাইপ (Fundamantal data Type) :

প্রাথমিকভাবে একে চার ভাগে ভাগ করা যায়–



চিত্র ঃ ফান্ডামেনাল ডাটা টাইপ

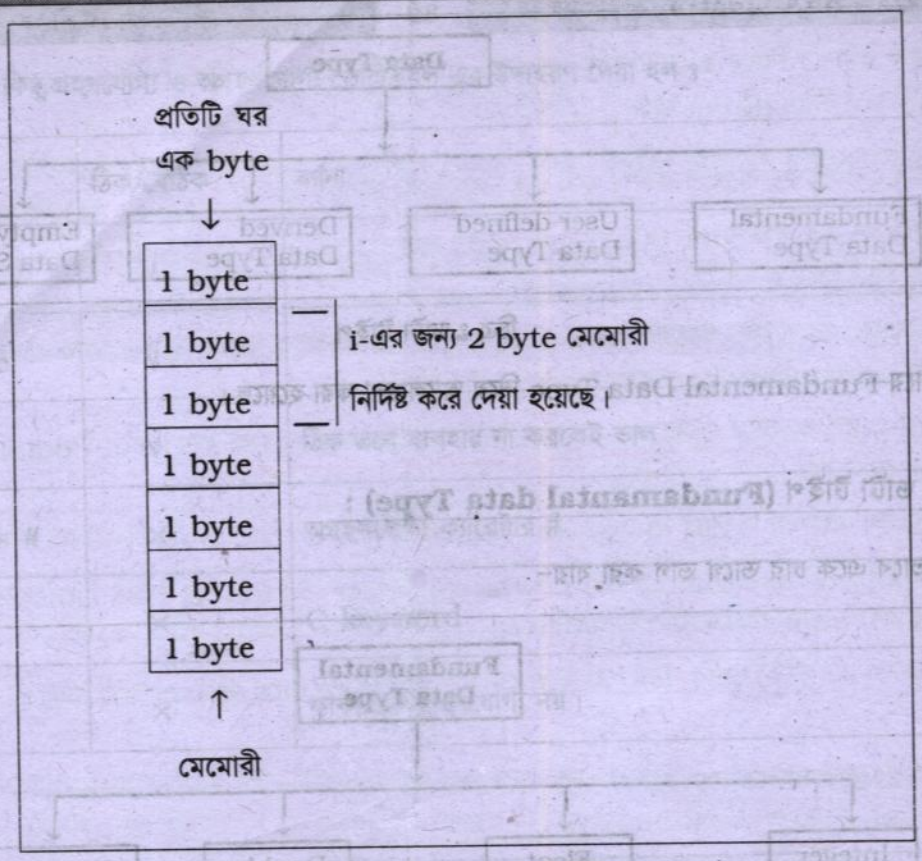
int ভেরিয়েবল

Integer-এর অর্থ হল পূর্ণ সংখ্যা অর্থাৎ ঃ ১০,২০,999,১১১২ ইত্যাদি। এই Integer-এর সংক্ষিপ্ত নাম হল int.

C প্রোগ্রামে পূর্ণ সংখ্যা ব্যবহার করার জন্য int টাইপ ভেরিয়েবল ব্যবহৃত হয়। কোন ভেরিয়েবলকে int টাইপ ঘোষণা (declare) করলে C কম্পাইলার মেমোরীতে ঐ ভেরিয়েবল-এর জন্য 2 byte স্থান নির্দিষ্ট করে। যেমন–

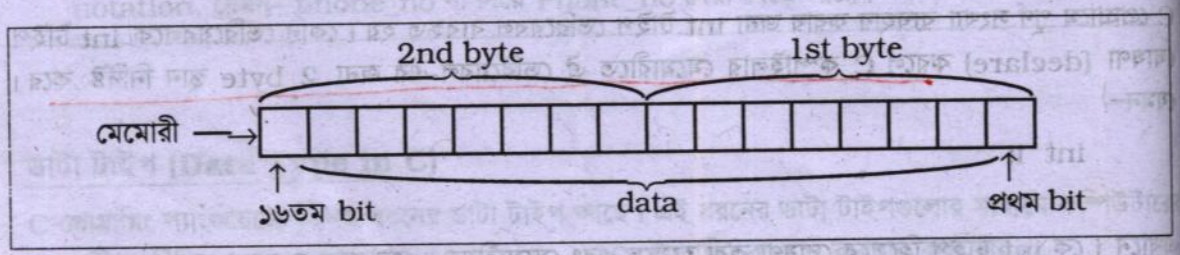
```
int i;
```

এখানে i কে int টাইপ হিসেবে ঘোষণা করা হয়েছে এবং মেমোরীতে i-এর জন্য 2 byte (16 bit) জায়গা নেয়া হয়েছে।



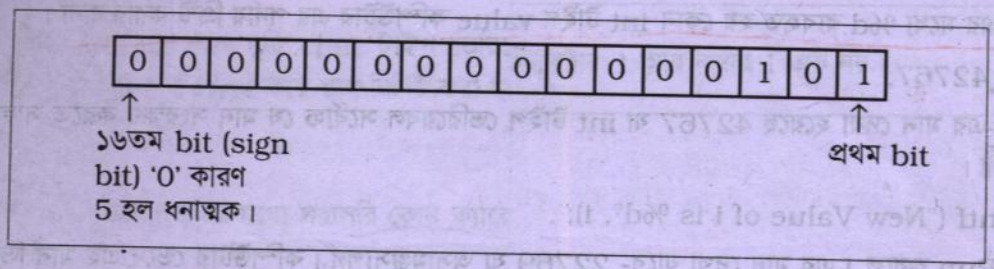
চিত্র ঃ মেমোরীতে int ভেরিয়েবল

কোন ভেরিয়েবলকে int টাইপ ঘোষণা করলে তার মধ্যে সর্বোচ্চ +32767 ও সর্বনিম্ন –32768 ($+2^{15}-1$ থেকে -2^{15}) মানের সংখ্যা রাখা যায়। এর কারণ হল int টাইপ ভেরিয়েবল 2 byte অর্থাৎ ১৬ bit মেমোরী দখল করে। এর মধ্যে প্রথম থেকে ১৫তম bit সংখ্যা রাখার জন্য ব্যবহৃত হয়। এবং ১৬তম bit ব্যবহৃত হয় সংখ্যাটি ধনাত্মক নাকি ঋণাত্মক তা নির্দিষ্ট করার জন্য।



১৬তম bit যদি 0 হয় তবে সংখ্যাটি হবে ধনাত্মক (+) এবং 1 হলে সংখ্যাটি হবে ঋণাত্মক (–)।

যেমন- 5 একটি পূর্ণ সংখ্যা এবং এর binary মান হল 101 এবং এটা মেমোরীতে নিম্নোক্তভাবে সংরক্ষিত থাকবে



শিক্ষার্থীরা integer. c প্রোগ্রামটি লক্ষ্য করুন।

Program **integer.c**

```
# include < stdio.h>
# include <conio.h>
  {
     int i;
     i= 32767;
     printf ("Value of i is % d", i);
     i = 42767;
     printf ('\n New value of i is % d", i);
     getch ( );
  }
```

Output

```
Value of i is 32767
New Value of i is –22769
```

প্রোগাম বিশ্লেষণ (Analysis of Program)

- int i;
  i = 32767;

এখানে i-কে int টাইপ ভেরিয়েবল হিসেবে ঘোষণা করা হয়েছে এবং i এর মান দেয়া হয়েছে 32767।

- prinif ("Value of i is %d", i);

prinif-এর মধ্যে %d ব্যবহৃত হয় কোন int টাইপ value কম্পিউটার এর পর্দায় প্রিন্ট করার জন্য।

- i = 42767;

এখানে i-এর মান দেয়া হয়েছে 42767 যা int টাইপ ভেরিয়েবল সর্বোচ্চ যে মান সংরক্ষণ করতে সক্ষম তার চেয়ে বেশী।

- printf ("New Value of i is %d", i);

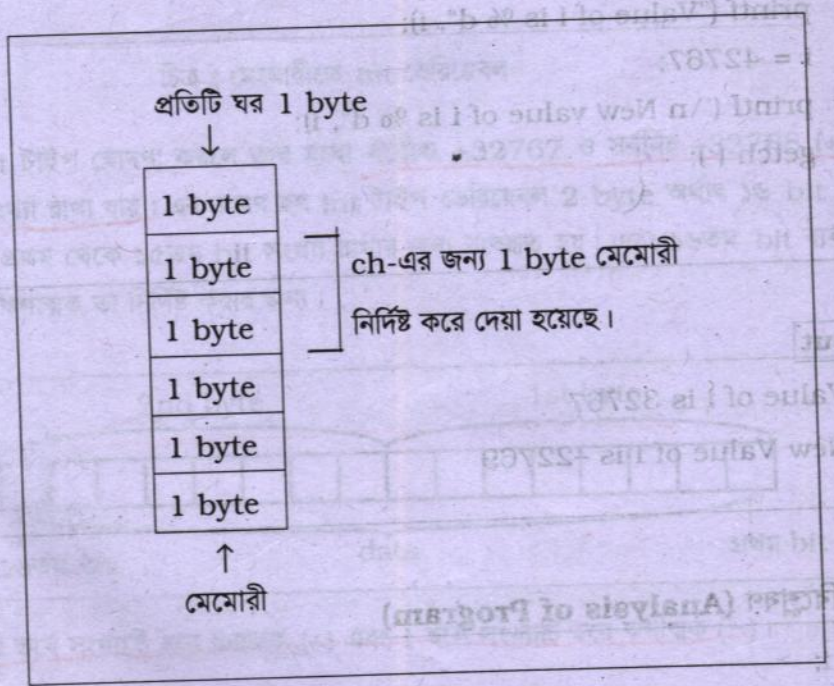
প্রোগাম Run করলে i এর মান দেখা যাবে- 22769 যা অসামঞ্জস্যপূর্ণ। কম্পিউটার ভেদে এই মান ভিন্ন হতে পারে তবে অবশ্যই সামঞ্জস্যপূর্ণ হবে না। কারণ int type ভেরিয়েবল সর্বোচ্চ 32767 পর্যন্ত মান সংরক্ষণে সক্ষম।

Char ভেরিয়েবল

C প্রোগামে character ব্যবহারের উদ্যেশ্যে ভ্যারিয়েবল -কে ক্যারেক্টার টাইপ ঘোষণা করা হয়। কোন ভেরিয়েবলকে char টাইপ ঘোষণা করলে C কম্পাইলার মেমোরীতে ঐ ভেরিয়েবল এর জন্য 1 byte স্থান নির্দিষ্ট করে। যেমন—

```
Char ch;
```

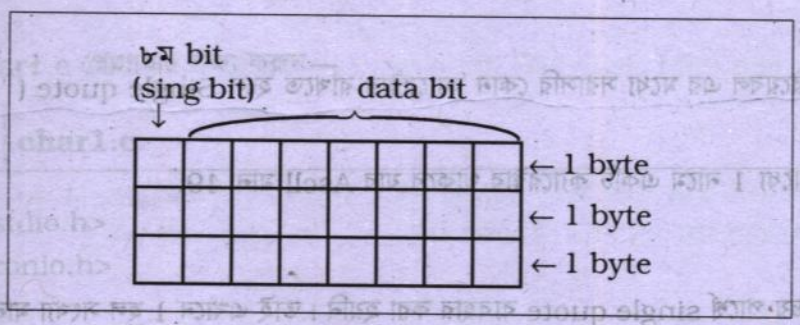
এখানে ch কে char টাইপ ঘোষণা করা হয়েছে এবং মেমোরীতে ch-এর জন্য 1 byte (8 bit) জায়গা নেয়া হয়েছে।



চিত্র ঃ মেমোরীতে char ভেরিয়েবল

কোন ভেরিয়েবলকে char টাইপ ডিক্লেয়ার করলে তার মধ্যে সর্বোচ্চ +127 ও সর্বনিম্ন –128 মানের ASCII ক্যারেক্টার সংরক্ষণ করা যায়।

char টাইপ ভেরিয়েবল-এর জন্য মেমোরীতে 1 byte স্থান নির্দিষ্ট হয় এবং এর মধ্যে ৮ম বিট ব্যবহৃত হয় সংখ্যাটি ধনাত্মক (+) না ঋণাত্মক (–) তা নির্দিষ্ট করার জন্য।



নিম্নের charval.c প্রোগ্রামটি লক্ষ্য করুন

Program **charval.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
     char ch1, ch2;
     clrscr ( );
     ch1 = '1';
     ch2 = 1;
     printf ("ch1 is %c", ch1);
     printf (" \n ch2 is %c ", ch2);
     getch ( );
  }
```

output

```
ch1 is 1
ch2 is
```

Charval.c প্রোগ্রাম বিশ্লেষণ (Analysis of program)

❑ char ch1, ch2;

এখানে ch1 ও ch2 নামে দুটি char টাইপ ভেরিয়েবল ঘোষণা করা হয়েছে। অর্থাৎ মেমোরীতে ch1 ও ch2 প্রত্যেকের জন্য 1 byte করে স্থান নির্দিষ্ট করা হয়েছে।

❑ ch1 = '1';

কোন char ভেরিয়েবল এর মধ্যে সরাসরি কোন ক্যারেক্টার রাখতে হলে Single quote (' ') ব্যবহার করা হয়।

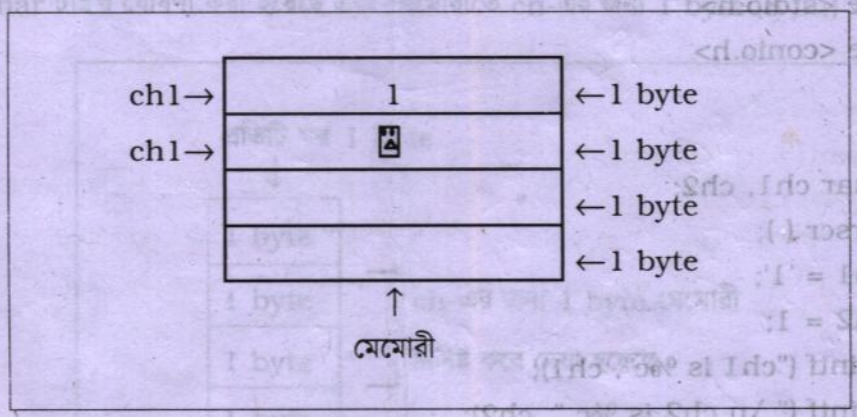
এখানে ch এর মধ্যে 1 নামে একটি ক্যারেক্টার থাকবে যার Ascll মান 49.

❑ ch2 = 1;

এখানে 1 এর উভয় পার্শ্বে single quote ব্যবহার করা হয়নি। তাই এখানে 1 হল সংখ্যা মান।

আবার আমরা জানি কোন ভেরিয়েবলকে char টাইপ ডিক্লেয়ার কররে তার মধ্যে শুধু character সংরক্ষণ করা যায়।

তাই এখানে ch2 এর মধ্যে 1 এর সমমানের ASCII ক্যারেক্টার সংরক্ষিত হবে।



❑ printf ("ch1 is %c", ch1);

এখানে OUTPUT হবে 1, কারণ ch1 এর মধ্যে ক্যারেক্টার 1 সংক্ষিত আছে।

%c printf ( ) -এর মধ্যে ব্যবহার করা হয় কম্পিউটার screen -এ ক্যারেক্টার প্রদর্শনের জন্য।

❑ printf (" \n ch2 is %c", ch2);

এখানে OUTPUT হবে ☺ কারণ মেমোরীতে ch2-এর মধ্যে ☺ সংরক্ষিত আছে।

NOTE

এখানে আমরা প্রধান যে কথাটা বুঝতে পারলাম তা হল ক্যারেক্টার 1 আর নাম্বার 1 এক নয়।

এবার নিম্নের char1.c প্রোগ্রামটি লক্ষ্য করুন—

Program **char1.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
    char ch;
    clrser ( );
    ch = 'A' ;
    pritf ("Value of %c is %d", ch, ch);
    ch = ch + 1;
    printf ("\n Value of ch is increased by 1");
    printf ("\n Value of %c is %d", ch, ch);
    getch ( );
  }
```

OUTPUT

```
Value of A is 65 65
Value of ch is increased by 1
Value of B is 66
```

প্রোগ্রাম বিশ্লেষণ (Analysis of Program)

❑ char ch;

এখানে ch একটি ক্যারেক্টার টাইপ ভেরিয়েবল।

❑ clrscr ( );

কম্পিউটারের screen (পর্দা) clear করার জন্য clrscr ( ) পিটমেন্ট ব্যবহৃত হয়।

❑ ch= 'A'

এখানে ch ভেরিয়েবল-এ A ক্যারেক্টারটি জমা থাকবে। লক্ষনীয় যে A কে single quoteএর মধ্যে রা হয়েছে। A এর ASCII মান 65.ch= 'A' লেখা আর ch = 65 লেখা একই কথা।

❑ ch = ch + 1

এক্ষেত্রে ch-এর মান বেরে হয়েছে 66।

❑ printf ("\n Val\e of %c is %d", ch, ch);

যেহেত ch-এর মান বেড়ে হয়েছে 66 তাই এবার A প্রদর্শিত না হয়ে B প্রদর্শিত হবে কারণ B এর ASC মান 66.

printf এর মধ্যে %c ক্যারেক্টার প্রদর্শনের জন্য ব্যহার করা হয়।

নিম্নের char2.c প্রোগ্রামটি লক্ষ্য করুন

Program **char2.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
     char ch;
     clrscr ( );
     ch = 127;
     printf ("Value %d is equivalent to %c", ch, ch);
```

Continue

Program **char2.c**

```
    ch = 127+1;
    printf ("\n Value of ch is increased by 1");
    printf ("\n Now value of ch is %d", ch);
    printf (" \n Value %d is equivalent to %c", ch,ch);
    getch( );
}
```

Continue

OUTPUT

```
Value 127 is equivalent to Δ
Value of ch is inereokd by 1
Now Value of ch is –128
Value –128 is equvalent to ζ
```

প্রোগ্রাম বিশ্লেষণ (Analysis of program)

- ch = 127;

ch হল character টাইপ ভেরিয়েবল। ch -এর মধ্যে কোন value (মান) রাখা যাবে না বরং কোন value -এর সমমানের ক্যারেক্টার রাখা যাবে।

এখানে ch -এর মধ্যে Δ ক্যারেক্টারটি সংরক্ষিত হয়েছে কারণ Δ এর ASCII মান 127.

- ch = 127+1;

এখানে ch-এর মান বেড়ে +128 না হয়ে বড়ং –128 হবে কারণ char টাইপ ভেরিয়েবল এর সর্বোচ্চ মান +127 হতে পারে।

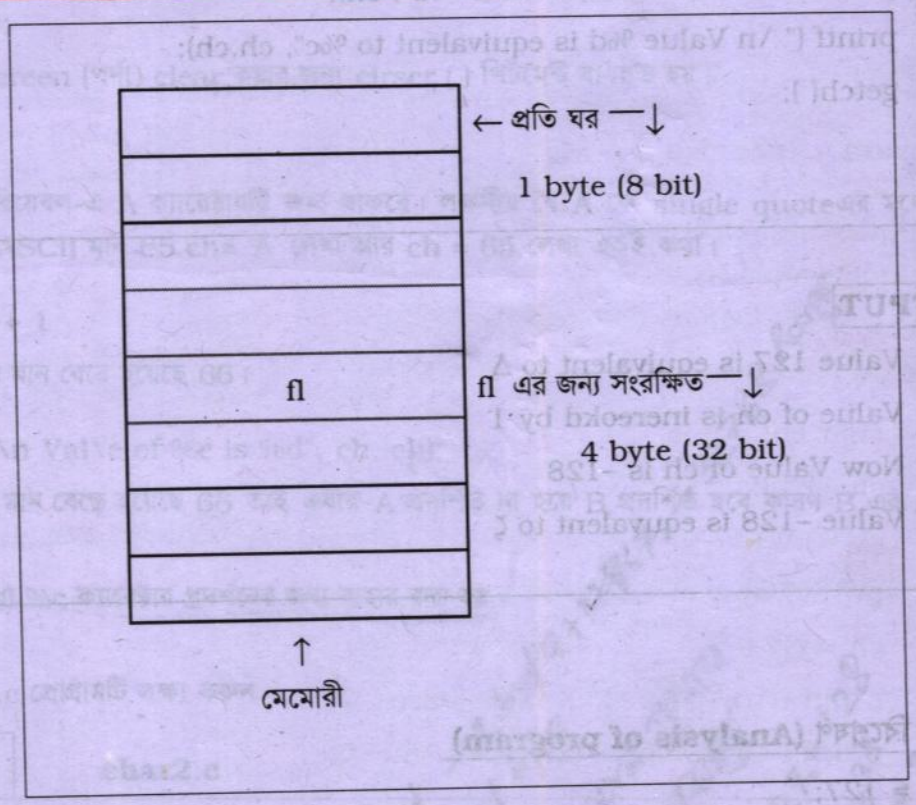
- printf ("\n Value %d is equivalent to %c", ch, ch);

এখানে ζ ক্যারেক্টারটি প্রদর্শিত হবে কারণ ch এর বর্তমান value –128 এবং ζ এর ASCII মান –128.

Float টাইপ ভেরিয়েবল (Float type variable)

Float টাইপ ভেরিয়েবল দশমিক বা ভগ্নাংশ সম্বলিত সংখ্যা সংরক্ষণের জন্য ব্যবহৃত হয়। যেমন— আমরা যদি প্রোগ্রাম 10.01 নিয়ে কাজ করতে চাই তবে একে Float টাইপ ভেরিয়েবল এর মধ্যে রাখতে হবে।

Compiler float টাইপ ভেরিয়েবল এর জন্য মেমোরীতে 4 byte স্থান সংরক্ষণ করে। যেমন—



float fl;

fl এর জন্য Compiler মেমোরীতে 4 byte স্থান সংরক্ষিত করবে। এই 4 byte -এর মধ্যে প্রথম 23 bit দশমিকের আগের পূর্ণ সংখ্যা রাখার জন্য, পরের ৮ bit দশমিকের পরের ভগ্নাংশ সংখ্যা রাখার জন্য ব্যবহৃত হয়। সর্বশেষ bit টি সংখ্যাটি ধনাত্মক না ঋণাত্মক তা নির্ণয়ের জন্য সংরক্ষিত হয়।

কোন ভেরিয়েবলকে float টাইপ হিসেবে ঘোষণা করলে তার মধ্যে সর্বোচ্চ 3.4×10^{38} থেকে সর্বনি 3.4×10^{-38} মানের সংখ্যা রাখা যায়।

Turbo c কম্পাইলার float টাইপ ভেরিয়েবল এর জন্য floating point math package ব্যবহার ক এবং দশমিকের পর সর্বোচ্চ ৬ সংখ্যা হিসেবে রাখে।

নিম্নের float1.c প্রোগ্রামটি লক্ষ্য করুন

Program float1.c

```
# include <stdio.h>
# include <conio.h>
main ( )
   {
      float fl;
      clrscr ( );
      fl = 1.0000000000;
      printf (" Value of fl is %f", fl);
      getch ( );
   }
```

দশটি শূণ্য

OUTPUT

```
value of fl is 1.000000;
```

ছয়টি শূণ্য

Float1.c প্রোগ্রাম বিশ্লেষণ (Analys's of float1.c program)

❑ float fl;

এখানে fl -কে float টাইপ ভেরিয়েবল হিসেবে ঘোষণা দেয়া হয়েছে।

❑ fl = 1. 0000000000;

এখানে fl-এর মান নির্দিষ্ট করে দেয়া হয়েছে। fl এর মান হল 1 ও 1 এর পর দশটা শূন্য।

❑ printf ("Value of fl is %f", fl);

এখানে printf ( ) এর মধ্যে %f ব্যবহার করা হয়েছে float সংখ্যা screen (পর্দা) এ প্রদর্শনের জন্য।

এখানে fl-এর মান দেখা যাবে 1.000000 অর্থাৎ দশমিকের এর পর ছয়টা শূন্য। কারণ float ভেরিয়েবল দশমিকের পর ছয়ের বেশি সংখ্যা রাখে না।

নিচের floal2.c প্রোগ্রাম fl-এর মান দেয়া হয়েছে 1 কিন্তু প্রোগ্রাম Run করালে fl এর মান দেখা যাবে 1·000000

Program **char2.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
   {
      float fl;
      clrscr ( );
      fl = 1;
      printf ("Value of fl is %f", fl);
      getch ( );
   }
```

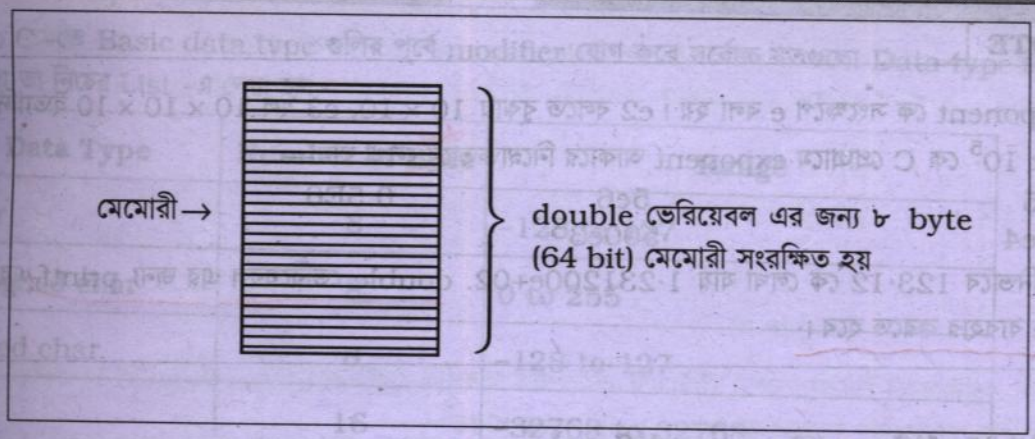
OUTPUT

```
Value of  fl is 1.000000
```

double টাইপ ভেরিয়েবল (double Type variable)

double টাইপ ভেরিয়েবল হল কিছুটা floal টাইপ ভেরিয়েবল এর মত। double টাইপ ভেরিয়েভলের মান মেমোরীতে দুইটি অংশে সংরক্ষিত থাকে। সংখ্যার মান সংরক্ষিত থাকে। সংখ্যার মান সংরক্ষিত থাকে mantissa অংশে এবং এর power থাকে exponent অংশে। একে exponential notation ও বলা হয়।

double টাইপ ভেরিয়েবল এর জন্য মেমোরীতে ৮ byte স্থান নির্দিষ্ট হয়। এই টাইপ ভেরিয়েবল মধ্যে সর্বনিম্ন $1·7 \times 10^{-308}$ থেকে সর্বোচ্চ $1.7 \times 10^{+308}$ মান সংরক্ষণ করা যায়।



নিম্নের double1.c প্রোগ্রামটি লক্ষ্য করুন

Program **double1.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
      double dl;
      clrscr ( );
      dl = 123.12;
      printf ("Value of dl is %e", dl);
      dl = 1234.12;
      printf ("\n Value of dl is %e", dl);
      getch ( );
  }
```

OUTPUT

```
Value of dl is 1.231200e + 02
Value of dl is 1.234120e + 03
```

NOTE

Exponent কে সংক্ষেপে e বলা হয়। e2 বলতে বুঝায় 10×10, e3 হল $10 \times 10 \times 10$ ইত্যাদি। 5×10^5 কে C প্রোগ্রামে exponent আকারে নিম্নোক্তভাবে লেখা যায়—

5e5 .5e6 0.5E6
50e4 500e3

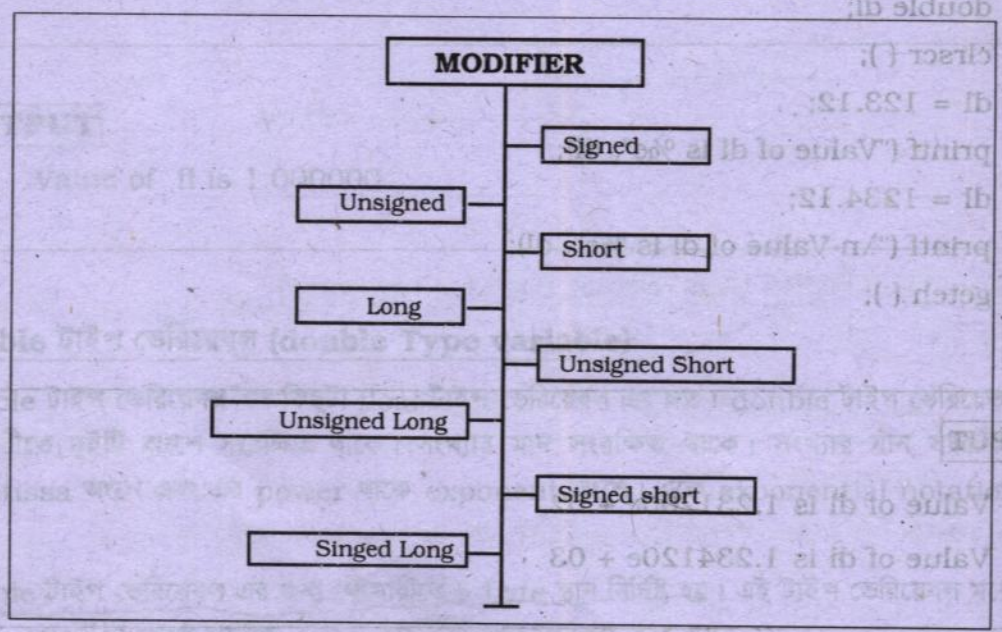
এমনিভাবে 123·12 কে লেখা যায় 1·231200e+02. double ভেরিয়েবল এর জন্য printf-এর মধ্যে %e ব্যবহার করতে হবে।

টাইপ Modifiors (Type modifiers)

int, char, float ও double এই চারটি ভেরিয়েবল টাইপকে Basic Data Type বলা হয়। Modifier এই Base টাইপগুলির অবস্থা পরিবর্তনের জন্য ব্যবহৃত হয়।

যেমন— char টাইপ ভেরিয়েবেল এর মধ্যে সর্বনিম্ন −128 থেকে +127 পর্যন্ত Value -এর সমমানের ক্যারেক্টার রাখা যায়। কিন্তু, কখনো যদি আমাদের ঋণাত্মক মানের ক্যারেক্টার এর প্রয়োজন না হয় তবে আমরা char -এর জন্য সংরক্ষিত ৮ bit এর সর্বশেষ bit কে sign bit হিসেবে ব্যবহার না করে এর মধ্যে সংখ্যা বা তথ্য রাখতে পারি। একে করে আমরা char টাইপ ভেরিয়েবল -এ o থেকে 255 সমমানের ক্যারেক্টার রাখতে পারবো।

TC কম্পাইলার যে কয় ধরনে modifier সমর্থন করে সেগুলো হল— signed, unsigned, short, long, unsingned short, unsigned long, signed short, signed long.



চিত্র ঃ বিভিন্ন ধরনের modifier.

Turbo C -তে Basic data type গুলির পূর্বে modifier যোগ করে সর্বোচ্চ যতগুলো Data type তৈরি করা যায় তা নিচের List -এ দেয়া হল—

Data Type	Number of Byte	Range
char	8	–128 to 127
unsigned char	8	0 to 255
signed char	8	–128 to 127
int	16	–32768 to 32768
unsigned int	16	0 to 65535
signed int	16	–32768 to 32767
short int	16	–32768 to 32768
unsigned short int	16	0 to 65535
signed short int	16	–32768 to 32767
Long int	32	–2147483648 to 2147683647
unsigned long int	32	0 to 4294967295
signed long int	32	–2147483648 to 2147483647
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E + 308
long double	64	1.7E-308 to 1.7E+ 308

চিত্র ঃ Turbo c এর basic data types এবং modifiers.

ভেরিয়েবল ঘোষণা (Variable Declaration)

ভেরিয়েবল ব্যবহার করার পূর্বে অবশ্যই ভরিয়েবল ঘোষণা করতে হবে। ভেরিয়েবল ডিকলেয়ার করার নিয়ম হল ঃ

```
typename variablename,.....n;
```

type name হল data type এবং variable name হল ভেরিয়েবল এর নাম যা আমরা প্রোগ্রাম ব্যবহার করব। .....n দ্বারা বুঝানো হচ্ছে এখানে একাধিক ভেরিয়েবল এর নাম ব্যবহার করা যাবে। যেমন–

```
int i, j, k;
int s;
float p, q, r;
double a;
```

কোন ভেরিয়েবল ডিক্লেয়ারেশন ষ্টেটমেন্ট এর শেষে অবশ্যই সেমিকোলন (;) দিতে হবে। একই ষ্টেটমেন্টে একাধিক ভেরিয়েবল থাকলে তা কমা (,) দ্বারা পৃথক থাকবে।

ভেরিয়েবল ইনিসিয়ালাইজেশন (Initializing variables)

ভেরিয়েবল ডিকলেয়ার করার সময়ও এতে মান দেয়া যায়। নিচের initial. c প্রোগ্রামটি লক্ষ্য করুন

Pragram **initial.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
    int salary = 1000;
    float Bonus = 100.50;
    char star = '*';
    printf ("salary is %d", salary);
    printf ("Bonus is % f", Bonus);
    printf ("star is like %c", star);
    getch ( );
  }
```

NOTE

main ( ) এর বাইরেও ভেরিয়েবল ডিকেলেয়ার করা যায়। এটা ডিকলেয়ার করতে হয় # include এর পর কিন্তু main ( ) এর পূর্বে।

ইনপুট/আউটপুট (input/output in c)

কম্পিউটারে keybard থেকে সংখ্যা শব্দ, অক্ষর ইত্যাদি ঢুকানো যায় এবং এগুলো মেমোরীতে রেখে পর্দায় (screen) দেখানো (Display) যায়। আমরা এর পূর্বে বিভিন্ন প্রোগ্রামে printf ( ) ফাংশন ব্যবহার করেছি। এটা screen এ কোন কিছু প্রদর্শন করে।

Keyboard থেকে input নেয়ার জন্য Turbo.c তে বেশ কয়েকটি ফাংশন আছে, যেমন– scanf ( ), getche ( ) ইত্যাদি। এখানে printf ( ) এবং scanf ( ) ফাংশন নিয়ে বিস্তারিত আলোচনা করা হল। printf ( ) ও scanf ( ) ফাংশনের জন্য stdio.h হেডার ফাইলটি # include করতে হবে।

printf ( ) ফাংশন

printf ( ) ফাংশন ফরমেটেড অডিটপুট (formatted output) ফাংশ কারণ screen-এ output কি রকম হবে তা printf ( ) এর মাধ্যমে পূর্বেই নির্ধারণ করা যায়। কম্পিউটার screen এ কোন কিছু প্রিন্ট করার সময় তাকে বিভিন্নভাবে সাজানো, বিভিন্ন চিহ্ন ব্যবহার করা, দশমিক এর পরের ঘর নির্ণয় করা ইত্যাদির জন্য printf ( ) ফাংশনে বিভিন্ন ঐচ্ছিক (optional) ও আবশ্যিক (mandatory) নিয়ম আছে।

printf ( ) ফাংশনে এগুলো লেখার নিয়ম হল ঃ

%[flag] [field_width] [.[precision]] [l] conversion_character.

এখানে ঐচ্ছিক নিয়মগুলো [ ] এ আবদ্ধ। % এবং conversion_cheracter অবশ্যই printf ( ) এর মধ্যে ব্যবহার করতে হবে।

conversion character.

Code	বর্ণনা
c	screen-এ একটি ক্যারেক্টার প্রদর্শন করে
s	কয়েকটি ক্যারেক্টার সম্বলিত string প্রদর্শন করে
d	signed integer প্রদর্শন করে
f	float নাম্বার প্রদর্শন করে
e	float নাম্বার প্রদর্শন করে exponent সহ
x	unsigned hexa int প্রদর্শন করে।
o	unsigned octal int প্রদর্শন করে।
l	d, x, o সহ long int প্রদর্শন করে।

Pragram **Format.c**

```
main ( )
  {
     int i = 0;
     long int li = 80000L;
     clrscr ( ) ;
     printf ("Octal value of % d is % 0 \n", i, i);
     prnintf ("Haxa value of % d is % x \n", i, i);
     printf ("Hexa value of % d with ox is % # x \n", i, i);
     printf ("long value % ld \n", li);
     getch ( );
  }
```

OUTPUT

```
Octal value of 10 is 12
Hexa value of 10 is a
Hexa value of 10 is A
Hexa value of 10 wit ox is OXA
Long value 80000
```

প্রোগ্রাম বিশ্লেষণ (Program Analysis)

- printf ("Octal value of % d is % O \n", i, i);

  এখানে printf () এর মধ্যে % 0 ব্যবহার করা হয়েছে int value এ octal মান পাওয়ার জন্য।

- Long int li = 80000L;

  li একটি Long int টাইপ ভেরিয়েবল এবং এর মান 80000L। এখানে L দ্বারা long int নির্দেশ করা হয়।

- printf ("Hexa Value of % d is %X \n", i, i);

  %x ব্যবহার করা হয় int-কে hexadecimal মানে রূপান্তরের জন্য।

- printf ("Hexa value of % d with ox is % # X \n", i, i);

  Hexa value-এর পূর্বে ox দেখতে চাইলে printf ( ) এর মধ্যে # ব্যবহার করতে হবে।

- printf ("Long value % ld \n", li);

  l এর পর d ব্যবহার করা হয়েছে long value প্রদর্শনের জন্য।

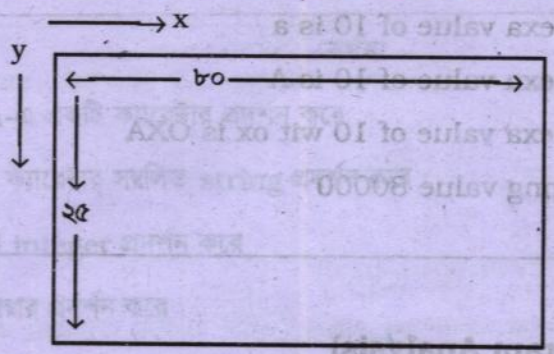
NOTE

Octal সংখ্যা 0-7 নিয়ে গঠিত

Hexa সংখ্যা 0–15 নিয়ে গঠিত

field-width

কম্পিউটারে Screen-এ একই Line-এ সর্বোচ্চ ৮০টি ক্যারেক্টার এবং সর্বোচ্চ ২৫ লাইন লেখা যায়



চিত্র ঃ কম্পিউটার Screen

একটি ক্যারেক্টার যতটুক জায়গা দখল করে তাকে Field width বলে।

প্রোগ্রাম Run করলে প্রোগ্রামের output সর্বনিম্ন কতগুলো Field ব্যবহার করবে তা এই Field-width মাধ্যমে নির্দিষ্ট করা যায়। নিম্নের প্রোগ্রামটি লক্ষ্য করুন।

নিম্নের Width.c প্রোগামটি লক্ষ্য করুন

Pragram **Width.c**

```
#  include <stdio.h>
#  include <conio.h>
main ( )
   {
      int i = 5678;
      clrscr ( );
      printf (" % d \n", i);
      printf (" % 6d \n", i)
      printf ("%—6d \n",i)
      printf ("% 06d \n", i)
      printf ("% d",—i);
      getch ( );
   }
```

OUTPUT

5	6	7	8			একটি ফিল্ড
		5	6	7	8	
5	6	7	8			
0	0	5	6	7	8	
—	5	6	7	8		

নিচের fwidth.c প্রোগ্রামটি লক্ষ্য করুন

Pragram **fwidth.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
    float fl = 10.123;
    clrscr ( );
    printf ("% 6.3 f \n", f1);
    printf ("%f \n", lf);
    printf ("%6. 2f \n", fl);
    printf ("%–6.2f \n", fl);
    printf ("%f", –fl);
    getch ( );
 }
```

OUTPUT

1	0	.	1	2	3	
1	0	.	1	2	3	
	1	0	.	1	2	
1	0	.	1	2		
—	1	0	.	1	2	3

Escape Sequence

\n একটি Escape sequence-এর উদাহরণ যা printf ( ) এর মধ্যে ব্যবহৃত হয় output নতুন লাই দেখার জন্য।

Turbo c তে আরো কতগুলো Escape sequence আছে। যেমন- \t ব্যাবহৃত হয় Tab-এর জন্য, ব্যবহৃত হয় bell শব্দের জন্য। নিম্নে Escape sequence-এর List দেয়া হল-

Saquence	বর্ণনা
\a	Bell
\b	Backspace
\n	নূতন Line
\\	ব্যাকস্লাস
\?	?
\'	single quote
\"	double quote

নিম্নের প্রোগ্রামটি লক্ষ করুন

Pragram **s_sen.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
     printf ("my \t Name \t is \t Tareq");
  }
```

OUTPUT

My ␣ Name ␣ is ␣ Tareq

tab tab tab

প্রোগ্রাম বিশ্লেষণ

এখানে printf ( ) এর মধ্যে \t ব্যবহার করার ফলে My এর Name এর মধ্যে এক tab ফাকা স্থান প্রদর্শিত হচ্ছে।

Keyboard থেকে Input নেয়া (scanf ( ) ফাংশন)

scanf ( ) ফাংশনের মাধ্যমে keyboard থেকে কোন মান, ক্যারেক্টার শব্দ (word) ইত্যাদি input নিয়ে কোন ভেরিয়েবল-এ সংরক্ষণ করা যায় এবং পরে এগুলোকে নিয়ে process করে screen-এ প্রদর্শন করা যায়।

scanf ( ) হল একটি Formatted input ফাংশন।

নিচের scant.c প্রোগামটি লক্ষ্য করুন

Pragram **scanf.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
   {
      int hours, days;
      printf ("Enter how many days : ");
      scanf ("%d", & days);
      hours = 24 * days;
      printf ("\n %d days = %d hours", days, hours);
      getch ( );
   }
```

OUTPUT

```
Enter how many deys : 10
10 days = 240 hours
```

প্রোগ্রাম বিশ্লেষণ (Analysis of program)

❑ int hours, days;

এখানে hours ও days নামে দুটি int টাইপ ভেরিয়েবল ডিকলেয়ার করা হয়েছে।

❑ printf ("Enter how many days :");

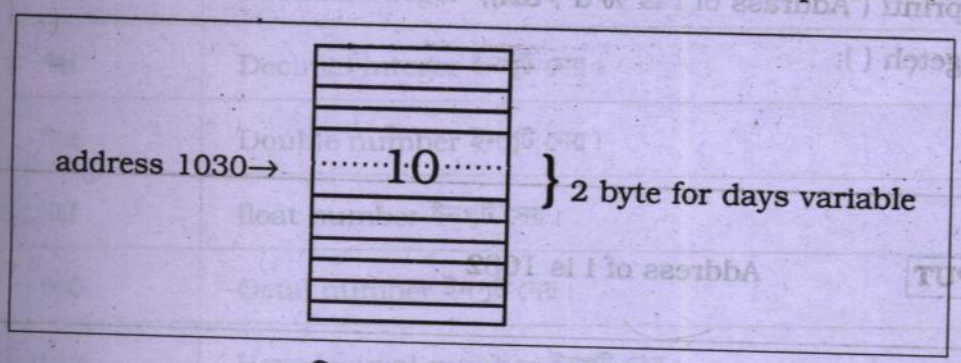
এর মাধ্যমে প্রোগ্রাম ব্যবহারকারি'র কাজে জানতে চাওয়া হচ্ছে যে সে কত দিন input দিতে চাচ্ছে।

❑ scanf ("%d", & days);

↑ address operator (ampersand)

scanf ( ) ফাংশন এর মাধ্যমে keyboard থেকে input নেয়া হয়। %d এর মাধ্যমে বোঝানো হচ্ছে এখানে int টাইপ মান (value) input নেয়া হবে। & হল ampersand ক্যারেক্টার এবং একে Turbo C-তে address operator বলা হয়। কোন ভেরিয়েবল এর পূর্বে & ব্যবহার করলে বুঝতে হবে ঐ ভেরিয়েবল এর মেমোরীর addrss কে নির্দেশ করা হচ্ছে।

মূলত এই লাইনের মাধ্যমে বুঝানো হচ্ছে যে keyboard থেকে int টাইপ Value input করতে হবে এবং এই value int টাইপ ভেরিয়েবল days এর সংরক্ষিত থাকবে। ধরি, এখানে 10 input দেয়া হল।



চিত্র ঃ scanf-এর &-এর কাজ

- hours = 24 * days;

আমরা জানি এক দিন হল 24 ঘন্টার সমান। এখানে 24 এর সাথে keyboard থেকে যে input দেয়া হবে তা গুন করে hours ভেরিয়েবল এর মধ্যে রাখা হবে।

- printf ("\n %d days = %d hours", days, hours);

প্রথম দ্বিতীয় প্রথম দ্বিতীয়

এখানে প্রথম % d দ্বারা days কে এবং হয় % d দ্বারা hours কে নির্দেশ করা হচ্ছে।

এড্রেস অপারেটর (&)

নিম্নের প্রোগ্রামটি Run করলে i এর মান দেখাবে না বরং address দেখাবে। কারণ এখানে printf এর মধ্যে &i ব্যবহার করা হয়েছে।

Program **adfress.c**

```
# include <stdio.h>
# include <conio.h>
```

Continue

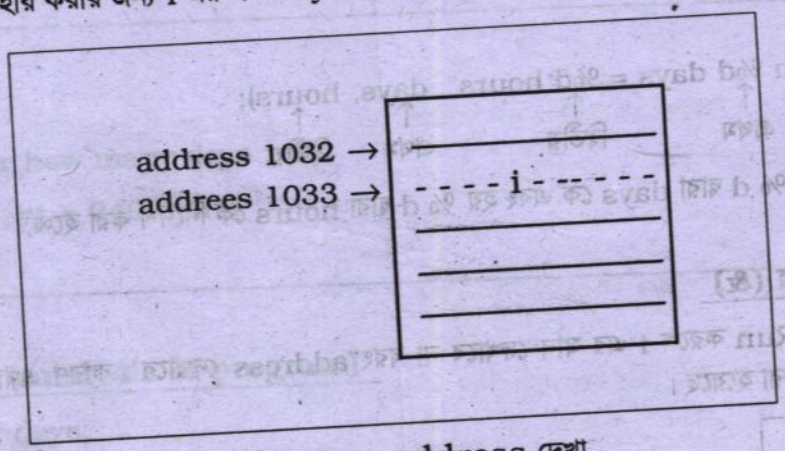
Pragram adfress.c Continue

```
main ( )
  {
    int i = 10;
    printf ("Addrass of i is % d", & i);
    getch ( );
  }
```

OUTPUT Address of i is 1032

প্রোগ্রাম বিশ্লেষণ

আমরা জানি কোন ভেরিয়েবল int টাইপ হলে তার জন্য মেমোরীতে 2 byte memory নির্দিষ্ট হয়। pri এর মধ্যে &i ব্যবহার করার জন্য i এর প্রথম byte-এর address দেখা যাবে (address 1032)



চিত্র ঃ i-এর address দেখা

scanf ( ) এর conversion character

scanf এর convorsion character প্রায় printf ( ) এর conversion character এর অনু
তবে কিছু পার্থক্যও আছে। নিম্নে scanf ( ) এর conversion character এর List দেয়া হল।

Code	বর্ণনা
%c	Keyboard থেকে একটি ক্যারেক্টার নেয় (Read)।
%s	Keyboard থেকে অনেকগুলো ক্যারেক্টার একসাথে input নেয়।
%d	Decimal integer ইনপুট নেয়।
%i	Decimal integer ইনপুট নেয়।
%e	Double number ইনপুট নেয়।
%f	float number ইনপুট নেয়।
%0	Octal number ইনপুট নেয়।
% x	Hexadecimal number ইনপুট নেয়।

Example

নিম্নের প্রোগ্রামটি Run করলে int ভেরিয়েবল এর জন্য কত byte মেমোরীর প্রয়োজন তা জানা যাবে। sizeof ( ) একটি operator যা এর parentheis (( )) এর মধ্যে লিখিত ভেরিয়েবল কত byte তা নির্দেশ করে।

Pragram **sizeof.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
   {
   printf ("An int is %d bytes", sizeof (int));
   getch ( );
   }
```

Example

নিম্নের প্রোগ্রামটি লক্ষ্য করুন। কোন expression-এ ক্যারেক্টার ব্যবহার করলে C কম্পাইলার ে ক্যারেক্টারের সংখ্যাগত মান automatically বের করে নেয়। A ও B এর ASCII মান যথাক্রমে 65 ও 6 এবং এদের বিয়োগফল 1।

Pragram **char3.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
    char ch;
    clrscr ( );
    ch = 'B'—'A';
    printf (" \' B\'–\'A\' = %d equvalent to % c", ch, ch);
    getch ( );
  }
```

Example

নিচের float3.c প্রোগ্রামে float ভেরিয়েবল এর মধ্যে Int ভেরিয়েবল এর মান নেয়া হয়েছে। Int ভেরিয়েব এর মান ছিল 2। কিন্তু এটা float ভেরিয়েবল এ রাখলে OUTPUT এ এর মান দেখা যাবে 2.000000

Pragram **float3.c**

```
# Include <stdio.h>
# include <conio.h>
main ( )
  {
    float fl;
    int it;
```

Continue

Pragram **float3.c** Continue

```
    clrscr ( );
    fl = 1;
    it = 2;
    printf ("Value of fl is %f", fl);
    printf ("\nValue of it is % d", it);
    fl = it;
    printf ("\n value of fl is % f", fl);
    getch ( );
  }
```

Example

নিম্নোক্ত float4.c প্রোগ্রাম float ভেরিয়েবল fl এর মান দেয়া হয়েছে 1 কিন্তু printf ( ) এর মধ্যে %d ব্যবহার করা হয়েছে। এজন্য প্রোগ্রামের OUTPUT হবে শূন্য (০)। এখানে printf এর মধ্যে %f ব্যবহার করতে হবে।

Pragram **float4.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
    float fl;
    clrscr ( );
    fl = 1;
    printf ("Value of fl is% d", fl);
    getch ( );
  }
```

Example

নিচের প্রোগ্রামটি Run করলে graphics character প্রদর্শিত হবে।

Pragram **achar.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
   {
      clrscr ( );
      printf ( "Graphics charucter is \xDE \n");
      printf ("Graphics character is \xDD \n");
      printf ("Graphics character is \xDC \n");
      getch ( );
   }
```

NOTE

যে সব ক্যারেক্টারের মান ASCII 128 এর বেশি সেগুলো হল Graphics ক্যারেক্টার।

Example

একটি scanf ( ) ফাংশন এর মাধ্যমে বিভিন্ন টাইপ ডাটা input নেয়া যায়। যেমন–

Program **dif-inp.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
   {
      int i;
```

Continue

Pragram **dif-inp.c** Continue

```
    float k;
    char c;
    printf ("Enter A int, float, char data");
    scanf ("%d %f %c", & i, & k, & c);
    printf ("\n int = %d, float = %f, char = %c", i, k, c);
    getch ( );
}
```

Example

নিম্নের প্রোগ্রামে ভুল data টাইপ input দিলে প্রোগ্রাম error message দেখাবে।

Pragram **scam-err.c**

```
# include <conio.h>
# include <stdio.h>
main ( )
 {
    int i;
    float f;
    printf ("Enter value for i And f");
    printf ("\nIf you Enter other than int or float then program will show a
                                    error message");
    if (scanf ("%d %f", & i, &f) = = 2)
    printf ("\n i = % d, f = % f", i, f));
    else
    printf ("Input Error");
    getch ( );
 }
```

Example

নিচের প্রোগ্রামে মাইলকে কিলোমিটারে রূপান্তরিত করা হয়।

Program **mile km.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
   {
      int miles ;
      float km;
      printf ("Enter miles to covert");
      scanf ("%d", & miles);

      km = 1.609 *miles;
      printf ("%d Miles is equvalent to % f km", miles, km);
      getch ( );
   }
```

Q&A:

1.Q. নিম্নের প্রোগ্রাম

```
# include <s
main ( )
   {
      int i = 1
      printf ("
   }
```

উত্তর : এখানে i হল
লিখতে হবে।

2.Q. নিচের প্রোগ্রামে

```
# include <
main ( )
   {
      int i;
      i = 32
      printf
   }
```

উত্তর : -32768

3.Q. নিম্নের প্রোগ্রামে

```
# include <
main ( )
   {
      clrscr
      int i =
      printf
   }
```

উত্তর : clrscr ( )

Q&A:

1.Q. নিম্নের প্রোগ্রাম ভুল কোথায়?

```
# include <stdio.h>
main ( )
 {
   int i = 10;
   printf ("Value of i is% f', i);
 }
```

উত্তর ঃ এখানে i হল int টাইপ ভেরিয়েবল কিন্তু printf ( ) এর মধ্যে %f ব্যবহার করা হয়েছে। এখানে %d লিখতে হবে।

2.Q. নিচের প্রোগ্রামের output কি হবে?

```
# include <stdio.h>
main ( )
 {
    int i;
    i = 32767 +1;
    printf ("i = %d", i);
 }
```

উত্তর ঃ -32768

3.Q. নিম্নের প্রোগ্রামের ভুল কোথায়?

```
# include <stdio.h>
main ( )
 {
    clrser ( );
    int i = 10;
    printf ("i = %d", i);
 }
```

উত্তর ঃ clrscr ( ) এর পূর্বে ভেরিয়েবল ডিকলেয়ার করতে হবে।

4.9. নিম্নের number গুলি exponential notation আকারে লিখুন।

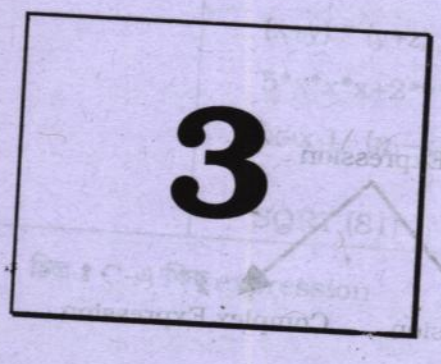
ক. 2,000,000
খ, –999.51
গ. 0.000,005

উত্তর ঃ ক 2.0e6
খ. 9.9951e2
গ. 5.0e5

Exercise

1. Variable গঠন করার নিয়ম কি?
2. Int ও char ডাটা টাইপের মধ্যে পার্থক্য কি?
3. A ও a এর ASCII মান কত?
4. printf ( ) ও scanf ( ) এর জন্য প্রোগ্রামে কোন Header File include করতে হয়?
5. Conversion ক্যারেক্টার কি?
6. Escape sequence কি?
7. Field width কি?
8. কম্পিউটার এর screen-এ একলাইনে সর্বোচ্চ কত ক্যারেক্টার লেখা যায়?

ude করতে হয়?



3

Expression, Statement And Token
(expression, স্টেটমেন্ট এবং টোকেন)

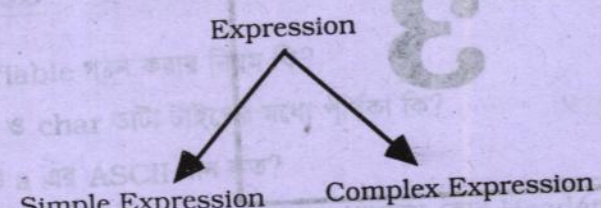
- **expression ও statement**
- **Token**

Expressions

expression হল প্রোগ্রামের এমন এক অংশযার নির্দিষ্ট সাংখ্যিক(numerie) মান আছে। যেমন— in বা # define PI ইত্যাদি। Expression-কে assignment (=) এর সাহায্যে মূল্যায়িত(evaluate করা হয়।

Variable = expression

Expression দুই প্রকার।



Simple Expression : Simple Expression কোন একক অংশ নিয়ে গঠিত। যেমন- ভেরি Constant ইত্যাদি।

Simple Expression	বর্ণনা
Name	Variable
20	Literal constant
MINUTE_PER_HOUR	Symbolic constant

Complex Expression : যখন simple expression কোন operater (+,-,%, ইত্যাদি সংযুক্ত হয় তখন তাকে Complex Expression বলে। যেমন— 2 + 8, x = y + 5, x = y ইত্যাদি।

আমরা বিভিন্ন বিষয়ে যে গাণিতিক সূত্রগুলো পরেছিলাম তা প্রোগ্রামের ও লেখা যায়। নিম্নে কিছু গাণিতিক সূত্র C প্রোগ্রাম কিভাবে লিখতে হয় তা দেখানো হল-

গাণিতিক সূত্র	C Expression
$a \times b$	a * b
$x\,(y+z)$	x * (y+z)
$(x+y)\,(y+z)$	(x+y) * (y+z)
$5x^3 + 2y^2+c$	5*x*x*x+2*y*y+c
$\frac{x-x_1}{x_1-x_2} = \frac{y-y_1}{y1-y_2}$	$(x-x_1)/(x_1-x_2) = (y-y_1)/(y_1-y_2)$
$\sqrt{81}$	SQRT (81)

চিত্র ঃ C-এ কিছু expression

STATEMENTS

আমরা জানি প্রোগ্রামের মাধ্যমে কম্পিউটারকে কোন কাজ করার নির্দেশ দেয়া যায়। Statement হল প্রোগ্রামের কোন অংশ যার মাধ্যমে কম্পিউটারকে কোন কাজ করার নির্দেশ দেয়া হয়। Statement এক লাইন বা একাধিক লাইনের হতে পারে। প্রতিটি Statement -এর শেষে semicolon (;) থাকে (# include ও # define ব্যতীত)। যেমন—

```
a = 5 + 10;

p = q;
```

Statament-এ white space ব্যবহার করা যায়। space, tab, ফাঁকা লাইন কে White space বলে। যেমন- a = 5 + 10 কে C প্রোগ্রামে নিম্নোক্তভাবেও লেখা যায়।

```
a
=
5
+
10;
```

Compound statements

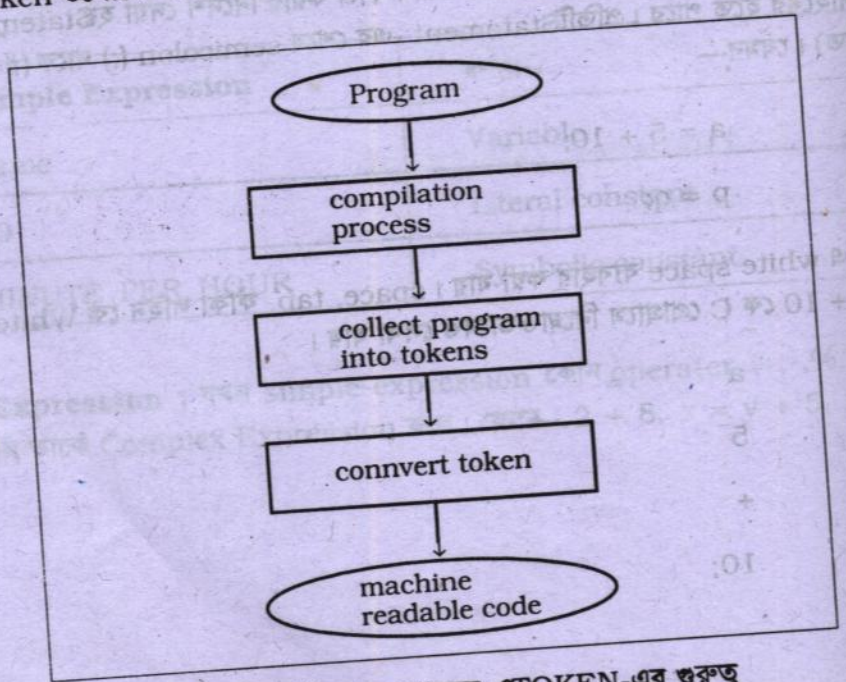
একাধিক statement যখন { } দ্বারা আবদ্ধ থাকে তখন তাকে compound statement বা block বলে। যেমন–

```
{
    printf ( " COMPOUND");
    printf ("STATEMENTS");
    printf ("EXAMPLE")
}
```

TOKEN

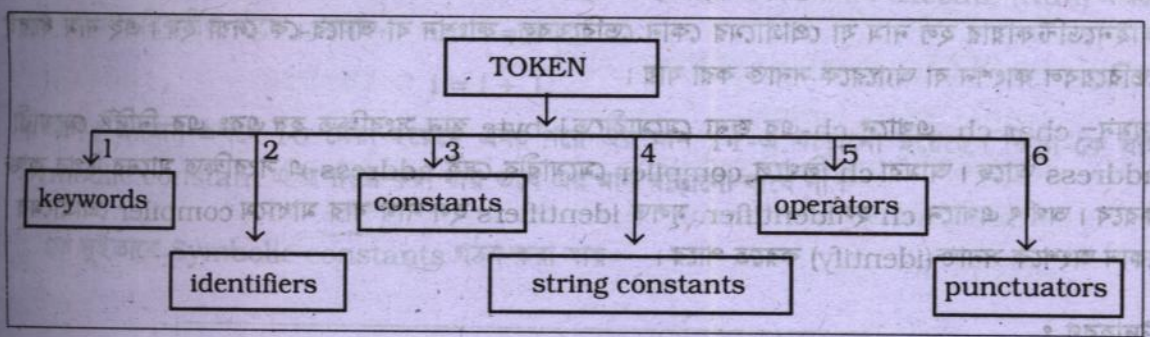
প্রোগ্রাম কম্পাইল করলে কম্পাইলার সম্পূর্ণ প্রোগ্রাম এর সঠিকতা নির্ণয় করে। কোনো প্রোগ্রামে ভুল (syntax error) না থাকলে কম্পাইলার সম্পূর্ণ প্রোগ্রামকে কতগুলো Token হিসেবে গণ্য করে।

কোন ভাষা যেমন অসংখ্য শব্দ, চিহ্ন ইত্যাদি দ্বারা গঠিত তেমনই C ল্যাংগুয়েজ Token নিয়ে গঠিত। Compiler Token গুলোকে কম্পিউটারের উপযোগী code-এ পরিণত করে।



চিত্র ঃ কম্পাইলেশন প্রোসেস-এTOKEN-এর গুরুত্ব

Turbo C-তে মোট ছয় প্রকারেToken আছে। নিম্নের চিত্রে তা দেখানো হল–



চিত্র ঃ Token এর প্রকারভেদ

KEYWORDS

Keywords হল কিছু সংরক্ষিত শব্দ (reserved words) যা কম্পাইলার সনাক্ত করতে পারে এবং তদানুযায়ী কাজ করতে পারে।

যেমন- Char একটি keyword এবং কোন ভেরিয়েবল কেchar টাইপ ডিকলেয়ার করলে compiler সেই ভেরিয়েবল এর জন্য 1 byte মেমোরী নির্দিষ্ট করে। এখানেchar হল keyword.

ANSI C সর্বমোট 32 টি keyword নির্দিষ্ট করেছে তবেTurbo C-তে 43 keyword আছে।

ANSI C STANDARD KEYWORDS			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

অতিরিক্ত Turbo c Keywords			
asm	_cs	_ds	_es
-ss	cdecl	far	huge
interrupt	near	pascal	

IDENTIFIFRS

আইনডেন্টিফায়ার হল নাম যা প্রোগ্রামের কোন ভেরিয়েবল, ফাংশন বা অ্যারে-কে দেয়া হয়। এই নাম ধরে ভেরিয়েবল ফাংশন বা অ্যারেকে সনাক্ত করা যায়।

যেমন- char ch, এখানে ch-এর জন্য মেমোরীতে 1 byte স্থান সংরক্ষিত হয় এবং এর নির্দিষ্ট মেমো address আছে। আমরা ch লিখলে compiler মেমোরীর সেই address-এ সংরক্ষিত মানের উপর ক করবে। অর্থাৎ এখানে ch হলidentifier. মূলত identifiers হল নাম যার মাধ্যমে compiler প্রোগ্রামে কোন অংশকে সনাক্ত(identify) করতে পারে।

উদাহরণ ঃ

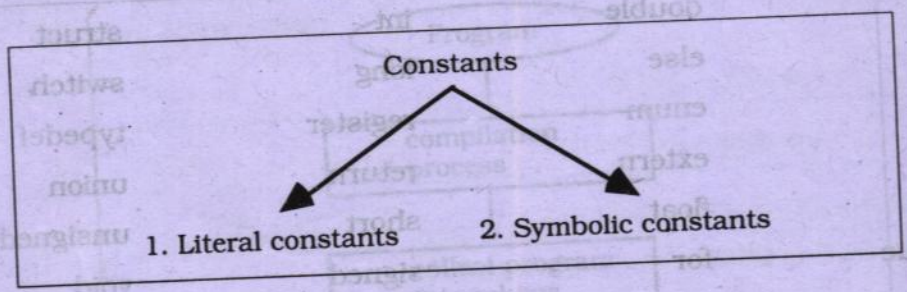
```
int i;
char ch [10];
int add (int i, int j);
```

— পরের অধ্যায়গুলোতে Array ও fanction নিয়ে বিস্তারিত আলোচনা করা হয়েছে।

CONSTANTS

Constant বলতে এমন কোন মানকে বুঝানো হয় যা প্রোগ্রাম Run করার পর পরিবর্তন করা যায় না। অ constant -এর মান প্রোগ্রাম execute করলে অপরিবর্তনীয় থাকে।

C-এ দুই প্রকারের constants আছে-



Literal Constants :

$$5x^2 + 9y+c = 11;$$

উপরের সমীকরণে 5 ও 9 হল Literal constants কারণ প্রোগ্রাম Run করার পর 5, 9 ও 11 পরিবর্তন করা যাবে না।

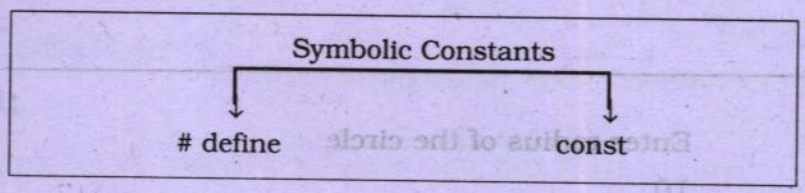
Symbolic Constants :

এ ধরনের constants-কে নির্দিষ্ট নামের মাধ্যমে উপস্থাপিত করাহয় এবং প্রোগ্রাম execute (Run) করলে এর মান অপরিবর্তীত থাকে। যেমন– int i = 10;

i = i + 1;

এখানে i এরমান প্রথমে 10 দেয়া হয়েছে এবং পরে এর মান 11-এ বাড়ানো হয়েছে। কিন্তু i-কে যদি Symbolic constant এ রূপান্তর করা যায় তবে এর মান বাড়ানো যাবে না।

C-তে দুইভাবে Symbolic constants গঠন করা যায়–



define :

define হল Preprocessor directive. # define এর শেষে semicolon (;) দেয়া যায় না। এটা লেখার নিয়ম হল–

Syntax— # define CONSTANTNAME Literal

যেমন—

```
# define    MAX   100
    ↑       ↑     ↑
preprocessor | NAME | Value
directive   space  space
```

আমরা জানি বৃত্তের ক্ষেত্রফল $=\pi r^2$ PI.c প্রোগ্রামটি বৃত্তের ক্ষেত্রফল নির্ণয় করবে।

Progsam **PI.c**

```
# include <stdio.h>
# include <conio.h>
# define PI 3.1415
main ( )
  {
```

Program PI.c Continue

```
    float area, radius;
    printf ("Enter radius of the circle");
    scanf ("%f "& radius);
    area = PI * (radius * radius);
    printf ("Area = %f", area);
    getch ( );
}
```

OUTPUT

```
Enter radius of the circle
10
Area = 314.15
```

PI.c প্রোগ্রাম বিশ্লেষণ

- # difine PI 3.1415;

এখানে PI হল constant name এবং এর মান 3.1415 এই স্টেটমেন্ট লেখার ফলে কম্পাইলার প্রোগ্রা। মধ্যে যতস্থানে PI পাবে তা 3.1415 দ্বারা প্রতিস্থাপিত করবে।

- scanf ("%f", & radius);

এখানে, radius ভেরিয়েবল এর মধ্যে Input দিতে হবে।

- area = PI * (radius * radius);

আমরা জানি বৃত্তের ক্ষেত্রফল হল πr^2-এর π (পাই) এর মান হল 3.1415। এখানে প্রথমে radius সাথে radius এর গুণ হবে এবং তার সাথে PI এর গুণ হবে। কম্পাইলার PI কে 3.1415 হিসেবে ক করবে। এভাবে প্রাপ্ত ফলাফল area -এর মধ্যে সংরক্ষিত থাকবে

NOTE

define নিয়ে preprocessor-এর অধ্যায় বিস্তারিত আলোচিত হয়েছে।

নিম্নের define1.c প্রোগ্রামটি লক্ষ্য করুন। এই প্রোগ্রামটি compile করলে কম্পাইলার Error message (ভুল ধরবে) দেখাবে।

Program **define1.c**

```
# include <stdio.h>
# include <conio.h>
# define i 10
main ( )
  {
      i = i +1;
  }
```

define1.c প্রোগ্রাম বিশ্লেষণ

প্রোগ্রামটি compile করলে কম্পাইলার নিম্নোক্ত erros message দেখাবে–

Error DEFINE 1.c 6 : Lvalue required

প্রোগ্রাম Line 6-এ i এর মান বাড়ানো হয়েছে। কিন্তু, যেহেতু হল constant তাই এর মান বাড়ানো যাবে না। তাই compiler ভুল ধরেছে।

নিম্নের i.c প্রোগ্রামটি লক্ষ্য করুন

Program **i.c**

```
# include <stdio.h>
# include <conio.h>
# define i 10
main ( )
  {
      printf ("i = % d", i);
      getch ( );
  }
```

OUTPUT

i = 10

i.c প্রোগ্রাম বিশ্লেষণ

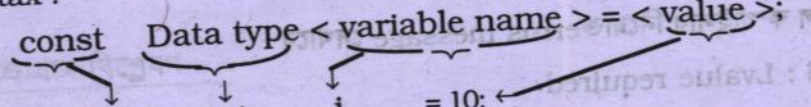
- এই প্রোগ্রামে i হল constant name এবং মান 10
- printf ("i=%.d", i);

এখানে printf ( ) এর মধ্যে constant name ব্যবহার করা হয়েছে।

Const :

Const একটি keyword, const ডিকলেয়ার করার Rules হল ঃ

Syntax :



যেমন— const int i = 10;

কোন ভেরিয়েবলকেconst হিসেবে ডিকলেয়ার করলে তারমান পরিবর্তন করা যায় না। নিম্নেরcons 1.c প্রোগ্রামটি compile করলে কম্পাইলার errer message দেখাবে।

Program **constan 1.c**

```
# include <stdio.h>
# include <conio.h>
main ( )
  {
     const int i = 10;
     clrscr ( );
     printf ("i = %d", i);
```

Continue

Program **PI.c** Continue

```
    i = i+1;
    printf ("i = %d", i);
    getch ( );
}
```

Constan1.c প্রোগ্রাম বিশ্লেষণ ঃ

- const int i = 10;

এখনে int i কে const হিসেবে ডিকলেয়ার করা হয়েছে। এবং এর মান 10 দেয়া হয়েছে।

- i = i + 1;

যেহেতু i একটি const ভেরিয়েবল তাই এর মান বাড়ানো যাবে না। এমানে কম্পাইলাক্সিনোক্ত error message দেখাবে)

Error CONSTAN 1.C 8 : Cannot modify a constant object

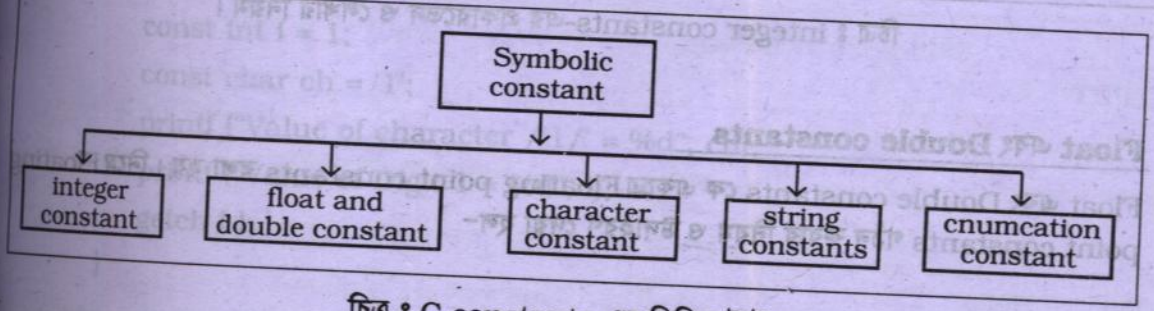
NOTE

কোন ভেরিয়েবলকেconst ডিকলেয়ার করলে তা ভেরিয়েবল ডিকলেয়ার করার সময়ই করতে হবে এবং তখনই তার মান দিয়ে দিতে হবে। যেমন– const int i;

i =10;

এভাবে constant লেখা যায় না।

Turbo C-তে বিভিন্ন প্রকারের Symbolic constant রয়েছে, যেমন–



চিত্র ঃ C constants এর বিভিন্ন টাইপ

Integer constants

integer constants পূর্ণ সংখ্যা নিয়ে গঠিত। integer cnstants লেখার কতগুলো নিয়ম আ
integer constant তিন প্রকার। নিম্নের chart-এ এগুলো উদাহরণসহ দেখানো হল–

integer constants-এর প্রকারভেদ	উদাহরণ	Rules (নিয়ম)
Decimal	1 123 –12 + 10	সর্বনিম্ন একটি সংখ্যা নিয়েinteger constants গঠিত হবে। এটা ধনাত্মক বা ঋণাত্মক হতে পারে কমা বা ফাকা স্থান ব্যবহার করা যাবে না। দশমিক চিহ্ন (.) গ্রহণযোগ্য নয়।
Octal	012 0 061	Otal integer constamts o থেকে 7 পর্যন্ত সংখ্যা নিয়ে যে কোন সংখ্যা হতে পারে। প্রথম সংখ্যা অবশ্য 0 হবে। কমা বা ফাকা স্থান গ্রহণযোগ্য নয়। দশমিক চিহ্ন গ্রহণযোগ্য নয়।
Hexadecimal	0x 0x 1 0xAB 0xF	Hexadecimal integer constant 0 থেকে 9 ও A থে পর্যন্ত সংখ্যা নিয়ে গঠিত যে কোন সংখ্যা হতে পারে। প্রথম সংখ্যা অবশ্যই0x বা 0x হবে দশমিক চিহ্ন গ্রহণযোগ্য নয় কমা বা ফাকা স্থান গ্রহণযোগ্য নয়।

চিত্র ঃ integer constants-এর প্রকারভেদ ও লেখার নিয়ম।

Float এবং Double constants

Float এবং Double constants কে একত্রে Floating point constants বলা হয়। নিম্নে Fl
point constants গঠন করার নিয়ম ও উদাহরণ দেয়া হল–

float এবং double constants	উদাহরণ	Rules (নিয়ম)
Float এবং Double constants	0.123 123.0 .123 – .123 0. 12e3 –2. 2e1 – 2.2e-1	সর্বনিম্ন একটি সংখ্যা নিয়ে float বা double constants গঠিত হতে হবে। এর decimal point থাকতে হবে কমা বা ফাকা স্থান গ্রহণযোগ্য নয় সংখ্যা exponential আকারে লিখলে mantissa এবং exponential অংশ অবশ্যই e দ্বারা পৃথক থাকবে, যেমন 3.1e2 এবং এক্ষেত্রে exponent-এ অবশ্যই সর্বনিম্ন এক সংখ্যা থাকতে হবে। Mantissa ও exponent ধনাত্মক বা ঋণাত্মক হতে পারে।

Character Constants

ক্যারেক্টার constant হল একটি সংখ্যা, অক্ষর বা বিশেষ চিহ্ন যদুটি single quote (apostrophes) দ্বারা আবদ্ধ থাকে। কোন character constants-এ সর্বোচ্চ একটি ক্যারেক্টার থাকতে পারবে। উদাহরণ–

'A' '2' '=' '!' ' '

প্রতিটি character constants-এর সমমানের ASCII মান রয়েছে। ক্যারেক্টার 1' এবং সংখ্যা 1 এক নয়। ক্যারেক্টার '1' এর ASCII মান 49 কিন্তু, 1 হল একটি সংখ্যা।

নিম্নের উদাহরণটি লক্ষ্য করুন–

Program **constval.c**

```
# include <stdio.h>
main ( )
  {
     const int i = 1;
     const char ch = '1';
     printf ("Value of character /'1/' = %d", ch);
     printf ("Value of Digit 1 = %d", i);
     getch ( );
  }
```

OUTPUT

```
Value of character '1' = 49
Value of Digit 1 = 1
```

Constval.c প্রোগ্রাম বিশ্লেষণ (Analysis of program)

- const int i = 1;

এখানে i হল const int টাইপ ভেরিয়েবল যার মান 1

- const char ch = '1';

এখানে ch হল const char টাইপ ভেরিয়েবল যার মধ্যেক্যারেকটার '1' রয়েছে এবং এরASCII মান 4

Turbo C- তে Backslash (\) Character constans গঠন করা যায়।

যেমন- '\n' কে constant হিসেবে ডিকলেয়ার করা যায়। নিম্নের প্রোগ্রামটি লক্ষ্য করুন-

Program **Constbac.c**

```
# include <stdio.h>
main ( )
   {
        const char ch = '\n';
        printf ("1%c2%c3%cHello," ch, ch, ch);
        getch ( );
   }
```

OUTPUT

```
1
2
3
Hello
```

Constbac. c প্রোগ্রাম বিশ্লেষণ

- const char ch = '\n';

এখানে ch হল const char যার মান '\n' অর্থাৎ Newline.

- printf ("1%c2%c3% Hello", ch, ch, ch);

এখানে printf ( ) এর মধ্যে তিনটি Newline ক্যারেকটার প্রিন্ট করা হয়েছে। তাইHello লেখাটা চতুর্থ লাইনে প্রদর্শিত হবে।

Turbo C-তে নিম্নোক্ত Backslash character constant রয়েছে–

Constout	বর্ণনা
'\a'	Ring (bell)
'\b'	Back space
'\f'	form feed
'\r'	carriage raturn
'\t'	Horizontal tab
'\v'	Vertical tab
'\' '	Single quote
'\" '	Double quote
'\?'	Question quote
'\\'	Backslash
'\o'	NULL

চিত্র ঃ Twrbo c-তে Backslash character constant লিষ্ট

String Constants

Double quote এর মধ্যে একাধিক সংখ্যকcharacter নিয়ে string constant গঠিত হয়। যেমন–

"TAREQ" " DHAKA, BANGLADESH" " "

"\n Newline \n Newline" " 1/(i+2) "

নিম্নের conststr.c প্রোগ্রামটি লক্ষ্য করুন–

Progsam **conststr.c**

```
# include <stdio.h>
# include <con io.h>
main ( )
  {
     const char * str = "How Are You?";
     printf ("%s", str);
     getch ( );
  }
```

OUTPUT How are you?

Conststr.c প্রোগ্রাম বিশ্লেষণ

- const char * str = "How Are you?";

string হল কতগুলো ক্যারেকটারের সমষ্টি। Trbo C-তেstring নামে কোন ডাটা টাইপ নেই। তবে ভেরিয়েবল এর পূর্বে* (Asterisk) চিহ্ন দিয়ে সেই char ভেরিয়েবল এর মধ্যে একাধিক ক্যারেক্টার string রাখা যায়। এ্যারের (ch []) মাধ্যমেও এই কাজটি করা যায় তবে এগুলোArray অধ্যায়ে বি আলোচনা করা হবে।

- যrintf ("%s", str);

  %s ব্যবহার করা হয়েছে screen-এ String প্রিন্ট করার জন্য।

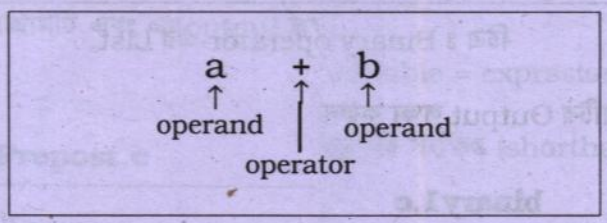
NOTE

Enumeration constant সম্বন্ধে User Defined data Type অধ্যায়ে বিস্তারিত আলোচনা হয়েছে।

এই অধ্যায়ে প্রথমে Operator নিয়ে আলোচনা করায়েছে C প্রোগামে অপারেরি এর গুরুত্ব অত্যধিক। তাই এখানে বিভিন্ন ভাগেoperator কে বর্ণনা করা হয়েছে।Operator এর পরে expression নিয়ে আলোচনা করা হয়েছে। সবশেষে comments এর বর্ণনা দেয়া হয়েছে।

OPERATORS

C প্রোগ্রামে Operator হল কোন বিশেষ চিহ্ন(symbol) বা শব্দ (word) যা কম্পাইলারকে কোন বিশেষ গাণিতিক বা যৌক্তিক কাজ করতে নির্দেশ করে। যেমন- a+b তে+' হল একটি অপারেটর। কম্পাইলার এখানে a ও b কে যোগ করবে। অপারেটর যেসব ডেটা, ভেরিয়েবল বাexpression-এর উপর কাজ করে তাদেরকে বলে Operand, যেমন a+b এখানে a ও b হলOperand.



চিত্র

C-তে বিভিন্ন ধরনের Operator আছে। এইOperator গুলোকে তাদের প্রকৃতি ও কার্য অনুযায়ী বিভিন্ন ভাগে ভাগ করা যায়। যেমন-

C operators

- Mathematical operators
  - Unary mathematical operators
  - Binary mathematical operators
- Assignment operators
- Logical operators
- Relational operators
- Conditional operators
- Bitwise operators
- Special operators

Mathematical Operators

C-এর mathematical operator গুলো গাণিতিক কাজ করে থাকে যেমন- যোগ, বিয়োগ, গুণ ইত্যাদি। C-তে মোট 5 টা binary এবং 2 টা Unary mathematical operator আছে।

Binary Operators : C-এর প্রতিটি binary operator একবারে দুটিOperand নিয়ে কাজ করতে পারে। নিচে এর একটি লিষ্ট দেয়া হল-

Operator	Symbol (চিহ্ন)	কাজ (Operation)
Addition	+	দুটি Operand-কে যোগ করে
Subtraction	—	প্রথম Operandথেকে ২য় Operand বিয়োগ করে
Multiplication	*	দুটো Operand কে গুণ করায়
Division	/	প্রথম Operand কে ২য় Operand দিয়ে ভাগ করে
Modulus	%	প্রথম Operand কে ২য় Operand দিয়ে ভাগ করার পর ভাগশেষ থাকে, তা দেখায়।

চিত্র ঃ Binary operator-এর List.

নিম্নের binary 1.c প্রোগাটির Output লক্ষ্য করুন

Program **binary1.c**

```
# include <stdio. h)
main ( )
   {
      int x = 20, y = 6;
      printf ("%d", x/y);
      printf ("%d", x%y);
      y = x + y;
      printf ("%d", x/y);
      printf ("%d", x/y);
      getch ( );
   }
```

OUTPUT

```
3
2
0
20
```

binary1.c প্রোগ্রাম বিশ্লেষণ

- যেহেতু x ও y উভয়ই int টাইপ ভেরিয়েবল, তাই ভাগফল অবশ্য intটাইপ হবে। এজন্য 20/6 হবে ও এবং দশমিকের পরের অংশ বাদ যাবে(20/6 = 3.3)

- x%y

  Modulus operator x কে y দিয়ে ভাগ করলে যে ভাগশেষ থাকবে তা দেখাবে। এখানে 20%6=2 হবে। % অপারেটর float বা doblle-এর ক্ষেত্রে ব্যবহৃত হয় না।

- y = x+y

  এখানে y-এর মান 26 হয়েছে।

- x/y

  এর output হবে 0 কারণ 20/26 একটি integer division

- x%y

  এখানে output হবে 20 কারণ 20/26-এর ভাগশেষ হল 20.

Unary Operator : যে operator শুধুমাত্র একটি Operand নিয়ে কাজ করতে পারে তাকেUnary Operator বলে। নিম্নে এর একটিList দেয়া হল।

Operator	Symbol (চিহ্ন)	কাজ (Operation)
Inorement	++	Operand এর সাথে 1 যোগ করে।
Decrement	--	Operand থেকে 1 বিয়োগ করে।
Unary minus	—	Operand এর আগে বসে এর ঋণাত্মক মান প্রকাশ করে। যেমন : –a, –x ইত্যাদি।

চিত্র ঃ Vnary operator-এর List.

++x হল x = x +1 এর অনুরূপ। ++ যদি Operand-এর আগে থাকে তবে তাকে বলে Preincrements, যেমন ঃ ++x।

++ যদি Operand-এর পর থাকে তবে তাকে বলেPostincrement. যেমন ঃ x ++

তদ্রূপ -- Oprand-এর আগে কিংবা পরে থাকলে তাকে যথাক্রমে Predecrement বা Postdecrement বলে। যেমন ঃ --n, x--

উদাহরণ ঃ (Postincrement)

```
x = 10
y = x++
```

এখানে, x -এর মান 10, y -এর মান 10 -এবং পরে x এর মান বেড়ে11 হয়েছে।

উদাহরণ ঃ (preincrement)

```
x = 10
y = ++x;
```

এখানে x এর মান প্রথমে 10, পরে x এর মান বেড়ে হয়েছে 11 এবং y এর মান 11.

নিম্নের prepost.c প্রোগ্রামটি এবং এরoutput লক্ষ্য লক্ষ্য করলে preincrement ও postincrem সহজে বোঝা যাবে।

Program **Prepost.c**

```
# include <stdio.h>
main ( )
  {
     int i = 10, j = 10;
     ++i;
     J++;
     printf ("i=%d, J=%d", i, j);
     printf ("\n i=%d, j=%d", ++i, j++);
     printf ("\n i=%d, j =%d", i++, ++j);
     getch ( );
  }
```

OUTPUT

```
i = 11, j = 11.
i = 12, j = 11.
i = 12, j = 13
```

Assignment Operators :

C-তে assignment operator হিসেবে '=' কে ব্যবহার করা হয়। কোন expression বা ভেরিয়েবল এর মানকে অপর কোন ভেরিয়েবল-এ রাখার জন্য '=' ব্যবহার করা হয়। যেমন–

```
int x = 5, y = 2, z;
z = x + y;
```

এখানে x -এর মান 5 ও y-এর মান রাখা হয়েছে2।

প্রথমে x ও y -এর মান যোগ হবে এবং পরে এই সম্মিলিত মানz -এর মধ্যে রাখা হবে। এখানেz-এর মান হবে 7.

Assignment operator লেখার সাধারণ নিয়ম হল variable = exprssion;

প্রোগ্রামে assignment operator ব্যবহারের বেশ কয়েকটি সংক্ষিপ্ত (shorthand) রূপ রয়েছে। যেমন–

x = x + 1 কে x + =1 লেখা যায়
x = x + (y + 1) কে x + = y + 1 লেখা যায়

NOTE

সাধারণ গণিতে x = y লেখার অর্থ xহল y-এর সমান (equal) কিন্তু, C-তে x = y লিখলে compiler y-এর মান কে x-এর মধ্যে রাখে।

Turbo C-তে একাধিক assignment একবারে লেখা যায়এর 'একে multiple assignment বলে। যেমন–

```
x = y = 10;
```

নিম্নের assign. c প্রোগ্রামটি লক্ষ্য করুন

Program **assign.c**

```
# include <stdio.h>
main ( )
  {
    int i = j = 15;
```

Continue

Program **PI.c** Continue

```
    printf ("i =% d, j = % d", i, j);
    i + = j;
    printf ("\ni=%d, j=%d", i,j);
    getch ( );
}
```

OUTPUT

```
i = 15 j = 15
i = 30 j = 15
```

assign. c প্রোগ্রাম বিশ্লেষণ

i এবং j -এর মান একসাথে15 দেয়া হয়েছে এবং পরেi + = j -এর মাধ্যমে i এর মান 30 হয়েছে কা
+ =j হল i = i + j এর সংক্ষিপ্ত রূপ

Relational Operators

C প্রোগ্রাম কোন ভেরিয়েবল এর সাথে অপর কোন ভেরিয়েবল বা মানের তুলনা(Comparison) ক
Relational operator ব্যবহার করা হয়। যেমন ঃa কি b -এর চেয়ে ছোট?'এই উক্তিটি
নিম্নোক্তভাবে লেখা হয়–

```
a<b
```

যে expression-এ relational Operatar থাকে তাকে Relational expression
Relational expression-এর মান হয় 1 কিংবা 0 হবে এবং C প্রোগ্রামে 1-কে True (সত্য) ও
False (মিথ্যা) হিসেবে ধরা হয়। যেমন ঃ

5 <10 ——→ সত্য

10 < 5 ——→ মিথ্যা

c-তে মোট ৬ টা Relational operator আছে। নিম্নের List -এগুলো দেখানো হল ঃ

Operator	চিহ্ন (Symbol)	উদাহরণ	প্রশ্ন বা তুলনার বর্ণনা
Equal	= =	a == b	a-এর মান কি b-এর সমান?
Greater than	>	a>b	a-এর মান কি b-এর বেশি?
Less than	<	a<b	a-এর মান কি b-এর কম?
Greater than or equal to	> =	a>=b	a-এর মান কি b-এর বেশি কিংবা সমান?
Less than or equal to	< =	a<=b	a-এর মান কি b-এর কম কিংবা সমান?
Not equal	! =	a! = b	a-এর মান কি b-এর সমান না?

চিত্র ঃ C-এর Relational Operator

Example

মনে করি p ও q দুটি int টাইপ ভেরিয়েবল এবংp-এর মান 10 এবং q -এর মান 5।

Relational Expression	True/False	প্রোগ্রাম মান
P = = Q	False	0
P! = Q	True	1
P>Q	True	1
P> = Q	True	1
P*15> Q*100	False	0

Relational Expression-এর ফলাফল যে 1 কিংবা0 হয় তা নিম্নের প্রোগ্রামের মাধ্যমে বোধগম্য হবে।

Program **True Fals.c**

```
# include <stdio.h>
main ( )
{
```

Continue

Continue

Program **True Fals.c**

```
{
    int marks = 72;
    printf (" Marks %d Equal to 65?%d", marks = = 65);
    printf ("\n Marks%d Greater than 65?%d", marks>65);
}
```

OUTPUT

```
Marks 72 Equal to 65? 0
Marks 72 creater than 65?1
```

TrueFalse.c প্রোগ্রাম বিশ্লেষণ

- printf (" Marks%d Equal to 65%d", marks = = 65)

এখানে Printf ( ) এর মধ্যে একটি পূর্ণ Relational Expression ব্যবহার করা হয়েছে marks 65. marks এখানে 65 এর সমান নয়। সুতরাং এই expression টি মিথ্যা (False) তাই Print এখানে 0 প্রদর্শন করেছে। একইভাবে marks>65 expression টি সত্য এবং এর মান প্রদর্শিত হ

NOTE

= হল assignment operator, কোন ভেরিয়েবল বা মান অপর কোন ভেরিয়েবল-এ রাখার জ ব্যবহৃত হয়।

= = হল Equal to Operator

Logical Operator

C পোগ্রামে যৌক্তিক কাজ সম্পাদনে Lagical Operator ব্যবহৃত হয়। Logical Operator সাধার বা ততোধিক Relational expression-কে সংযুক্ত করে এবং একটি ফলাফল প্রদান করে।

যে expression-এ Logical oporatar থাকে তাকে Logical Expression বলা হয়, এবং সবসময়ই 1 কিংবা 0 হয়।

উদাহরণ ঃ 'marks 70-এর বেশি হলে এবং80-এর কম হলে সেই ছাত্রের grade হবে B' এই উক্তিটি প্রোগ্রামে লিখতে গেলে নিম্নের Logical Expression এর সাহায্য নিতে হবে–

```
marks> 70 && marks <80
```

c-এর Logical oprators -দের List নিম্নে দেয়া হল ঃ

Operator	চিহ্ন (Symbol)	উদাহরণ
AND	&&	a>b && b>c
OR	||	a>b || b>c
NOT	!	! a

চিত্র ঃ Logical operator -এর List.

AND -এর ক্ষেত্রে উভয় expression -এর মান 1 হলে ফলাফল 1 হবে। যেমন–

a	b	ফলাফল (Result) a && b
1	0	0
0	1	0
0	0	0
1	1	1

চিত্র ঃ AND table

OR-এর ক্ষেত্রে কমপক্ষে যে কোন একটি expression এরমান1 হলে ফলাফল 1 হবে। যেমন–

a	b	ফলাফল a || b
1	0	1
0	1	1
0	0	0
1	1	1

চিত্র ঃ OR table

NOT সত্যকে মিথ্যা কিংবা মিথ্যাতে সত্যকে পরিণত করে। যেমন–

a	b	! a	!b
0	1	1	0
1	0	0	1

চিত্র ঃ NOT table

Example

মনে করি p = 5, q = 6, r = 4. এবার নিম্নের experession গুলো লক্ষ্য করুন–

Expression	True/False	মান
(p>q) || (p<q)	True	1
(p>q) && (p<q)	False	0
(p>=r) || ((p+q)> = 5)	True	1
!p	False	0

চিত্র ঃ Logical Expression -এর উদাহরণ

CONDITIONAL OPERATOR

C-তে conditional operator হল ?:। Conditional opertor একই সাথে তিনটি operan কাজ করতে পারে বলে একে Ternary operatorও বলা হয়। যেমন–

expression1? expression 2 : expression 3

এখানে প্রথমে expression 1 যদি True (nonzers) হয় তবে expression মূল্যায়িত হবে এবং conditional expression -এর মান হবে expression2 এর মান। কিন্তু যদি expression False (zero) হয় তবে expression 3 মূল্যায়িত হবে এবংসম্পূর্ণ conditional expressio মান হবে expression 3 -এর মান।

Example :

মনে করি, a = 5, b = 6

c = (a>b) ? a : b

এখানে, a>b কথাটি False, তাই b -এর মানই হবে c -এর মান অর্থাৎ c -এর মান 6.

x -এর মান	y -এর মান	Statement	Result
x = 10	y = 5	z = (x>y)? x : y	z = 10
x = –5		2 = (x<0)? 0 : 1	z = 0
	y = 1	z = y? 1 : 5	z = 1

চিত্র ঃ conditional operatos -এর উদাহরণ ঃ

z = (x>y)? x : y; এই conditional statement টি নিম্নোক্তভাবেও লেখা যায়–

```
if (x>y)
z = x;
else
z = y;
```

একটি conditional expression-এর মধ্যে তার একটি conditional expression রাখা যায় এবং একে বলে Nested conditional expresion যেমন–

flag = (num <0)? –1 : (num = = 0) ? 0 :1

নিম্নের Flag.c প্রোগ্রামটি কোন সংখ্যা ধনাত্মক (+ ve) নাকি ঋণাত্মক তা নির্ণয় করবে।

নিম্নের max_num.c প্রোগ্রামটি দুটি সংখ্যার বড় সংখ্যাটি নির্ণয় করবে।

Program **max_num.c**

```
# include <stdio.h>
main ( )
```

Continue

Program **max-num.c** Continue

```
{
    int first, second, maximum;
    printf ("Enter First number :");
    scanf ("%d", & first);
    printf ("\n Enter second numbes :");
    scanf ("%d", & second);
    maximum = (first>second)? first : second;
    printf ("\n Maximum Number is:%d", maximum);
    getch( );
}
```

INPUT/OUTPUT

```
Enter First number : 10
Enter second number : 5
Maximum Number is : 10
```

max_num. c প্রোগ্রাম বিশ্লেষণ

প্রথমে দুটি সংখ্যা input নেয়া হয়েছে এবং পরে conditional operator -এর মাধ্যমে বড় সংখ্যাটি নি
করে maximum -এর মধ্যে রাখা হয়েছে। পরে maximum প্রদর্শন করা হয়েছে।

Program – **Flag.c**

```
# include <stdio.h>
main ( )
{
    int num, flag;
```

Continue

Program **Flag.c**

```
printf ("Enter a Number (–32768 to 32767):" );
scanf ("%d", & num);
clrscr ( );
printf ("-1 means Negative");
printf ("\n O means Zero").
printf ("\n 1 means Positive");
flag = (num <0)?–1 : (num = = 0)?0 : 1;
printf ("\n Number is%d", flag);
getch ( );
}
```

INPUT/OUTPUT

```
Enter a Number (–32768 to 32767) : –32
–1 means Negative
0 means zero
1 means Positive
Number is -1
```

SPECIAL OPERATORS

C-তে কিছু special operator আছে যেগুলো বিশেষ বিশেষ ক্ষেত্রে ব্যবহৃত হয়। যেমন-Comma operator (,), sizeof operator, pointer operator (*)। এখানে comma operator ও sizeof operator নিয়ে আলোচনা করা হল।pointer operator নিয়ে pointer অধ্যায়ে আলোচনা করা হয়েছে। Bitwise operator নিয়ে Bitwise operator অধ্যায়ে আলোচনা করা হয়েছে।

Comma Operator

C-তে বিভিন্নভাবে comma operatar-কে ব্যবহার করা যায়। (,) এর মাধ্যমে একাধিক subexpression কে যুক্ত করে একটি পূর্ণঙ্গ expression গঠন করা যায়। এক্ষেত্রে প্রথমে সর্ববামের

sub expression টি মূল্যায়িত (evaluated) হয় পরে বাম থেকে ডান দিকের subexpression evaluated হতে থাকে এবং সর্বডানের Subexpression টির মান হবে পূর্ণাঙ্গ expression টির নিম্নের উদাহরণটি লক্ষ্য করুন–

ধরি, x = 5, y = 3, z = 0 প্রোগ্রামে নিম্নের statement টি লেখা হল–

```
z = (x + 1, y + 1)
```

এখানে সর্ববামের Subexpression টি প্রথমে মূল্যায়িত হবে অর্থাৎ x -এর মান 1 বেড় হবে 6, প -এর মান 1 বেড়ে হবে 4 এবং সম্পূর্ণ Expression (x+1, y+1) -এর মান হবে y এর মান অর্থাৎ 4 এই 4 হবে z -এর মান।

comma. c প্রোগ্রামটি লক্ষ্য করুন

Program comma.c

```
# include <stdio.h>
main ( )
  {
     int a, b, c, total;
     total = (a = 5, b = 4, c = a, a + b + c);
     printf ("total = %d", total);
     getch ( );
  }
```

OUTPUT total = 14

comma. c প্রোগ্রাম বিশ্লেষণ

a+b+c এর মানই হবে total -এর মান। তাই output হবে 14.

একই ডাটা টাইপ এর একাধিক ভেরিয়েবল একত্রে লিখতে হলে comma operator ব্যবহার কর যেমন–

```
int a, b, c;
char ch1, ch2;
float fl = 1.0, fl1 = 2.0;
```

বিভিন্ন loop-এর মধ্যেও comma operator ব্যবহার করা যায়। যেমন–

```
for (j = 0, k = 1; k<l; ++j, ++k)
```

বিভিন্ন loop সম্পর্কে পরবর্তী আধায়ে আলোচনা করা হবে।

cast operator

আমরা জানি int ভেরিয়েবলকে int ভেরিয়েবল দিয়ে ভাগ করলে ভাগফল হবে int ডাটা টাইপ, কিন্তু, int ভেরিয়েবলকে float ভেরিয়েবল দিয়ে ভাগ করলে ভাগফল হবে floal ডাটা টাইপ। যেমন,

```
int x = 10, y = 5;
```

এখানে, x/y = 2 হবে। কিন্তু, y যদি float টাইপ হয় তবে–

```
int x = 10;
float y = 5;
```

এবার, x/y = 2.000000 হবে। কারণ y হল float data type. কোন কোন সময় প্রোগ্রামে একটি সম্পূর্ণ expression এর ডাটা টাইপ সাময়িক পরিবর্তন করার প্রয়োজন হয়। এ কারণে cast operator ব্যবহৃত হয়। cast ব্যবহার করার নিয়ম হল

```
(type) expression
```

নিম্নের **cast1.c** প্রোগ্রাম লক্ষ্য করুন

Program **cast1.c**

```
# include <stdio.h>
main ()
 {
    float x;
    int p=12, q = 8;
```

Continue

Program **cast1.c** Continue

```
    x = p/q;
    printf ("Value of x =%f", x);
}
```

OUTPUT Value of x = 1.000000

cast1.c প্রোগ্রাম বিশ্লেষণ

❑ x = p/q;

p ও q হল int ডাটা টাইপ, তাই 12/8 এর ভাগফল 1.5 -এর 5 বাদ যাবে এবং থাকবে 1, এই 1 ই মধ্যে যেয়ে হবে 1.000000 কারণ x হল float.

p কিংবা q- যে কোন একটি ভেরিয়েবলকে float ডিকলেয়ার করলে ভাগফল 1.5 পাওয়া যাবে।

কিন্তু, আমরা যদি p ও q কে int টাইপ রেখে ভাগফল 1.5 পেতে চাই তবে p/q expression-এ operator ব্যবহার করতে হবে। cast1.c প্রোগ্রামে নিম্নের statement টি লিখলে ভাগফল 1.50 হবে।

```
x = (float) p/q;
```

PRECEDENCE

কোন expresion-এ একাধিক operator ব্যবহৃত হলে কোন operator -এর কাজ আগে হবে তা operator-এর precedence এর উপর নির্ভর করে। অর্থাৎ, যে operator এর precedence সেটার কাজ আগে হবে। যেমন–

```
i = 2 + 3 *4
```

যেহেতু, (*) এর precedence (+) operator -এর চেয়ে অধিক, তাই প্রথমে 3 * 4 হয়ে গুণফল সাথে যোগ হবে। i -এর মান হবে 14.

```
i = (2+3) * 4
```

() এর precedence * operator -এর অধিক, তাই ( ) এর মধ্যের 2 + 3 যোগ হয়ে যোগফল 4 এর সাথে গুণ হবে। i এর মান হবে 20 নিম্নের preced. c প্রোগ্রামটি লক্ষ্য করুন–

Program **preced.c**

```
# include <stdio.h>
main ( )
  {
     int x = 3, y = 4, z = 3, k = 1, p;
     p = x<y || x <z && z<k; /*Using no parentheses*/
     printf ("x=%d", x);
     p = (x<y || x<z) && z<k; /* Using parentheses*/
     printf ("\n x=%d", x);
     getch ( );
  }
```

ASSOCIATIVITY

কোন expression-এ যদি একাধিক operator ব্যবহৃত হয় এবং এই সকল operator-এর prcedence যদি সমান হয় তবে এদের কাজ ডান থেকে বামে নাকি বাম থেকে ডানে হবে তা এই operator গুলোর associativity -এর উপর নির্ভর করবে। যেমন–

```
v = 3 * 5/7;
```

এই statement এ * ও / অপারেটর এর precedence সমান কিন্তু এখানে * এর কাজ আগে হবে কারণ *, / এবং % এর associativity বাম থেকে ডানে এবং 3*5/ 7-এ * সর্ববামে রয়েছে।

নিম্নে বিভিন্ন ধাপে (step) associativity -এর উদাহরণ দেয়া হল ঃ

Expression	Operation (কাজ)
3*4/5 + 5/5+9–3+6/9	*
12/5+5/5 + 9–3 + 6/9	/
2+5/5 + 9–3+6/9	/
2+1 + 9–3+6/9	/
2+1 + 9–3+0	+
3+9–3+0	+
12–3+0	–
9+0	+
9	

চিত্র ঃ Associativity -এর উদাহরণ ঃ

C অপারেটর এর Precedence ও associativity

Operators	বর্ণনা	Associativity	Level
( ) [ ] → .	function expression Array expression Structure operator Structure dot operator	বাম থেকে ডান	1
– ++ –– ! ~ * & sizeof ( ) (type)	Unary minus increment Dcorement Logical negation One's complement Pointer reference Address Size in bytes Type cast	ডান থেকে বাম	2

Operators	বর্ণনা	Associativity	Level
* / %	Multiplication Division Modulus	বাম থেকে ডান	3
+ -	Addition Substraction	বাম থেকে ডান	4
<< >>	Left shift Right shift	বাম থেকে ডান	5
< <= > >=	Less than Less tham or equal to Greater than Greater than or equal to	বাম থেকে ডান	6
== !=	Equal to Not equal to		7
&	Bitwise AND	বাম থেকে ডান	8
^	Bitise exclusive OR	বাম থেকে ডান	9
|	Bitwise inclusive OR	বাম থেকে ডান	10
&&	Logical AND	বাম থেকে ডান	11
||	Logical OR	বাম থেকে ডান	12
?:	conditional expression	ডান থেকে বাম	13
= * = /= %= + = - = & = ^ = != <<= >>=	Assignment operators	ডান থেকে বাম	14
,	Conma operator	ডান থেকে বাম	15

Example

নিম্নের preced 1.c পেপ্রাগ্রামের মাধ্যমে বোঝা যাবে যে relational operator এর mathematical oporator -এর precedence বেশি।

Program **preced1.c**

```
# include <stdio.h>
main ( )
   {
      int x = 2, y = 4;
      printf ("Value is%d", x+1<y);
      printf ("\n Value is%d", y <x+4);
      getch ( );
   }
```

Example :

decrement (- -) অপারেটর এর উদাহরণ

Program **decremen.c**

```
# include <stdio.h>
main ( )
   {
      int value = 0;
      printf ("Value =%d \n", value);
      printf ("Value = % d \n", value -- );
      printf ("Value = %d \n", value);
      printf ("Value = %d \n",-- value);
      getch ( );
   }
```

Example :

নিম্নের প্রোগ্রামটি কোন সংখ্যার অ্যাবসুলেট (absolute) মান নির্ণয় করবে। (কোন ঋণাত্মক সংখ্যার absolete মান ধণাত্মক)

Program **absolute.c**

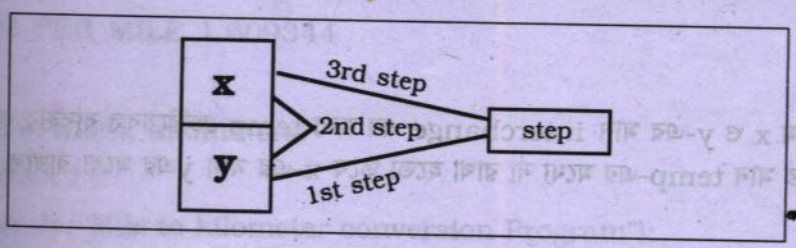
```
# include <stdio.h>
main ( )
  {
    int number, abs_val;
    printf ("Enter a number:");
    scanf ("%d", & number);
    abs_val = (namber <0)?–number : number;
    printf ("\n Absolute Value is:%d", abs_val);
    getch ( );
  }
```

INPUT/OUTPUT

```
Enter a number : –50
Absolute Value is : 50
```

Example :

নিম্নের swap. c প্রোগ্রামটি x-এর মান y-এ রাখবে এবং y-এর মান x-এ রাখবে। এখানে একটি temporary (অবস্থায়) ভেরিয়েবল ব্যবহার করা হয়েছে। এখানে প্রথমে y-এর মান temp-এর মধ্যে



রাখা হয়েছে (temp = y) পরে y এর মধ্যে x এরমান রাখা হয়েছে (y = x) এবং শেষে x এর মধ্যে temp এর মান রাখা হয়েছে (x = temp)। একে swapping বলা হয়।

Program **swap.c**

```
# include <stdio.h>
main ( ) {
        int x, y, temp;
        printf ("Enter Value for x:"); scanf ("%d", & x);
        printf ("Enter Value for y:"); scanf ("%d", &y);
        temp = y:
        y = x;
        x = temp;
        printf ("Value of x = %d", x);
        printf ("Value of y =x d", y);
        getch ( )
        }
```

INPUT/OUTPUT

```
Enter Value for x : 10
Enter Value for y : 5
Value of x = 5
Value of y = 10
```

Example :

swap. c প্রোগ্রাম x ও y-এর মান interchange এর জন্য temp ভেরিয়েবল ব্যবহার করা হয়েছে প্রথমে যদি y -এর মান temp-এর মধ্যে না রাখা হতো তবে x-এর মান yএর মধ্যে রাখলে y-এর মান যেতো।

নিম্নের swap1.c প্রোগ্রামে কোন temporary ভেরিয়েবল ব্যবহার করা হয়নি।

Program **swap1.c**

```
# include <stdio.h>
main ( ) {
        int a, b;
        printf ("Enter Value for a");
        scanf ("%d", & a);
        printf ("\n Enter Value for b");
        scanf ("%d", & b);
        b = a+b;
        a = b–a;
        b = b–a;
        printf ("\n Value of a =%d", a);
        printf ("\n Value of b=%", b);
        getch ( );
        }
```

Example :

নিম্নের km.c প্রোগ্রামটি মাইলকে রূপান্তরিত করে কিলোমিটার বানাবে। 1 mile = 1.609344 kilometers.

Program **km.c**

```
# include <stdio.h>
# define KM_PER_MILE 1.609344
main ( ) {
        float miles, kilometers;
        clrscr ( );
        prinft ("Mile to kilometer conversion Program");
        printf ("\n Enter Miles");
```

Continue

Program **km.c** Continue

```
        scanf ("%f", & miles);
        printf ("\n%f Miles = %f kilometers", miles, miles * KM_PER_MILE);
        getch ( );
        }
```

Example :

নিম্নের root.c প্রোগ্রামটি যে কোন int টাইপ সংখ্যার বর্গমূল বের করবে। এজন্য এখনে sqrt ( ) ফ ব্যবহার করা হয়েছে। sqrt ( ) ফংশনটির জন্য math. h হেডার ফাইল include করতে হবে।

Program **root.c**

```
# include <stdio.h>
#include <math.h>
main ( ) {
        int i;
        printf ("Enter a int Type number");
        scanf ("%d", &i);
        printf ("\n square root of %d = %d", i, sqrt (i));
        getch ( );
        }
```

Example :

নিম্নের celsius. c প্রোগ্রামটি Fahrenheit তাপমাত্রাকে celsius -এ রূপান্তরিত করবে।

সূত্র ঃ $\frac{c}{5} = \frac{F-32}{9}$

Program **celsius.c**

```
# include <stdio.h>
main ( ) {
        float fahrenheit, celsius;
        printf ("Fahrenheit to celsius conversion program \n");
        printf ("Enter Degree Fahrenheit :");
        scanf ("%f", & fahrenheit);
        celsius = ((Fahrenheit–32.0) * 5.0) /9.0;
        printf ("\n Degree celsius = %f", celsius);
        getch ( );
        }
```

Example :

নিম্নের প্রোগ্রামটি pound কে gram-এর পরিবর্তন করে।

Program **poun.c**

```
# include <stdio.h>
main ( )
  {
     const int gram_per_pound = 454;
     long grams, pound;
     printf ("Enter Pound");
     scanf ("%d", & pound);
     grams = pound * gram_per_pound;
     printf ("%d pounds = %d grams", pound, grams);
     getch ( );
  }
```

Example :

নিম্নের avg.c প্রোগ্রামটি দুটি সংখ্যার গড় নির্ণয় করবে।

Program **avg.c**

```
# include <stdio.h>
main ( )
   {
      int x, y;
      float average, Z;
      printf ("Enter First number");
      scanf ("%d", x);
      printf ("% \n Enter Sccond unmber");
      scanf ("%f", & y);
      Z = x + y;
      average = Z/2;
      printf ("\n Average is%f", average);
      getch ( );
   }
```

Example :

cast ব্যবহার করলে ডাটা টাইপ সাময়িক পরিবর্তিত হয় এবং এর পূর্বের ডাটা টাইপ বহাল থা

Program

```
# include <stdio.h>
main ( )
   {
      float p = 5.5;
      printf ("Real value of p =%f", p);
      printf ("\n (int) cast Value of p = %d", (int) p);
      printf ("\n Real Value of p = % f", p);
      getch ( );
   }
```

Q&A:

Q.1. ক্যারেক্টার a ও A-

উত্তর : a এর ASCII মা

Q.2. a ও b হল int ট

উত্তর : a/b = 0

Q.3. 50%11 =?

উত্তর : 5%11 = 6

Q.4. compound ass

উত্তর : + = , - =, * =

Q.5. 5^3 কে c Cmpre

উত্তর : pow (5,3). Pow

Q.6. c তে কোন ফাইল

উত্তর : Header File-

Q.7. c কম্পাইলার True

উত্তর : 0 হল False এব

Q.8. কোন চলমান বস্তুর

: সময় পরে তার শেষবেগ

v = u +

এমন একটি প্রোগ্রাম লিখুন

উত্তর : float u, a, t, v;

printf ("Enter V

scanf ("%f %f

v = u + a * t;

printf ("\n Velo

Q.9. নিম্নের প্রোগ্রাম expr

int i = 5, j = 6,

u = = i + j;

printf ("k = %d

উত্তর : k = = i + j;

অপারেটর যা কেবলমাত্র co

Q&A:

Q.1. ক্যারেক্টার a ও A-এর মধ্যে পার্থক্য কি?

উত্তর ঃ a এর ASCII মান 97 ও A -এর ASCII মান 65

Q.2. a ও b হল int টাইপ ভেরিয়েবল এবং a = 5, b = 6 হলে a/b = ?

উত্তর ঃ a/b = 0

Q.3. 50%11 =?

উত্তর ঃ 5%11 = 6

Q.4. compound assignment অপারেটরগুলো কি কি?

উত্তর ঃ + = , − =, * = , % =

Q.5. 5^3 কে c Cmpression এ রূপান্তরিত করুন?

উত্তর ঃ pow (5,3), Pow ( ) হল একটি ফাংশন যা math.h হেডার ফাইল-এ সংরক্ষিত থাকে।

Q.6. c তে কোন ফাইল এর extension হল .h?

উত্তর ঃ Header File-এর File extension হল .h

Q.7. c কম্পাইলার True ও False কিভাবে বোঝে?

উত্তর ঃ 0 হল False এবং 0 ছাড়া অন্য যে কোন সংখ্যা হল True।

Q.8. কোন চলমান বস্তুকণার আদি বেগ (initial velocity) যদি u হয় এবং ত্বরণ যদি a হয়, তবে t সময় পরে তার শেষবেগ

$$v = u + at$$

এমন একটি প্রোগ্রাম লিখুন যা u, a ও t এর মান input হিসেবে নিবে এবং v এর মান প্রদর্শন করবে।

উত্তর ঃ
```
float u, a, t, v;
printf ("Enter Value for u, a, t:");
scanf ("%f %f %f", & u, &a, &t);
v = u + a * t;
printf ("\n Velocity v =%f", v);
```

Q.9. নিম্নের প্রোগ্রাম expression এর ভুল কি?

```
int i = 5, j = 6, k;
u = = i + j;
printf ("k = %d", k);
```

উত্তর ঃ k = = i + j, এখানে = = এর স্থানে = অপারেটর ব্যবহার করতে হবে। = = হল equal to অপারেটর যা কেবলমাত্র conditional statement-এ ব্যবহার করা যায়।

Exercise

1. নিম্নের প্রোগ্রামের ভুল কোথায়?

   z = (a = b)? a : b

2. নিম্নের Quadratic Equation টি প্রোগ্রাম আকারে লিখুন।

$$x = \frac{-b + \sqrt{b^2-4ac}}{2a}$$

3. a^{b^c} প্রোগ্রামে কিভাবে লিখতে হয়?

4. ক্যারেক্টার \r, \t, \f এর কাজ কি?

5. নিম্নের Oscillations এর কম্পাঙ্কের সূত্রটি প্রোগ্রাম আকারে লিখুন

$$n = \frac{1}{2\pi\sqrt{Lc}}$$

6. $\mu = \frac{1}{\text{sinc}}$ -এর মান প্রোগ্রামের মাধ্যমে বের করুন।

7. Token কি?

8. int i;
   float f;
   i/f;

এখানে i/f -এর মান কোন্ ডাটা টাইপের হবে।

9. -- a ও a-- এর মধ্যে পার্থক্য কি?

10. ASCII শব্দের অর্থ কি?

11. C প্রোগ্রাম নিম্নের সূত্রটি লিখুন ঃ

$$R = \sqrt{A^2+B^2+2ABCos\theta}$$

12. নিম্নের expression গুলোতে True/False নির্ণয় করুন।

    i = 1, j = 2, k = 3

    i) i>j && j>k

    ii) j <k || k<i

    iii) (k>i = = j>i) && k<i

13. C-তে কোন statement এর পর semicolon দিতে হয় না?

14. Type conversion কি?

15. Mathematical operator ও Arithmatic operator কি একই operator?

16. Operator এর ক্ষেত্রে precedence ও associativity কি?

4

প্রোগ্রামে সিদ্ধান্ত গ্রহণ এবং লুপ নিয়ন্ত্রণ
(Decision Control And Loop Control)

- **if**
- **if else**
- **else if**
- **switch**
- **goto**
- **for**
- **while**
- **do...while**
- **break**
- **continue**
- **exit**
- **Examples**

সিদ্ধান্ত নেয়া (Decision Control)

আমরা জানি C প্রোগ্রাম হল কতগুলো statement-এর সমষ্টি এবং computer প্রতিটি statement গুলো Line by line পরে এবং সেই অনুযায়ী কাজ করে। অর্থাৎ কম্পিউটার C প্রোগ্রামের প্রতিটি লাইন উপর থেকে নিচের দিকে Sequentially (ক্রমানুসারে) execute করে। কিন্তু, কখনো কখনো সিদ্ধান্ত নেয়ার প্রয়োজন পরে এবং এই সিদ্ধান্ত অনুযায়ী কম্পিউটার কাজ করে। যেমন- যদি এর সমান বা বেশি নাম্বার পাওয়া যায় তবে কম্পিউটারের পর্দায় pass লেখাটি উঠবে নতুবা Fail প্রদর্শিত হবে।

C প্রোগ্রামে সিদ্ধান্তমূলক সমস্যা সমাধানের জন্য চার ধরনের Decision control statements এগুলো হল–

1. if এবং if else
2. else if
3. switch
4. goto

If Statement

if হল c ল্যাংগুয়েজ -এর একটি অত্যন্ত গুরুত্বপূর্ণ decision control statement.

if statement লেখার সাধারণ রূপ হল–

```
if (expression)
statement;
```

বা

```
if (expression)
{
statement1
statement2
}
```

if প্রথমে expression -এর মান নির্ণয় করে। Expression -এর মান যদি true (non-zero) হয় if এর সঙ্গের statement অনুযায়ী কম্পিউটার কাজ করবে অর্থাৎ statement-টি execute Expression-এর মান যদি false (zero) হয় তবে statement টি execute (কার্যকরী) হবে না

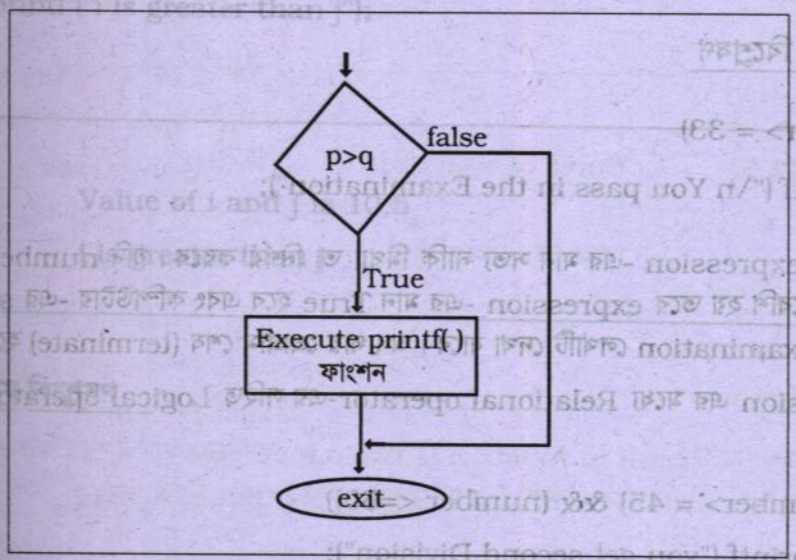
if -এর expression-এ সাধারণত Relational operator ব্যবহৃত হয় (যেমন– >, >-, <, <=

উদাহরণ ঃ

```
if (p>q)
    printf ("p is greater than q');
```

এখানে p যদি q-এর চেয়ে বড় হয় তবে printf ( ) ফাংশনটি execute (কার্যকরী) হবে। কিন্তু, p যদি q-এর সমান বা ছোট হয় তবে p>q expression টি মিথ্যা (false) হবে এবং printf ( ) ফাংশনটি execute হবে না।

Flow chart এর মাধ্যমে উপরোক্ত উদাহরণটি নিম্নে দেখানো হল–



চিত্র ঃ if statement -এর ফ্লোচার্ট

নিম্নের if1.c প্রোগ্রামটি লক্ষ্য করুন

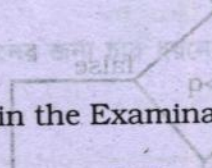
Program **if1.c**

```
# include <stdio.h>
main ( ) {
        int number;
        printf ("Enter your number :");
        scanf ("%d", & number);
        if (number> = 33)
            printf ("\n% you pass in the Examination");
        getch ( );
        }
```

INPUT/OUTPUT

```
Enter your number : 35
You pass in the Examination
```

If 1.c প্রোগ্রাম বিশ্লেষণ



❑

```
if (number> = 33)
        printf ("\n You pass in the Examination");
```

এখানে if প্রথমে expression -এর মান সত্য নাকি মিথ্যা তা নির্ণয় করবে। যদি number -এর মান 33 এর সমান কিংবা বেশি হয় তবে expression -এর মান True হবে এবং কম্পিউটার -এর sereen-এ pass in the examination লেখাটি দেখা যাবে। অন্যথায় প্রোগ্রাম শেষ (terminate) হবে।

if -এর expression এর মধ্যে Relational operator-এর সহিত Logical operatorও ব্যবহার করা যায়। যেমন–

```
if ((number> = 45) && (number <=60))
        printf ("you get second Division");
```

NOTE

```
If (a>b); /* Semicolon is invalid*/
printf ("a is greater than b");
```

এখানে if (a>b)-এর পরে সেমিকোলোন (;) হবে না

if এর সহিত একাধিক statement ব্যবহার করা যায়। নিম্নের if 2.c প্রোগ্রামে এটি দেখানো হল

Program **if 2.c**

```
# include <stdio.h>
main ( ) {
        int i = 10, j = 5;
```

Continue

Program **if 2.c**

```
if (i>j)
{
  printf ("Value of i and j is %d, %d\n", i, j);
  printf ("i is greater than j");
}
```

OUTPUT

```
Value of i and j is 10,5
i is greater than j
```

if 2.c প্রোগ্রাম বিশ্লেষণ

```
if (i>j)
  {
  .....
  .....
  .....
  }
```

if-এর সহিত একাধিক statement ব্যবহার করতে হলে তা {. . . .} এর মধ্যে লিখতে হয়।

Nested if

একটি if -এর মধ্যে একাধিক if ব্যবহার করা যায় এবং একে nested if বলে।

নিম্নের if3.c প্রোগ্রামটি লক্ষ্য করুন

Program **if3.c**

```
#include <stdio.h>
main ( ) {
    pintf ("Enter character h i \n");
    if (getche ( ) = = 'h' ) /* first if statement*/
```


Continue

Program **if3.c**

```
if (getche ( ) == 'i') /* second if statement*/
    printf ("\n you are obedient");
getch ( );
}
```

INPUT/OUTPUT

```
Enter character hi
hi
you are obedient
```

if3.c প্রোগ্রাম বিশ্লেষণ

- getche ( ) হল একটি built in function যা conio.h হেডার ফাইল-এ সংরক্ষিত থাকে। getche ( ) ফাংশনটি keyboard থেকে একবারে একটি ক্যারেকটার input নিতে পারে।

- if (getche ( ) = = 'h')
  if (getche ( ) == 'i')

এই প্রথম if statement টি True হবে যদি h ইনপুট দেয়া হয়। এই statement টি true হলে দ্বিতীয় statementটি exeute হবে এবং দ্বিতীয় if stntement টি True হলে printf ( ) ফাংশনটি execute হবে।

else -এর ব্যবহার

if -এর সহিত else ব্যবহার করা যায় তবে এটি ব্যবহার করা optional (করা/না করা ইচ্ছা)। if -এর else ব্যবহার করার নিয়ম হল ঃ

```
if (expression)
        statement 1;
else
        statement 2;
```

যদি if এর expression-এর মান True হয় তবে statement 1 execute হবে নতুবা statement excute হবে। অর্থাৎ if expression-এর মান False হলে else এর statemet 2 execute হবে।

নিম্নের if else.c প্রোগ্রামটি keyboard থেকে সন (year) input নিবে এবং সেই সন leap year কিনা তা বলবে।

Program **ifelse.c**

```
# include <stdio.h>
main ( ) {
    int year, r, r1, r2;
    printf ("Type year : ");
    seanf ("%d", & year);
    r = year %4;
    r1 = year % 100;
    r2 = year % 400;
    if((r == 0 && r1! = 0) || r2==0)
            printf ("\n This is leap year");
    else
            printf ("\n This is not leap year");
    getch ( );
    }
```

INPUT/OUTPUT

```
Type year : 1900
This is not leap year
(Re-run)
Type year :1600
This is leap year
```

ifelse. c প্রোগ্রাম বিশ্লেষণ

- আমরা জানি কোন বৎসরকে 4 দ্বারা ভাগ করলে ভাগফল যদি শূন্য হয় তবে সেই বৎসরটি Leap year. তবে যে বৎসরগুলো 100 দ্বারা বিভাজ্য সেই বৎসর যদি 400 দ্বারা বিভাজ্য (Divisible) না হয় তবে সেই বৎসরগুলো Leap year নয়। যেমন– 1900 সন 4 দ্বারা বিভাজ্য, 100 দ্বারা বিভাজ্য কিন্তু 400 দ্বারা বিভাজ্য নয় তাই এটা Leap year নয়। কিন্তু, 2000 বা 1600 সন Leap year.

NOTE

% হল remainder operator যা ভাগ শেষ নির্ণয় করে।

Nested if else

কোন প্রোগ্রামে একাধিক কোন if -এর সাথে একাধিক if else থাকলে মনে রাখতে হবে else তার পূর্ববর্তি if-এর সহিত সংযুক্ত। যেমন–

```
if (a>b)
    if (b>c)
        printf ("Inner else");
    else
        printf ("Inner else");
else
    printf ("Outer else");
```

এখানে Inner else-এর সহিত Inner if-এর সম্পর্ক রয়েছে কিন্তু outer else-এর সহিত প্রথম if (a) সংযুক্ত (associated)

else if

C প্রোগ্রাম else if-এর মাধ্যমে if else if চেইন তৈরি করা যায়। else if-এর গঠন নিম্নরূপ–

```
if (expression 1)
    statement 1;

else
    if (expression 2)
        statement 2;
else
    statement 3;
```

নিম্নের elseif.c প্রোগ্রামটি লক্ষ্য করুন

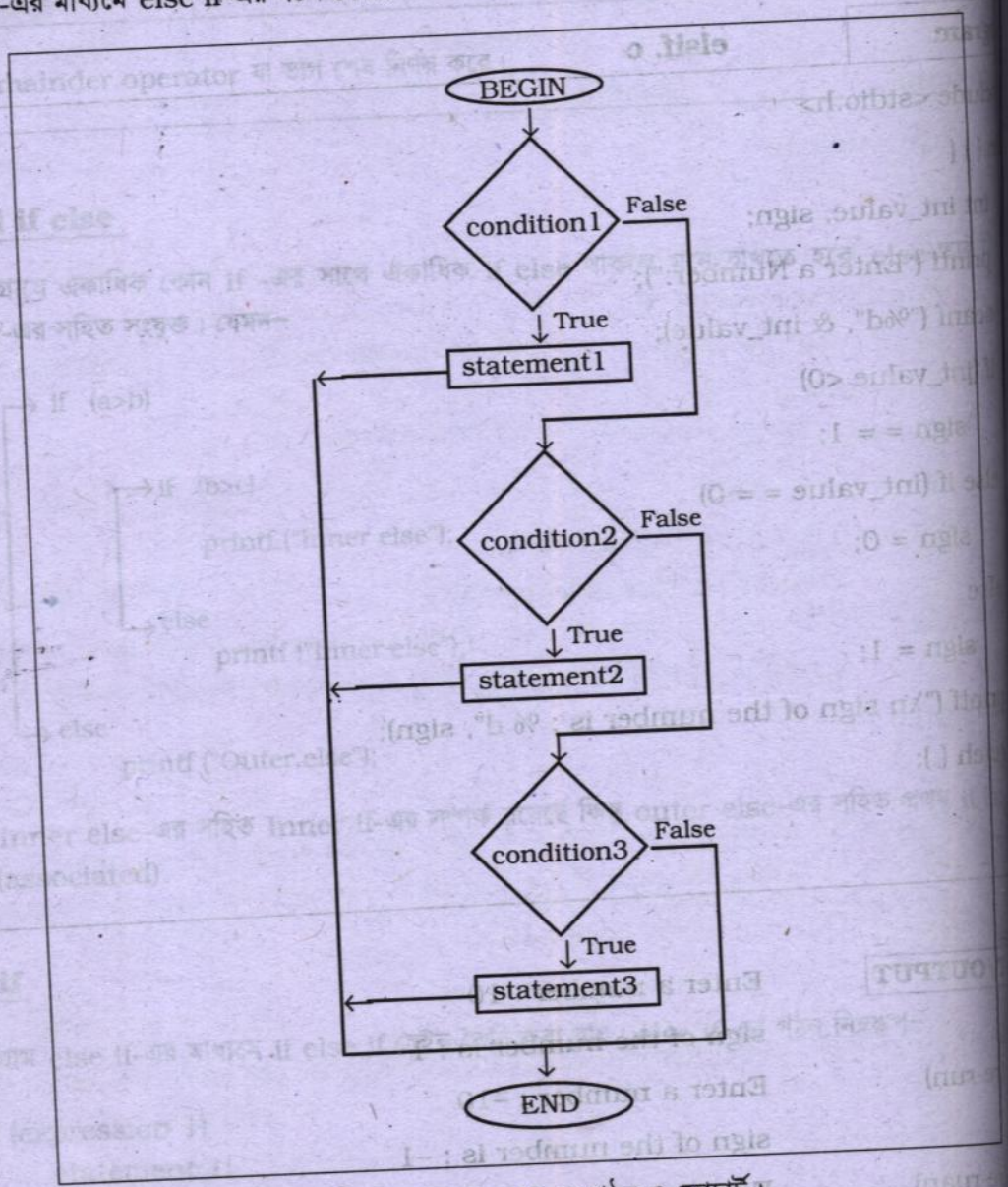
Program **elsif. c**

```
# include <stdio.h>
main ( ) {
    int int_value, sign;
    printf ("Enter a Number:");
    scanf ("%d", & int_value);
    if (int_value <0)
       sign = = 1;
    else if (int_value = = 0)
       sign = 0;
    else
       sign = 1;
    printf ("\n sign of the number is : % d", sign);
    getch ( );
}
```

INPUT/OUTPUT

```
Enter a number : 10
sign of the number is : 1
(re-run)
Enter a number : –10
sign of the number is : –1
(re-ruan)
Enter a nunber : 0
sign of the number is : 0
```

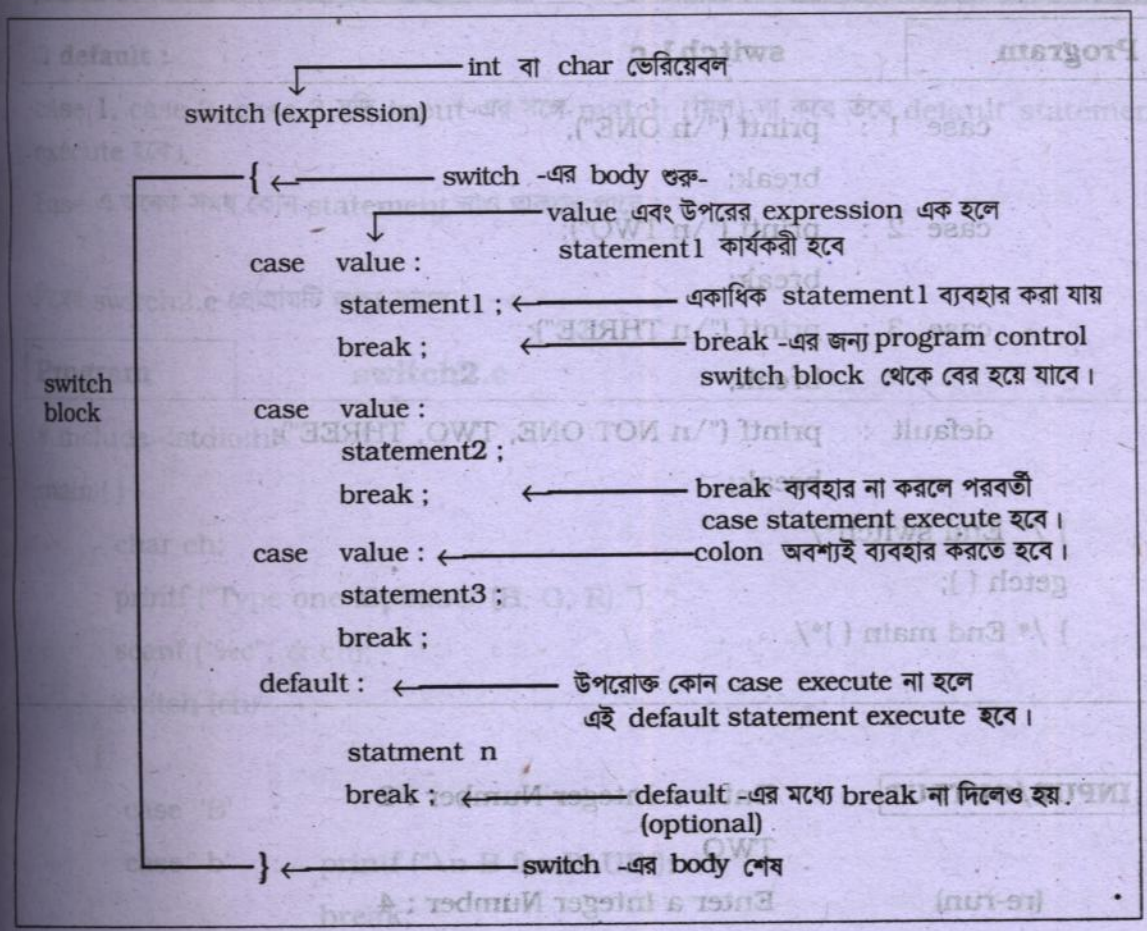
ফ্লোচার্ট-এর মাধ্যমে else if-এর গঠন দেখানো হল



চিত্র হবে ঃ else if-এর গঠন ও ফ্লোচার্ট।

switch

switch স্টেটমেন্টটি else if-এর সমগাঠনিক তবে এর format এবং readability যথেষ্ট উন্ন
প্রোগ্রামে যখন একাধিক জটিল if else থাকে তখন সেখানে if else-এর পরিবর্তে switch ব্যব
সহজবোধ্য। switch-এর format (গাঠনিক রূপ) নিম্নে দেখানো হল ঃ



নিম্নের switch1.c প্রোগ্রামটি লক্ষ্য করুন

Program **switch1.c**

```
# include <stdio.h>
main ( ) {
    int selection;
    printf ("Enter a Integer Number : ");
    scanf ("%d", & selection)'
    switch (selection)
    {
```

Program **switch1.c**

```
        case  1 :   printf ("\n ONE"),
                    break;
        case  2 :   printf ("\n TWO");
                    break;
        case  3 :   printf ("\n THREE");
                    break;
        default  :  printf ("\n NOT ONE, TWO, THREE");
                    break;
} /* End switch*/
getch ( );
} /* End main ( )*/
```

INPUT/OUTPUT

```
Enter a Integer Number : 2
TWO
(re-run)
Enter a Integer Number : 4
NOT ONE, TWO, THREF
```

switch1.c প্রোগ্রাম বিশ্লেষণ

- switch (sclection)

এখানে selection হল int টাইপ ভেরিয়েবল। কোন ক্যারেক্টার input দিলে default স্টেটমেন্ট এ হবে। কারণ case-এর সাথে 1, 2, 3 ব্যবহার করা হয়েছে int টাইপ ভেরিয়েবল হিসেবে।

- case 1: printf ("\n ONE");
  break;

1 input দিলে case 1 execut হবে এবং ONE প্রিন্ট হবে। break এর জন্য case blo প্রোগ্রাম control বের হয়ে যাবে।

❑ **default :**

case 1, case 2, case 3 যদি input-এর সঙ্গে match (মিল) না করে তবে default statement execute হবে।

case-এ অনেক সময় কোন statement নাও থাকতে পারে।

নিম্নের switch2.c প্রোগ্রামটি লক্ষ্য করুন

Program **switch2.c**

```
# include <stdio.h>
main ( ) {
        char ch;
        printf ("Type one alphabet (B, G, R):");
        scanf ("%c", & ch);
        switch (ch)
       {
         case 'B'   :
         case ' b'   :   printf ("\n B far BLUE");
                         break;
         case 'G'   :
         case 'g'   :   printf ("\n G for GREEN");
                         break;
         case 'R'   :
         case 'r'   :   printf ("\n R for RED");
                         break;
         default    :   printf ("\n you don't tyre B, G or R");
    } /* End switch*/
getch ( );
}  /* End man ( )*/
```

INPUT/OUTPUT

```
Type one alphabet (B, G, R); G
G for GREEN
Type one alphabet (B, G, R) : g
G for GREEN
```

(re-run)

Switch2.c প্রোগ্রাম বিশ্লেষণ

❑
```
case 'B' :
case 'b' : printf ("\n B for BLUE");
           break;
```

এখানে case 'B' : এর কোন স্টেটমেন্ট নেই। তাই B প্রেস করলে প্রোগ্রাম control প্রথমে case 'B' : যাবে এবং পরে case 'b' : তে transfer হবে। অর্থাৎ B যদি uppercase (B) বা lowercase (b) উভয় ক্ষেত্রেই B for BLUE প্রিন্ট হবে।

NOTE

case এর মধ্যে একাধিক statement ব্যবহার করা যায় তবে সেক্ষেত্রে {.....} ব্যবহার করতে হবে যেমন–

```
case 'A' : { printf ( );
             printf ( );
             break;
             }
```

nested switch

switch statement-এর মধ্যে একাধিক switch statement ব্যবহার করা যায়। যেমন–

```
switch (op1)
    {
      case 'A' : {
```

Continue

```
                switch (op2)
                {
                 case 'B' :
                 .............
                 ...........
                 } /* End switch (op2)*/
        } /* End case 'A'*/
} /* End switch (op1)*/
```

NOTE

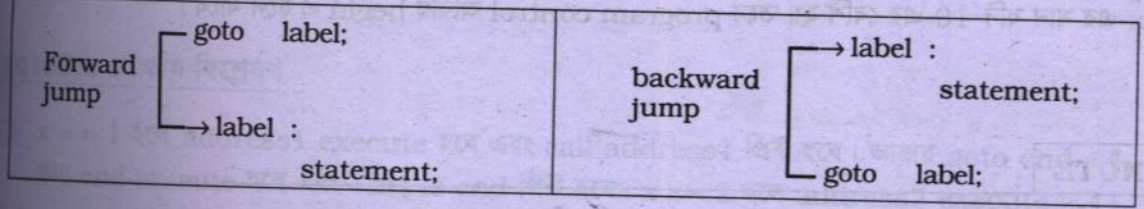
Switch-এর expression-এ float বা double ভেরিয়েবল ব্যবহার করা যায় না।

switch এর মধ্যে 256 সংখ্যক case ব্যবহার করা যায়।

goto

goto এমন একটি statement যার মাধ্যমে প্রোগ্রামের এক স্থান হতে অন্য স্থানে jump করা যায়। প্রোগ্রামে goto execute হলে প্রোগ্রামের control তখন goto দ্বারা নির্দেশিত স্থানে প্রতিস্থাপিত (branching) হয়। goto-কে অনেক সময় unconditional branching statement বলা হয়।

goto-এর মাধ্যমে প্রোগ্রামের control সামনে কিংবা পিছনে প্রতিস্থাপিত করা যায়। যেমন–



label হল কোন address বা নাম যেখানে প্রোগ্রাম jump করবে। যেমন নিম্নের goto1.c প্রোগ্রাম ল
করুন

Program **goto1.c**

```
# include <stdio.h>
main ( ) {
        int x;
        begin :
        printf ("Enter a Number.");
        scanf (" %d", & x);
        if (x> 10)
        goto begin;
        printf ("%d", x);
        getch ( );
        {
```

goto1.c প্রোগ্রাম বিশ্লেষণ

- begin :

এটা হল label বা address যেখানে প্রোগ্রাম control প্রতিস্থাপিত হবে।

- if (x>10)
  goto begin :

x-এর মান যদি 10-এর বেশি হয় তবে program control আবার begin এ চলে যাবে।

NOTE

goto1.c প্রোগ্রামটি backward jump প্রোগ্রাম

নিম্নের goto2.c হল forward jump প্রোগ্রামের উদাহরণ

Program **goto2.c**

```
#include <stdio.h>
main ( ) {
        int x;
        printf ("Enter a Number.");
        scanf ("%d", & x);
        if (x = = 1)
            goto address1;
        if (x = = 2)
            goto address2;
        address1:
             puts ("\n1 call address1");
             goto end;
        address2:
             puts ("\n2 call address2");
             goto end;
        end : printf ("\n This is end");
  getch ( ) ;
    }
```

goto2.c প্রোগ্রাম বিশ্লেষণ

- x = = 1 হলে address1 execute হবে এবং call address1 প্রিন্ট হবে। আবার goto end-এর জন্য end execute হবে এবং This is end পিন্ট হবে। x == 2 হলে address2 execute হবে।
- puts ( ) একটি built in ফাংশন যা stdio.h হেডার ফাইলে define (তৈরি) করা আছে। puts ( ) এর মধ্যে যা লেখে দেয়া হয়, কম্পিউটার screen-তাই প্রদর্শিত হয়।

NOTE

goto ব্যবহার করা ঠিক নয়। যে প্রোগ্রামে goto ব্যবহৃত হয় তাকে spaghetti code বলে।

Loop নিয়ন্ত্রণ

কখনো কখনো কোন নির্দিষ্ট কাজ (instruction) একাধিকবার করার প্রয়োজন হয়। সেক্ষেত্রে statement বারবার না লিখে Loop control statement ব্যবহার করা হয়।

C-তে মোট তিন ধরনের Loop statement রয়েছে ঃ

1. for
2. while
3. do.... while

for

C প্রোগ্রামে এক বা একাধিক statement একটি নির্দিষ্ট সংখ্যকবার execute করার জন্য for লুপ ব্যবহার করা হয়।

উদাহরণ ঃ HITLER লেখাটা কম্পিউটারের পর্দায় 50 বার দেখতে চাই। এক্ষেত্রে 50 বার printf না for লুপ ব্যবহার করা যেতে পারে–

```
for (i = 1; i<50; ++i)
    printf ("HITLER");
```

for লুপ-এর সাধারণ গঠন নিম্নরূপ

```
for (initialization; condition; increment)
        statement;
```

initialization : লুপ এর মান কত থেকে শুরু হবে তা এখানে নির্ধারণ করা হয়। এখানে স ভেরিয়েবল ও assignment (=) অপারেটর ব্যবহার করা হয়। যেমন i = 1।

condition : এখানে সাধারণত Relational expression ব্যবহৃত হয়। লুপ কখন শেষ হবে condition-এর উপর নির্ভর করে। condition যখন false হয় তখন লুপ শেষ (terminate) হয়।

inerement : initialization-এ ভেরিয়েবল -এর যে মান নির্ধারিত হয় সেই মান লুপ পুনঃপুনঃ হলে কত করে পরিবর্তিত হবে তা increment-এর মাধ্যমে নির্ধারিত হয়।

নিম্নের for1.c প্রোগ্রামটি 1 থেকে 10 পর্যন্ত প্রিন্ট করবে

Program **for1.c**

```
#include <stdio.h>
main ( ) {
        int i;
    for (i = 1; 1 <=10; ++i)
        printf ("%d",i);
    getch ( );
```

OUTPUT 1 2 3 4 5 6 7 8 9 10

For1.c প্রোগ্রাম বিশ্লেষণ

❑ For (i = 1; .....)

fov স্টেটমেন্ট এর initialization সর্বপ্রথমে হয়। এখানে প্রথমে i এর মান 1 নির্ধারিত হয়।

❑ for (.....; i<=10;.....)

initialization -এর পর condition পরীক্ষিত হয়। এখানে i এর মান 1 এবং 1 এখানে 10 এর চেয়ে ছোট, তাই condition টি true। condition যতক্ষণ True থকাবে ততক্ষণ printf ( ) ফাংশন execute হবে।

❑ for (.....;.....; ++i)

printf ( ) ফাংশন execute হবার পর ++i execute হবে এবং i এর মান 1 বাড়বে।

এভাবে লুপটি চলতে থাকবে যতক্ষণ condition False না হয়।

NOTE

For statement এর শেষে semicolon দেয়া যাবে না।

যেমন ঃ For (i = 1; i<10; ++i); ← ; হবে না।

নিম্নের প্রোগ্রামটি 10th Triangular নাম্বার বের করবে।

1+2+3+ .....+10 =?

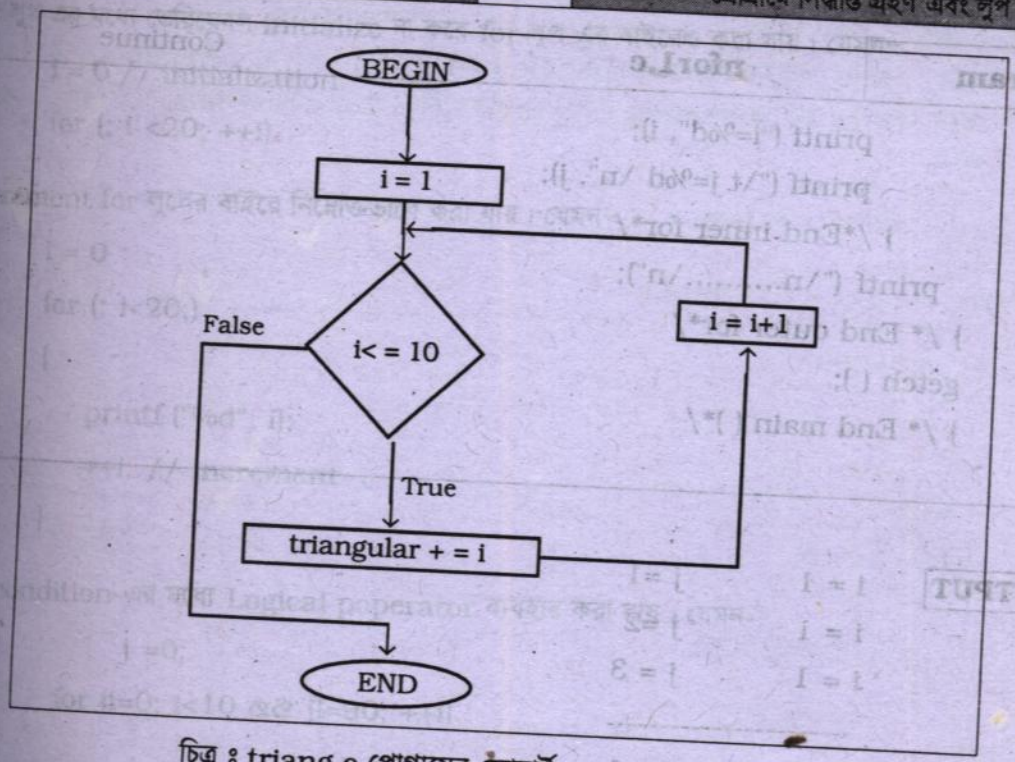
Program **triang.c**

```
# include <stdio.h>
main ( ) {
    int i, triangular = 0;
    for (i = 1; i<=10; ++i)
    triangular +=i;
    printf ("1+2+3+.....+10=%d" triangular);
    getch ( );
    }
```

OUTPUT 1+2+3+.....+10 = 55

triang.c প্রোগ্রাম বিশ্লেষণ

- triangular = triangular+i কে সংক্ষেপে triangular+ = i লেখা হয়েছে।
- প্রথমে triangular–এর মান থাকবে 0 এবং এর সাথে i এর মান 1 যোগ হয়ে যোগফল triang এ থাকবে। পরে i=2 হবে এবং triangular–এর 1 এর সহিত 2 যোগ হয়ে যোগফল 3 trian এ কারণে। এরপর i=3 হবে এবং i-এর মান triangular–এর 3 এর সহিত যোগ হয়ে যো triangular–এ থাকবে। এভাবে লুপ চলবে i<=10 পর্যন্ত। এবং সর্বশেষে triangular = 5 triang.c প্রোগ্রামকে নিম্নোক্ত Flow chart এর মাধ্যমে দেখানো যেতে পারে।



চিত্র ঃ triang.c প্রোগ্রামের ফ্লোচার্ট

Nested for Loop

একটি for লুপ এর মধ্যে এক বা একাধিক for লুপ ব্যবহার করা যায়। কখনো কখনো Nested for loop খুবই প্রয়োজনীয় হয়ে ওঠে।

নিম্নের nfor1.c প্রোগ্রামটি লক্ষ্য করুন

Program **nfor1.c**

```
# include <stdio.h>
main ( ) {
      int i, j;
  for (i = 1; i <2; ++i)
      {
       for (j=1; j<=3; ++j)
          {
```

Program **nfor1.c** Continue

```
        printf ("i=%d", i);
        printf ("\t j=%d \n", j);
      } /*End inner for*/
    printf ("\n.....\n");
  } /* End outer for*/
  getch ( );
  } /* End main ( )*/
```

OUTPUT

```
i = 1    j =1
i = 1    j =2
i = 1    j = 3
---------------
i = 2    j = 1
i = 2    j = 2
i = 2    j = 3
---------------
```

nfor1.c প্রোগ্রাম বিশ্লেষণ

- প্রথম for লুপ-এ i এর মান যখন 1 তখন দ্বিতীয় for loop এ j এর মান 1 থেকে 3। প্রথম for থেকে দ্বিতীয় for loop-এ প্রবেশ করলে দ্বিতীয় for loop false না হলে আবার প্রথম for প্রবেশ করবে না। দ্বিতীয় for loop false হলে প্রথম for loop এর মান বেড়ে হবে 2 এবং দ্বি loop এ প্রবেশ করবে। প্রথম for loop এর মান যখন ২ তখন দ্বিতীয় for loop এর মান 1 থে

বিভিন্ন রকম for লুপ (for loop variations)

for লুপ এর মধ্যে একাধিক ভেরিয়েবল initialize করা যায়। যেমন–

```
for (i=0, j=0; i<20; ++i)
for (p=1, q=1; p+q <20; ++p, q++)
```

for লুপ এর মধ্যে ভেরিয়েবল initialize না করে for লুপ এর বাইরেও করা যায়। যেমন–

```
i = 0 // initialization
for (; i <20; ++i)
```

increment for লুপের বাইরে নিম্নোক্তভাবে করা যায়। যেমন–

```
i = 0
for (; i<20;)
{
    printf ("%d", i);
    ++i; // increment
}
```

condition-এর মধ্যে Logical poperator ব্যবহার করা যায়। যেমন-

```
    j =0;
for (i=0; i<10 && j!=90; ++i)
```

অসীম for লুপ (Infinite for loop)

অসীম লুপ তৈরিতে for ব্যবহার করা হয়। এই অসীম (infinite) লুপ তৈরির জন্য for এর মধ্যে কোন initialization, increment ও condition দেয়া যাবে না। যেমন

```
For (; ; )
  {
    printf ("\n finite Loop \n");
  }
```

এই প্রোগ্রামটি Run করালে for loop থেকে সাধারণভাবে বের হওয়া যাবে না। প্রোগ্রাম থেকে বের হবার জন্য ctrl-Break একসাথে প্রেস করতে হবে।

while স্টেটমেন্ট

while স্টেটমেন্ট-কে while loopও বলা হয়। while স্টেটমেন্ট লেখার নিয়ম হল–

```
While (condition)
    statement;
```

condition হল যে কোন সঠিক C expression এবং এখানে সাধারণত relational expressi ব্যবহার করা হয়। while স্টেটমেন্টে প্রথমেই condition এর মান নির্ণয় করা হয়। condition-এর মান True হয় তবে while এর statement execute হয় কিন্তু, condition false হলে stateme execute হয় না।

নিম্নের while.c প্রোগ্রামটি লক্ষ্য করুন

Program **while.c**

```
# include <stdio.h>
main ( ) {
        int i;
        i = 1;
        while (i<= 5)
        {
            printf ("%d,", i);
            i ++;
        }
    getch ( );
     }
```

OUTPUT 1, 2, 3, 4, 5

while.c প্রোগ্রাম বিশ্লেষণ

- প্রথমে i কে initialize করা হয়েছে, i = 1.
- whle (i<=5)

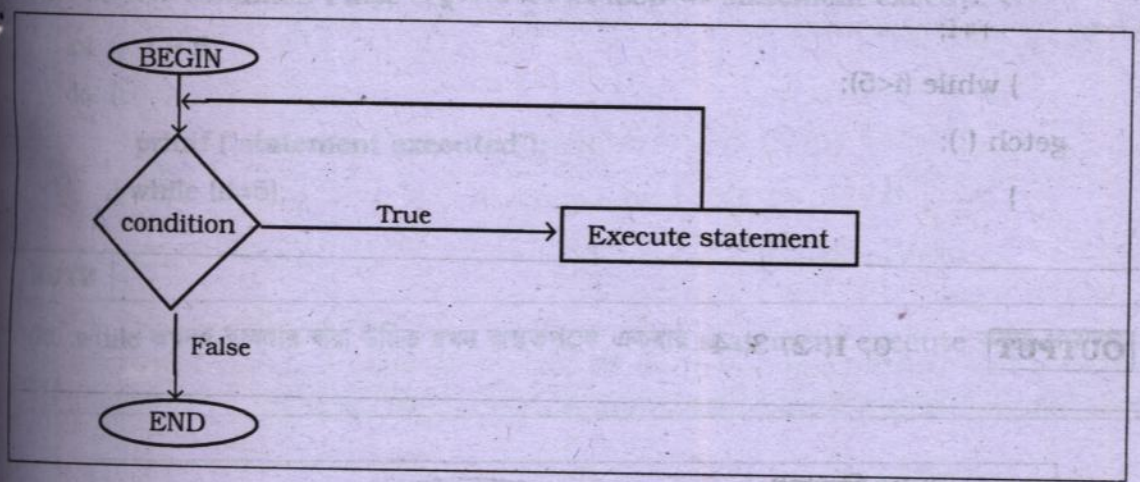
while প্রথমে i <=5 expression টি True তাই while-এর body execute হবে।

- i এর মান প্রিন্ট হবে এবং i এর মান 1 বৃদ্ধি পেয়ে 2 হবে।
- পুনরায় while এর condition পরীক্ষিত হবে। এবার i<=5 True হবে। আবার while এর execute হবে।
- এভাবে 1 এর মান বেড়ে 6 হলে i<=5 false হবে এবং তখন while লুপ শেষ হবে।

NOTE

while এর condition false হলে while এর body একবারও execute হবে না। যেমন ঃ

```
i = 5;
while (i< = 4)
    {
        printf (%d", i);
        ++i;
    }
```



NOTE

While (i) ব্যবহার করার চেয়ে while (! = 0) ব্যবহার করা ভাল

do.....while স্টেটমেন্ট

do....while লুপ এর condition -এর মান লুপের শেষে নির্ধারণ করা হয়। এটা লেখার নিয়ম হল–

```
do
  {
   statement;
  } while (condition);
```

নিম্নের dowhile.c প্রোগ্রামটি লক্ষ্য করুন

Program **dowhile.c**

```
# include <stdio.h>
main ( ) {
    int i = 0;
    do
     {
        printf ("%d",i);
        ++i;
     } while (i<5);
  getch ( );
     }
```

OUTPUT 0, 1, 2, 3, 4

dowhile.c প্রোগ্রাম বিশ্লেষণ

❑ do

```
{
    printf ("%d,", i);
    ----------
    ----------
```

এখানে প্রথমে i এর মান 0 প্রদর্শিত হবে।

❑ ++i;

```
} while (i<5);
```

এখানে i এর মান বেড়ে 1 হবে এবং condition check হবে। i<5 এখানে True তাই আবার pr
ফাংশন execute হবে এবং 1 প্রদর্শিত হবে।

❑ এভাবে i এর মান বেড়ে যখন 4 হবে তখন condition 4<5 true হবে এবং আবার printf ( ) ফাংশন execute হবে এবং 4 প্রদর্শিত হবে। এবার 4 এর মান বেড়ে 5 হবে এবং 5<5 false হবে। তাই do....while লুপ শেষ হবে।

NOTE

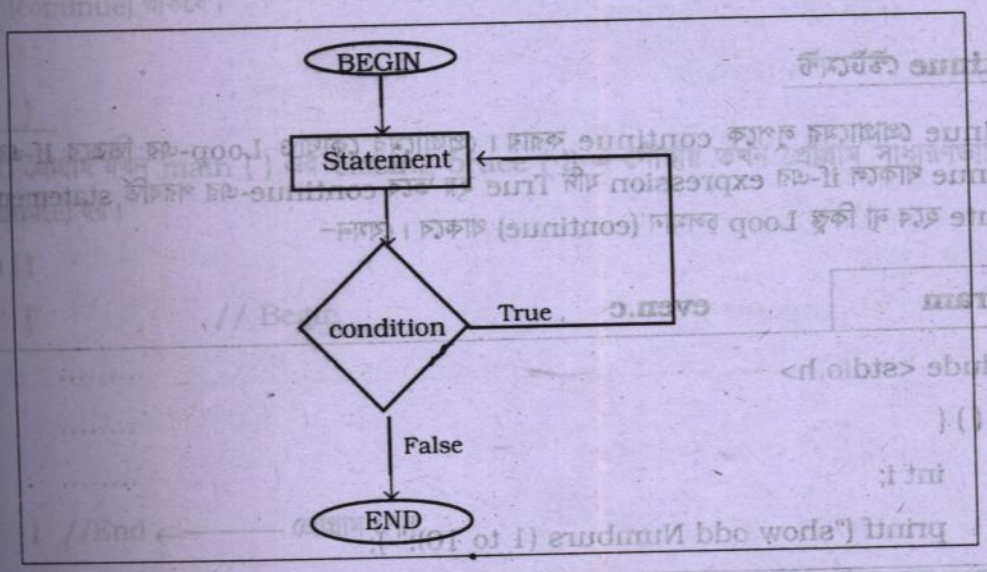
```
do {
    ....
    ....
    } while; /* Must put a semicolon here*/
```

নিম্নের প্রোগ্রামের condition False কিন্তু তাও একবার loop-এর statement execute হবে–

```
int     i = 5;
do {
      printf ("statement executed");
    } while (i!=5);
```

NOTE

do...while তখনই ব্যবহার করা উচিত যখন অন্তত পক্ষে একবার statement execute করার প্রয়োজন হয়।



চিত্র ঃ do...while-এর flow chart

NOTE

do.....while লুপকে exit-control loop বলা হয় কারণ এখানে লুপের শেষে condition ch
করা হয়।
while লুপকে entry-cotrol loop বলা হয় কারণ এখানে লুপের প্রথমেই condition check করা

break স্টেটমেন্ট

কোন লুপ থেকে বা switch statement থেকে তাৎক্ষণিকভাবে বের হবার জন্য break ব্যবহার কর
যেমন–

```
For (i =0; ; i++)
    {
      if (i = = 10) // Test condition
          break; // Exit from Loop
      printf ("%d", i);
    }
```

এখানে i এর মান 10 হলে Loop execution বন্ধ হবে। for এর মধ্যে condition দেবার প্রয়োজ
কারণ if (i==10) ব্যবহার করে for-এর condition-এর চাহিদা পূরণ করা হয়েছে।

continue স্টেটমেন্ট

continue প্রোগ্রামের লুপকে continue করায়। প্রোগ্রামের কোথাও Loop-এর ভিতরে If-
continue থাকলে if-এর expression যদি True হয় তবে continue-এর পরবর্তি stateme
execute হবে না কিন্তু Loop চলমান (continue) থাকবে। যেমন–

Program **even.c**

```
# include <stdio.h>
main ( ) {
      int i;
      printf ("show odd Numburs (1 to 10):" );
```

Program **even.c** Continue

```
for (i = 1; i<=10; i++)
  {
  if (i%2==0) continue;
  printf ("%d", i);
  }
getch ( )
}
```

OUTPUT 3, 5, 7, 9

cven.c প্রোগ্রাম বিশ্লেষণ

❑ it (i%2==0) continue;

এখানে i কে 2 দিয়ে ভাগ দিলে ভাগফল যদি '0' হয় তবে printf ( ) ফাংশন execute হবে না। কিন্তু Loop চলমান (continue) থাকবে।

exit ( )

কোন C প্রোগ্রাম যখন main ( ) এর closing brace (}) এ পৌছায় তখন প্রোগ্রাম সাধারণভাবে শেষ (terminate) হয়।

```
main ( )
  {          // Begin
    .....
    .....
    .....
  } //End ←——— প্রোগ্রাম শেষ
```

কিন্তু, exit ( ) ফাংশনের মাধ্যমে প্রোগ্রাম যে কোন সময় শেষ করা যায় অর্থাৎ প্রোগ্রাম থেকে বের হওয়া যায়।

exit ( ) ফাংশন লেখার সাধারণ গঠন। নিম্নরূপ–

```
exit (int status);
```

এখানে status হল integer টাইপ মান। status যদি '0' হয় তবে প্রোগ্রাম normal (সাধারণ) ভাবে শেষ হবে। যেমন— exit (o); status যদি '0' না হয় তবে বুঝতে হবে প্রোগ্রাম execution-এ কোন রকম ভুল আছে। যেমন— exit (1);

exit ( ) ফাংশনটি stdlib.h হেডার ফাইলে define করা থাকে। নিম্নের exit.c প্রোগ্রামে case স্টেটমেন্টে exit (0) ব্যবহার করা হয়েছে।

Program **exit.c**

```
# include <stdio.h>
main ( ) {
    char ch;
    ch = getchar ( );
    switch (ch) {
        case '1' : printf ("ONE");
                   break;
        case '2' : printf ("TWO");
                   break;
        case '3' : exit (0); /* Exit from program*/
    }
  getch ( );
  } /* End of pragram*/
```

Example :

নিম্নের প্রোগ্রামটি 0 থেকে 255 পর্যন্ত মানের ক্যারেকটারের ASCII টেবিল প্রদর্শন করবে।

Program **ASCII.c**

```
# include <stdio.h>
main ( ) {
      int ascii;
      for (ascii = 0; ascii<256; ascii ++)
      printf ("%3d=%c\t", ascii, ascii);
      getch ( );
      }
```

Example :

Serics1.c প্রোগ্রামটি নিম্নের সংখ্যারাশির মান নির্ণয় করবে।

$1+2^2+3^2+4^2+5^2 = ?$

Program **series1.c**

```
# include <stdio.h>
main ( ) {
      int i, total = 0;
      for (i=1; i<=5; ++i)
          total = total + i*i;
      printf ("Total = %d", total);
      getch ( );
      }
```

Example :

আমরা জানি !5 = 5*4*3*2*1 অর্থাৎ factorial n-এর মান হল

!n = n! (n–1)

এ সূত্রটি নিম্নের প্রোগ্রামে রূপান্তরিত করা হল

Program **fac.c**

```
# include <stdio.h>
main ( ) {
    int factorial;
    long total;
    factorial = 1;
    while (factorial ! = 0) /* Type 0 to exit*/
    {
        printf ("Enter a number for Factorial :");
        scanf ("%d", & factorial);
        total = 1;
        while (factorial> 1)
        total = total * factorial - - ;
        printf ("\n Factorial is : % ld", total);
    }
getch ( );
    }
```

Example :

নিম্নের aldisp.c প্রোগ্রামটি keyboard থেকে ক্যারেকটার input নেয় এবং ক্যারেকটারটি কোন সংখ্যা (digit) নাকি বর্ণ (alphabet) কিংবা অন্য কোন special ক্যারেকটার তা বলে দেয়।

Program **aldisp.c**

```
# include <stdio.h>
main ( ) {
      char ch;
      printf ("Type a character");
      scanf ("%c", & ch);
if ((ch> = 'a' && ch <='z') || (ch> = 'A' && ch <='Z'))
   printf ("\n you type an Alphabet");
else if (ch>'o' && ch <='9')
   printf ("\n you type Digit");
else
   printf "(\n you type special character");
getch ( );
```

Example :

নিম্নের প্রোগ্রামটি একটি ক্যালকুলেটর। এখানে প্রথমে operator (+, –,*, /) ইনপুট দিতে হবে এবং পরে দুটি সংখ্যা দিতে হবে। প্রোগ্রামটি operator অনুযায়ী সংখ্যা প্রোসেস করবে।

Program **calc.c**

```
# include <stdio.h>
main ( ) {
   float i = 0, j =0;
   char operator;
   printf ("Type a Operator (+, –, *,/) :");
   scanf ("%c", & operator);
   printf ("\n Type 2 Number");
```

continue

Program calc.c

```
scanf ("%f %f", & i, &j);
switch (operator)
    {
    case '+'    :  printf ("=%f", i+j);
                   break;
    case '–'    :  printf ("=%f", i–j);
                   break;
    case '*'    :  printf ("=%f", i*j);
                   break;
    case '/'    :  printf (" = %f", i/j);
                   break;
    default     :  printf ("wrong Operator");
    }
}
```

continue

Example :

কোন সংখ্যাকে যদি কেবল সেই সংখ্যা (নিজে) বা 1 দ্বারা ভাগ করা ছাড়া অন্য কোন সংখ্যা দিয়ে ভাগ করা না যায় তবে সেই সংখ্যাকে মৌলিক সংখ্যা বা prime number বলে।

যেমন ঃ 2 3 5 7 11 13 17 19 23 29 31 ইত্যাদি।

নিম্নের prime.c প্রোগ্রামটি 50 এর আগের prime number দেখাবে।

Progsam prime.c

```
# include <stdio.h>
main ( ) {
    int prime, flag, digit;
    for (prime = 2; prime <=50; ++prime)
        {
        flag = 1;
```

continue

Progsam

```
        for(digit=2; di
            if (prime
                flag
    if (flag!=0)
            printf ("%
    }
getch ( );
}
```

OUTPUT 2, 3,

Example :

নিম্নের প্রোগ্রাম রান করে এ
ক্যারেকটার এবং শব্দের সংখ্যা

Program

```
#include <stdio.h>
main ( ) {
    int .char_count
    int word_count
    char ch;
    printf ("Type a
    while ((ch = ge
```

Progsam **prime.c** continue

```
        for(digit=2; digit <prime; ++digit)
            if (prime%digit==0)
                flag =0;
        if (flag!=0)
            printf ("%d,", prime);
    }
  getch ( );
  }
```

OUTPUT 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

Example :

নিম্নের প্রোগ্রাম রান করে এক লাইনে একটি বাক্য (sentence) লিখে Enter চাপ দিলে প্রোগ্রামটি ক্যারেকটার এবং শব্দের সংখ্যা নির্ণয় করে দেখাবে।

Program **wecount.c**

```
#include <stdio.h>
main ( ) {
    int.char_count =0;
    int word_count =0;
    char ch;
    printf ("Type a sentence in One Line \n");
    while ((ch = getche ( ))! = '\r')
```

continue

Program wecount.c

```
        {
            char_count ++;
            if (ch = = ' ')
                word_count ++;
        }
    printf ("\n Total character is %d", char_count);
    printf ("\n Total word is %d", word_count+1);
    getch ( );
    }
```

Example :

নিম্নের প্রোগ্রামটি দুটি সংখ্যার গ. সা. গু নির্ণয় করবে

Program divisor.c

```
# include <stdio.h>
main ( ) {
        int p,q, temp;
printf ("Enter 2 Number (nonnegative int) : ");
scanf ("%d % d", & p, & q);
while (q!= 0)
    {
        temp = p%q;
        p=q;
        q= temp;
    }
```

continue

Program

```
printf ("\n Greatest
getch ( );
  }
```

INPUT/OUTPUT

Example :

নিম্নের প্রোগ্রামটি সংখ্যা ir
দেখাবে।

Program

```
# include <stdio.h
main ( ) {
    int num, digit;
    prinf ("Enter se
    scanf ("%d", &
    do
        {
            digit = nu
            printf ("%
            num = nu
        } while (nur
    printf ("\n");
    }
```

Program **divisor.c** continue

```
printf ("\n Greatest common devisor is : %d", p);
getch ( );
  }
```

INPUT/OUTPUT

Enter 2 number (nonnegative int) : 150 35

Greatest common dvisor is : 5→ গ. সা. গু

Example :

নিম্নের প্রোগ্রামটি সংখ্যা input নিবে এবং তা উল্টা করে দেখাবে। যেমন ঃ 1234 লিখলে প্রোগ্রামটি 4321 দেখাবে।

Program **reverse.c**

```
# include <stdio.h>
main ( ) {
   int num, digit;
   prinf ("Enter sequence of number");
   scanf ("%d", & num);
   do
      {
        digit = num % 10;
        printf ("%d", digit);
        num = num/10;
      } while (num!=0);
   printf ("\n");
   }
```

Example :

$ax^2+bx+c = 0$ হল একটি Quadratic Equation এবং একে নিম্নোক্তভাবে দেখানো যায়–

$$\text{Root value} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

নিম্নের প্রোগ্রামে $\pm \sqrt{b^2-4ac}$ এর জন্য (+) ও (–) উভয় চিহ্ন ধরে মান নির্ণয় করা হল ঃ

Program **quadra.c**

```
# include <stdio.h>
# include <math.h>
main ( ) {
    float a, b, c, r1, r2, d;
    printf ("Enter values for a, b and c \n");
    scanf ("%f %f %f", & a, & b, &c);
    d = b*b – 4*a*c;
    if (d<0) ("\n Imaginary Value");
    else {
        r1 = (–b+sqrt (d))/ (2.0*a);
        r2 = (–b-sqrt (d))/(2.0*a);
        pintf ("\n Root value for (+) :%f", r1);
        printf ("\n Root value for (–) : % f", r2);
        }
getch ( );
  }
```

Q&A:

1.Q. নিম্নের প্রোগ্রাম ভুল কোথায়?

```
main ( ) {
    int x;
    printf ("Enter Value");
    scanf ("%d", &x);
    if (k=2)
        printf ("\n Value is 2");
    else
        printf ("\n Value is not 2");
    }
```

উত্তর ঃ এখানে if (x=2) লেখা হয়েছে। এর অর্থ হল x-এর মান 2, কারণ = দ্বারা ভেরিয়েবল এর মান বসানো (assign) হয়।

এখানে আমরা input হিসেবে যে মানই দেই না কেন, প্রোগ্রাম output সব সময় value is 2 আসবে কারণ if (nonzero) সবসময় true হয়। এখানে x==2 লিখতে হবে।

2.Q. **While** এবং **do....while** লুপ এর মূল পার্থক্য কি?

উত্তর ঃ while লুপ এর condition প্রথমেই check হয় এবং condition false হলে লুপ execute হয় না। কিন্তু do....while লুপ অন্ততপক্ষে একবার execute হবে, যদিও condition false হয় কারণ এখানে condition পরে check হয়।

3.Q. **for** লুপের মধ্যে **variable**-এর মান কি শুধু **1** করে বারানো যায়?

উত্তর ঃ না, নিজের ইচ্ছে মত বাড়ানো যায়, যেমন–

```
for (i=1; i<10; i=i+2) // increase by 2
for (i=0.1; i<1; i=i+0.1) // increase by 0.1
```

4.Q. **for** লুপের মধ্যে ভেরিয়েবল এর মান কি কমানো **(decrement)** যায়?

উত্তর ঃ হ্যাঁ, যেমন–

```
for (i=10; i>0; --i) // decrease by 1
```

5.Q. নিম্নের প্রোগ্রামের ভুল কি?

```
i = 0;
while (i<10)
```

```
{
printf ("Value of i=%d", i);
printf ("In Next value");
}
```

উত্তর ঃ এখানে i এর মান 0 দেয়া হয়েছে কিন্তু while লুপের মধ্যে i এর মান বাড়ানো হয়নি, তাই i কখনো 10 বড় হবে না এবং প্রোগ্রাম অসীম হবে। এটা হল infinite loop. এখানে while লুপের মধ্যে ++i লিখতে হবে।

Exercise

1. নিম্নের গাণিতিক রাশিটির মান করার জন্য একটি প্রোগ্রামকরুন।

$$y = \frac{1}{1!} + \frac{2}{2!} + \frac{3}{3!} + \frac{4}{4!} + \frac{5}{!5}$$

2. for লুপের মান কিভাবে decrement করা যায়?
3. Nested লুপ কি?
4. π এর মান বের করার নিম্নের সূত্রটি প্রোগ্রাম আকারে লিখুন।

$$\frac{\pi}{4} = 1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + \ldots\ldots$$

5. নিম্নের **output** পাওয়ার জন্য একটি প্রোগ্রাম লিখুন।

```
1  2  3  4  5  6  7  8  9  10 11
1  2  3  4  5     7  8  9  10 11
1  2  3  4           8  9  10 11
1  2  3                 9  10 11
1  2                       10 11
1                          11
```

6. switch statement এর মাধ্যমে এমন একটি প্রোগ্রাম করুন যার মাধ্যমে keyboard-এর S,R,G,M key গুলো প্রেস করলে computer-এর speaker এ সা, রে, গা, মা শব্দ হবে।
7. নিম্নের for লুপটিকে কি বলা হয়?

```
for (; ;)
  {
    .....
    .......
  }
```

8. else if ladder কি?

9. $\frac{1}{1-x} = 1 + x + x^2 + x^3 + ... + x^n$ এর যোগফল নির্ণয় করুন।

10. break ও continue-এর পার্থক্য কি?

11. goto statement কেন ব্যবহার করা উচিত নয়?

12. নিম্নের সূত্রানুসারে Binomial coefficient-এর মান নির্ণয় করুন।

$$B(m,x) = \frac{!m}{!x!(m-x)}, \quad m> = x \ [B(0,0)=1]$$

13. NULL statement কি? ব্যাখ্যা করুন।

14. আমরা জানি কোন সংখ্যা x হলে x-এর পরম মান হল ঃ

av = x

এর জন্য একটি প্রোগ্রাম লিখুন।

15.
```
int i = 1;
if (i) {
    printf ("if");
    i = 0;
    }
if (!i) {
    printf ("!i");
    i = 1
    }
else
    printf ("i");
```

উপরের প্রোগ্রামের output কি হবে?

16. Exit control loop ও Entry control loop-এর পার্থক্য কি?

17. Flow chart -এ conditional statement কে কি চিহ্নের মাধ্যমে প্রকাশ করা হয়?

18. switch -এর মধ্যে সর্বোচ্চ কতটি case statement দেখা যায়।

19. Nested switch statement কি?

5

Array (অ্যারে)

- **Array**
- **Two dimensional array**
- **Three dimensional array**
- **Examples**

array

array হল একই টাইপ একাধিক ভেরিয়েবল এর সংগ্রহ (collection) যার জন্য মেমোরীতে পরস্পর সংলগ্ন byte নির্দিষ্ট (allocate) হয় এবং এদের একই নাম থাকে।

array ডিকলেয়ার করার সাধারণ নিয়ম নিম্নরূপ–

```
data_type array_name [size];
```

data_type বলতে array কোন্ টাইপ হবে তা বুঝায় (int, char ইত্যাদি)। একটি array মোট কতগুলো ভেরিয়েবল (element)-এর সমন্বয়ে গঠিত হবে তা size দ্বারা নির্দিষ্ট করা হয়।

উদাহরণ ঃ

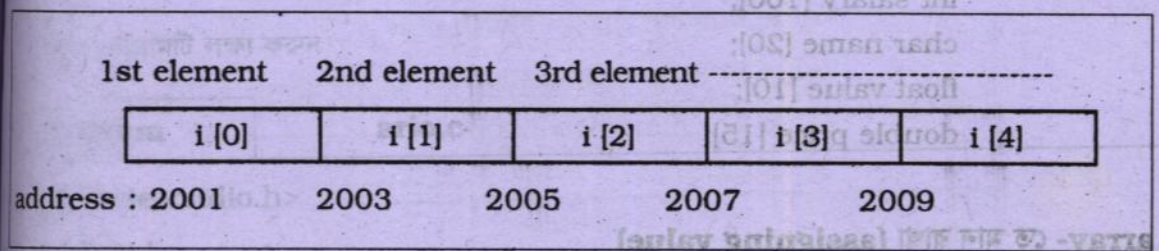
```
int i [5]
```

এখানে, i একটি int টাইপ array এবং এই array-তে মোট 5 টি element রয়েছে।

কোন array-এর প্রতিটি element-কে একটি সংখ্যা দ্বারা চিহ্নিত করা হয়। এবং একে index বা subscript বলা হয়। array element শুরু হয় zero থেকে।

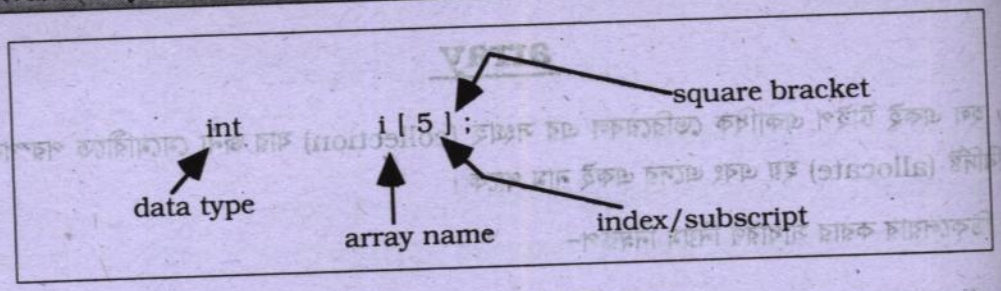
```
i [0]
```

এখানে, i [0] হল i [5] array-এর প্রথম element এবং i [0] -এর জন্য কম্পাইলার i [5] array-এর জন্য রক্ষিত মেমোরীর প্রথম অংশ নির্দিষ্ট করবে।



চিত্র ঃ array element গুলোর পরস্পর সংলগ্ন মেমোরী (byte)

কোন array-এর n সংখ্যক element থাকলে 0 থেকে n-1 পর্যন্ত subscript ব্যবহার করা যায়, i [5] array-এর i [0] থেকে সর্বোচ্চ i [4] পর্যন্ত ব্যবহার করা যাবে। i [5]-এর মধ্যে কোন মান রাখলে প্রোগ্রাম রান করবে তবে ভুল output দেখাবে।



চিত্র ঃ array-এর বিভিন্ন অংশ

NOTE

C কম্পাইলার array element-এর সর্বোচ্চ সীমা check করতে পারে না।

array কেন ব্যবহার করা হয়?

মনে করি, আমরা কোন প্রোগ্রামে ৩০ দিনের তাপমাত্রা আলাদাভাবে সংরক্ষণ করব এবং মাসের শেষে তার গড় তাপমাত্রা নির্ণয় করব। এজন্য আমাদের মোট ৩০টি ভেরিয়েবল ডিকলেয়ার করতে হবে। যেমন–

int first_day, second_day, third_day.....thirty_day; কিন্তু, এভাবে ভেরিয়েবল ডিকলেয়ার করে প্রোগ্রাম করা সম্পূর্ণ ভুল। কারণ ৩৬৫ দিনের গড় নির্ণয়ের জন্য ৩৬৫ টি ভেরিয়েবল ডিকলেয়ার করতে হবে। এক্ষেত্রে array ব্যবহার করে সংক্ষেপে কাজ করা যায়।

array ডিরলেয়ারেশন

যে কোন ডাটা টাইপের array ডিকলেয়ার করা যায়। যেমন ঃ

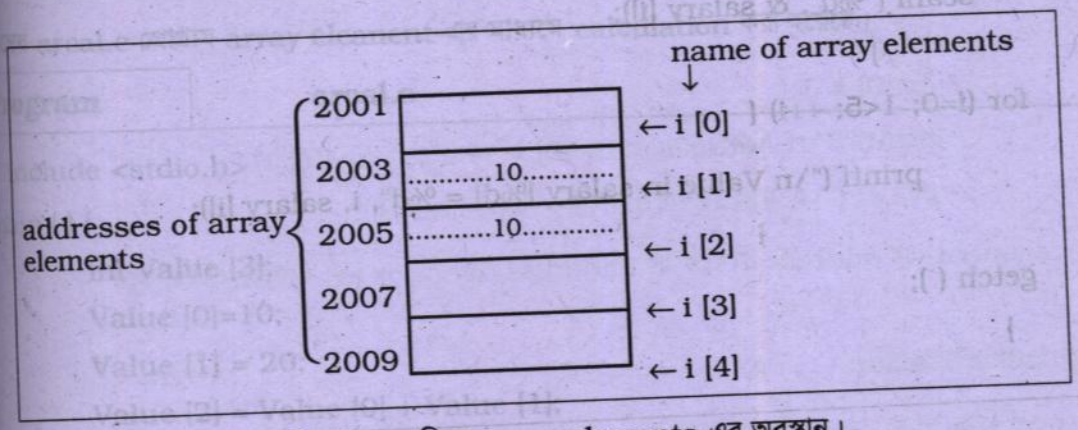
```
int salary [100];
char name [20];
float value [10];
double price [15];
```

array- তে মান রাখা (assigning value)

কোন array element প্রোগ্রামের যে কোন স্থানে ব্যবহৃত হতে পারে ঠিক সাধারণ ভেরিয়েবল এর মত। যেমন ঃ

```
int i [5];
i [2] = 10;
i [1] = i [2];
```

এখানে i[2] array element-এর মধ্যে 10 থাকবে এবং i [1] এর মধ্যেও 10 থাকবে কারণ i [1] = i [2]। মেমোরীতে এদের অবস্থান হবে নিম্নরূপ ঃ



চিত্র ঃ মেমোরিতে array elements-এর অবস্থান।

NOTE

i[5] যেহেতু int টাইপ array, তাই i[5]-এর প্রতিটি element-এর জন্য 2 byte করে স্থান নির্দিষ্ট হবে।

array তে ডাটা ইনপুট নেয়া ও প্রিন্ট করা ঃ

কোন array ভেরিয়েবল -এ ডাটা ইনপুট নেয়ার জন্য scanf ( ) ফাংশন ব্যবহৃত হয়। যেমন–

```
scanf ("%d", &i[0]);
```

নিম্নের প্রোগ্রামটি লক্ষ্য করুন

Program **ario.c**

```
#include <stdio.h>
main ( ) {
    int i;
    int salary [5];
    for (i=0; i<5; ++i) {
```

Continue

Program **ario.c** Continue

```
        printf ("\n Enter amount for salary [%d]:", i);
        scanf ("%d", & salary [i]);
            }
    for (i=0; 1<5; ++i) {
            printf ("\n Value in salary [%d] = %d", i, salary [i]);
                    }
    getch ( );
     }
```

INPUT/OUTPUT

```
Enter amount for salary [0]: 1000
Enter amount for salary [1] : 2000
Enter amount for salary [2] : 3000
Enter amount for salary [3] : 4000
Enter amount for salary [4] : 5000

Value in salary [0] = 1000
Value in salary [1] = 2000
Value in salary [2] = 3000
Value in salary [3] = 4000
Value in salary [4] = 5000
```

ario.c প্রোগ্রাম বিশ্লেষণ

❑ scanf ("%d", & salary [2];

এখানে i-এর মান যখন 0 তখন salary [0]-এ input নেয়া হচ্ছে। আবার, i-এর মান যখন 1 তখন salary [0]-এ input নেয়া হচ্ছে। এভাবে salay [4] পর্যন্ত input নেয়া হচ্ছে।

❑ printf ("Value in salary [%d] = %d", i, salary [i]);

এখানে printf ( ) এর মধ্যে salary [i] ব্যবহার করে প্রতিটি element-এর মান প্রিন্ট করা হয়েছে।

নিম্নের arcal.c প্রোগ্রাম array eleanent-এর মাধ্যমে calculation করা হয়েছে।

Program **arcal.c**

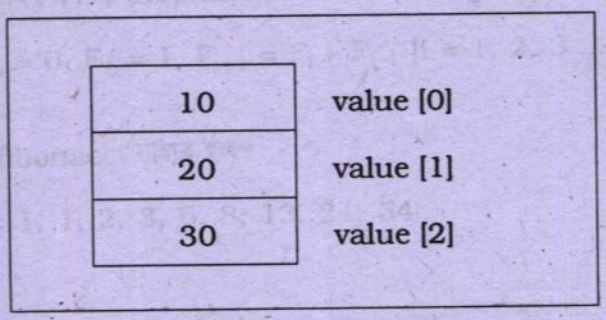
```
# include <stdio.h>
main ( ) {
        int Value [3];
        Value [0]=10;
        Value [1] = 20;
        Value [2] = Value [0] + Value [1];
        printf ("Value [0] + Value [1] = % d", Value [2]);
        getch( );
        }
```

OUTPUT Value [0] + Value [1] = 30

arcal.c প্রোগ্রাম বিশ্লেষণ

❑ Value [2] = Value [0] + value [1];

এখানে value [0] ও value [1]-এর মান যোগ হয়ে Value [2]-এ থাকবে।



চিত্র ঃ arcal.c প্রোগ্রাম বিশ্লেষণ।

array-এর address নির্ণয় ঃ

আমরা জানি, array-এর element গুলো পরস্পর সংলগ্ন থাকে এবং প্রতিটি element-এর নিজস্ব address থাকে। এই address আমরা দেখতে পারি।

```
int ar [2];
ar [0] = 1;
printf ("\n Address of ar [0] = % u", & ar [0]);
printf ("\n Value of ar [0] = % d", ar [0]);
```

array element-এর address দেখতে হলে address operator (&) ব্যবহার করতে হবে।

array initialization

যখন array ডিকলেয়ার করা হয় তখন এটাকে initialize করা যায় অর্থাৎ এর element মান নির্ধারণ করা যায়। যেমন ঃ

```
int series [4] = { 1, 2, 3, 4};
```

উপরের স্টেটমেন্টে array-এর size হল চার, কিন্তু, আমরা যদি array-এর size উল্লেখ না করি তবে কম্পাইলার তা বের করে নিবে। যেমন ঃ

```
int series [ ] = {1, 2, 3, 4};
```

এখানে কম্পাইলার হিসেব করবে মোট কতটি মান দেয়া হয়েছে এবং এই সংখ্যার সহিত 1 যোগ করে array-এর size নির্ণয় করবে।

```
int series [4] = { 1, 2};
```

উপরে, array-এর size দেয়া হয়েছে 4 কিন্তু মাত্র দুটির মান initialize করা হয়েছে। এখানে কম্পাইলার প্রথম দুটি element-এর মান 1 ও 2 নির্ধারণ করবে এবং পরের element গুলোর মান zero (0) বুঝাবে অর্থাৎ, এখানে

```
series [0] = 1;
series [1] = 2;
series [2] = 0;
series [3] = 0;
```

নিম্নের cloud.c প্রোগ্রামে ক্যারেক্টার array দেখানো হয়েছে ঃ

Program **cloud.c**

```
# include <stdio.h>
main ( ) {
        static char cloud [ ] = { 'C', 'L', 'O', 'U', 'D'};
        int i;
        for (i = 0; i <5; ++i)
        printf ("%c", cloud [i]);
        getch ( );
        }
```

OUTPUT CLOUD

NOTE

Character array সম্পর্কে string অধ্যায়ে বিস্তারিত আলোচনা করা হয়েছে।

Fibonacci Numbers

আমরা জানি, ফিবোনাক্কি (fibonacci) নাম্বার এর সিরিজ (series) হল ঃ

$$F_o = 0,\ F_1 = 1,\ F_{i+1} = F_i + F_{i-1}\ [i = 1, 2, 3.....]$$

প্রথম 10টি fibonacci নাম্বার হল–

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

অর্থাৎ, প্রতিটি fibonacci নাম্বার হল তার আগের দুটি সংখ্যার যোগফল। নিম্নে fiboaci.c প্রোগ্রামটি লেখা হল ঃ

Program **fibonaci.c**

```
# include <stdio.h>
main ( ) {
        int F [10], x;
        F [0] = 0; /* Fo is always 0*/
        F [1] = 1; /* F1 is always 1*/
        for (x=2; x<10; ++x)
            F [x] = F[x–2] +F[x–1];
        for (x=0; x<10; ++x)
            printf ("%d, ", F[x]);
        getch ( );
            }
```

OUTPUT o, 1, 1, 2, 3, 5, 8, 13, 21, 34

array size নির্ণয়

array তৈরির মাধ্যমে সর্বোচ্চ 64kb মেমোরী ব্যবহার করা যাবে। একটি array মোট কত byte স্থান দখল করে তা নির্ভর করে তার ডাটা টাইপ ও element-এর সংখ্যার উপর। নিম্নোক্ত প্রোগ্রামের মাধ্যমে কোন array-এর size নির্ণয় করা যাবে।

Program **arsize.c**

```
# include <stdio.h>
main ( ) {
      float farray [10];
```

Continue

Program **arsize.c** Continue

```
printf ("size of Float type array (10 elements) : %d bytes", sizeof (farray));
getch ( );
}
```

OUTPUT size of Float array (10 elements) : 40 bytes

NOTE

একটি float ভেরিয়েবল মেমোরীতে 4 byte স্থান দখল করে।

দ্বিমাত্রিক array (Two Dimensional Array) :

C প্রোগ্রামে Two Dimensional Array তৈরি করা যায়। একে অনেক সময় matrix বলা হয়। Two dimensional array-এর দু'টি subscript (index) থাকে। এর declaration এর নিয়ম হল ঃ

```
data_type array_name [row] [column];
```

এখানে row এবং column বলতে এদের size বোঝানো হচ্ছে।

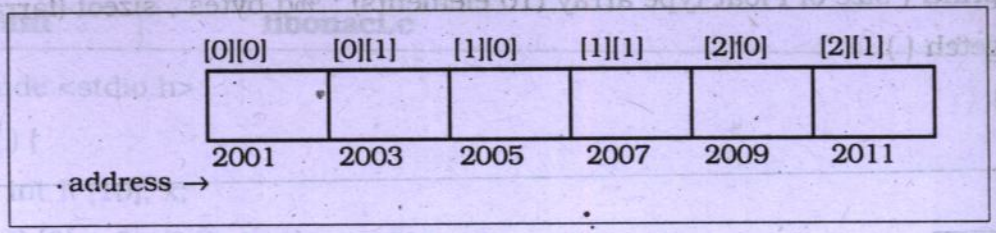
উদাহরণ ঃ `int table [3] [2];`

উপরের table একটি two dimensional array এবং এর row-এর সংখ্যা 3 টি এবং column হবে দু'টি। নিম্নের চিত্রের দিকে দেখা যাক–

		col.0	col.1
row	0	[0][0]	[0][1]
row	1	[1][0]	[1][1]
row	2	[2][0]	[2][1]

চিত্র ঃ দ্বিমাত্রিক array-এর স্বরূপ।

উপরের চিত্রে আমরা two dimensional array-এর মেমোরী দখলের যে চিত্র দেখছি তা আসলে বোঝার সুবিধার জন্য দেখানো হয়েছে। কিন্তু, মেমোরীতে এই array টি নিম্নোক্তভাবে গঠিত হবে ঃ



চিত্র ঃ মেমোরীতে দ্বিমাত্রিক array-এর সংগঠন।

2-Dimensional array initialization

দ্বিমাত্রিক array বিভিন্নভাবে initialize করা যায়। যেমন

```
int table [4] [2] = {
                    {1, 2},
                    {3, 4},
                    {5, 6},
                    {7, 8}
                  };
```

একে নিম্নোক্তভাবেও initialize করা যায়–

```
int table [4] [2] = { 1, 2, 3, 4, 5, 6, 7, 8};
```

Two dimensional array initialize করার সময় দ্বিতীয় index অবশ্যই উল্লেখ করতে হবে কিন্তু প্রথম index উল্লেখ না করলেও হবে। যেমন–

```
int table [ ] [2] = {1,2,3,4,5,6,7,8}; // correct.
```

নিম্নের প্রোগ্রামে two dimensional array ব্যবহার করে রোল নং ও নাম্বার input নেয়া হয়েছে এবং পরে তা screen-এ প্রদর্শিত হয়েছে।

Program **arrtwo.c**

```
# include <stdio.h>
# define ROLL 4
# define NUM 2
```

Continue

Program **arrtwo.c** Continue

```
main ( ) {
        int student [ROLL] [NUM];
        int     p;
        printf ("Type Roll and marks.");
        for (p=0; p<=3; p++)
    {
        scanf ("%d %d", & student [p] [o], & student [p] [1]);
    }
        for (p=0; p<=3; ++p)
    {
        printf ("\n%d,%d", student [p] [0], student [p] [1]);
    }
        getch ( );
    }
```

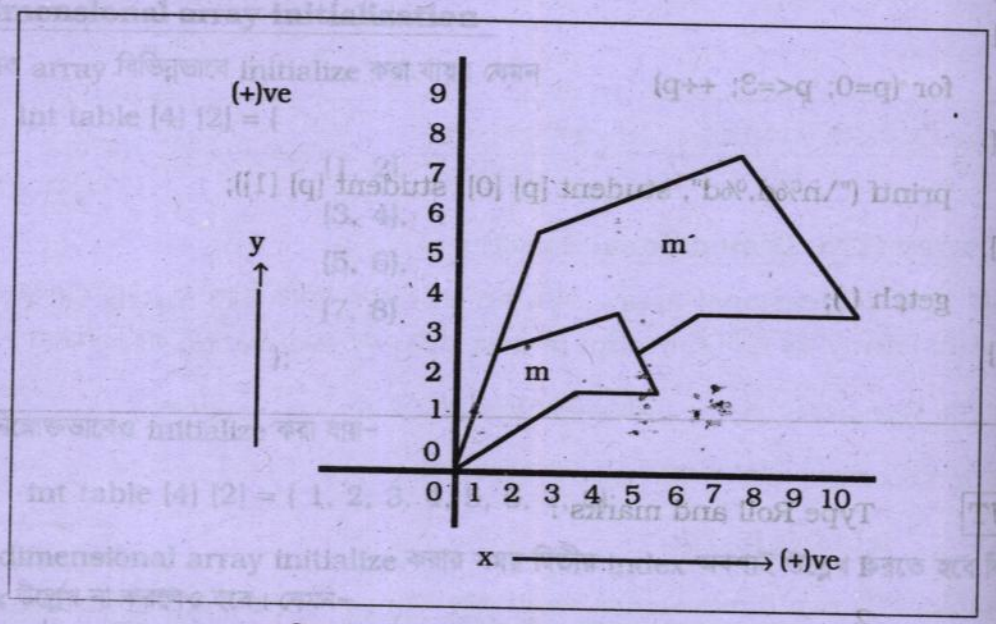
INPUT

```
Type Roll and marks :
1
2
3
4
5
6
7
8
```

OUTPUT

```
1,2
3,4
5,6
7,8
```

Multiply Matrix by scalar



চিত্র ঃ m হল এর বর্ধিত রূপ $\acute{m}$ ।

উপরের গ্রাফ-এ $\acute{m}$ হল m এর বর্ধিত রূপ। অর্থাৎ

$$\acute{m} = 2 * m$$

বা $\begin{bmatrix} 0 & 6 & 10 & 8 & 2 \\ 0 & 4 & 4 & 8 & 6 \end{bmatrix} = 2 * \begin{bmatrix} 0 & 3 & 5 & 4 & 1 \\ 0 & 2 & 2 & 4 & 3 \end{bmatrix}$

এখানে 2 হল scalar এবং কোন matrix কে কোন scalar দ্বারা বৃদ্ধি করাকে scaling বলে।

নিম্নের mscalar.c প্রোগ্রামটি scaling-এর কাজ করবে।

Program **mscalar.c**

```
# include <stdio.h>
main ( )   { int row, col, scalar;
             static int matrix [3] [5] = {
                                          { 1, 2, 3, 4, 5 },
                                          { 6, 7, 8, 9, 10 },
                                          { -1, 2, -3, 4, -5 }
                                         };
             printf ("Enter value for scalar");
             scanf ("%d", & scalar);
             for (row=0; row <3; ++row)
                  for (col =0; col <5; ++col)
                       matrix [row] [col] = matrix [row] [col] * scalar;
             printf ("\n RESULT is \n");
             for (row = 0; row <3; ++row)
                  for (col = 0; col <5; ++col)
                  {
                       printf ("%d", matrix [row] [col]);
                       printf ("\n");
                  }
             getch ( );
             }
```

INPUT/OUTPUT

```
Enter value for scalar : 2
RESULT IS
2 4 6 8 10
12 14 16 18 20
-2 4 -6 8 -10
```

NOTE

Matrix Multiplication প্রোগ্রাম example-এর মধ্যে বর্ণনা করা হয়েছে।

ত্রিমাত্রিক array (Three Dimensional Array) :

C প্রোগ্রামে three dimensional array নিয়ে কাজ করা যায়। মূলত three Dimensional array হল array-এর array। যেমন–

```
int table [10] [10] [10];
```

উপরোক্তভাবে কোন array ডিকলেয়ার করলে কম্পাইলার মেমোরীতে 10×10×10 = 1000 array elements-এর জন্য স্থান নির্দিষ্ট করবে।

মূলত একাধিক সংখ্যক two Dinmensional array নিয়ে three Dimensional array গঠিত।

নিম্নের arr_3d.c প্রোগ্রামটি লক্ষ্য করা যাক ঃ

Program **arr_3d.c**

```
# include <stdio.h>
main ( ) {      int i, j, k;
                int table [3] [4] [2] = {
                                        {1,2},
                                        {3,4},
```

Continue

Program **arr_3d.c** Continue

```
            {5,6},
            {7,8}
        },
        {
            {9,10},
            {11,12},
            {13,14}
            {15,16}
        },
        {
            {17,18},
            {19,20},
            {21,22},
            {23,24}
        }
        };
            for (i=0; i<3; ++i)
        {
            for (j=0; i<4: ++j)
        {
            for (k=0, k<2; ++k)
            printf ("\n Array [%d] [%d] [%d]=", i, j, k);
        } /* End for j*/
    } /* End for i*/
getch ();
}
```

OUTPUT

```
array [0] [0] [0] = 1
array [0] [0] [1] = 2
array [0] [1] [0] = 3
array [0] [1] [1] = 4
array [0] [2] [0] = 5
array [0] [2] [1] = 6
array [0] [3] [0] = 7
array [0] [3] [1] = 8
array [1] [0] [0] = 9
array [1] [0] [1] = 10
array [1] [1] [0] = 11
array [1[ [1] [0] = 12
array [1] [2] [0] = 13
array [1] [2] [1] = 14
array [1] [3] [0] = 15
array [1] [3] [1] = 16
array [2] [0] [0] = 17
array [2] [0] [1] = 18
array [2] [1] [0] = 19
array [2] [1] [1] = 20
array [2] [2] [0] = 21
array [2] [2] [1] = 22
array [2] [3] [0] = 23
array [2] [3] [1] = 24
```

arr_3d.c প্রোগ্রাম বিশ্লেষণ

- উপরের প্রোগ্রামে i, j, k ভেরিয়েবলগুলোর মাধ্যমে তিনটি for loop ব্যবহার করা হয়েছে।
- table [ ] [ ] [2]

এখানে প্রথমে ২টি element বিশিষ্ট একটি one dimensional array গঠিত হয়েছে। যেমন–

{1,2} ←———1 dimensional array

- table [ ] [4] [2]

এখানে চারটি one dimensional array একটার নীচে অন্যাটা বসে 2 dimensional array গঠিত হয়েছে, যেমন–

```
{
 { 1,2},
 {3,4},        চারটি 1dimensional array
 {5,6}
 {7,8}
}
```

- table [3] [4] [2]

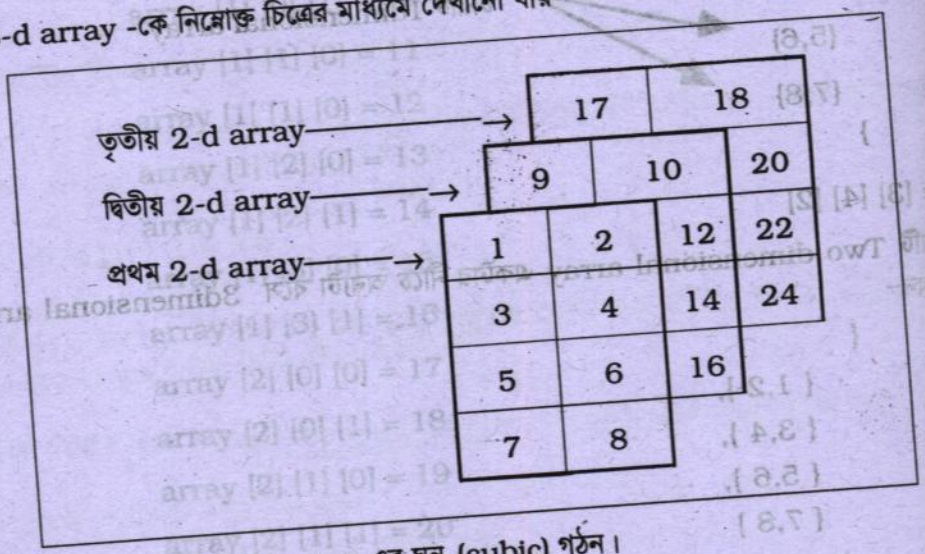
এখানে তিনটি Two dimensional array একটার নীচে অন্যটা বসে 3dimensional array গঠিত হয়েছে। যেমন–

```
{
   { 1,2 },
   { 3,4 },
   { 5,6 },
   { 7,8 }
  },
  {
   { 9,10 }          (3 টি 2-dimensional array)
   { 11,12 }
   { 13, 14 }
   { 15, 16 }
  },
```

Continue

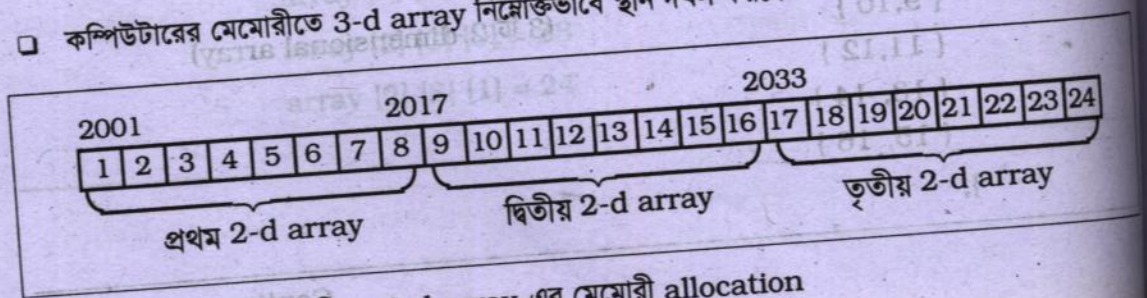
```
    {
      { 17, 18 },
      { 19,20 },
      { 21,22 },
      { 23,24 }
    }
  };
```

❑ উপরে 3-d array -কে নিম্নোক্ত চিত্রের মাধ্যমে দেখানো যায়—



চিত্র ঃ 3-d array-এর ঘন (cubic) গঠন।

❑ কম্পিউটারের মেমোরীতে 3-d array নিম্নোক্তভাবে স্থান দখল করবে—



চিত্র ঃ 3-d array-এর মেমোরী allocation

NOTE

STORAGE MAPPING FUNCTION : প্রোগ্রামে যখন array [a] [b] [c] ব্যবহার করা হয় তখন কম্পাইলার storage Mapping Function-এর সাহায্যে কোন নির্দিষ্ট array elemcnt-এ পৌছায়।

int array [3] [4] [2] কে নিম্নোক্তভাবে লেখা যায়–

int array [ ] [4] [2] বা int (* array) [4] [2]

Example :

Matrix Multiplication : গণিত ও গ্রাফিক্স প্রোগামিং-এ matrix multiplication খুবই গুরুত্বপূর্ণ একটি বিষয়। ধরাযাক

$$x = \begin{bmatrix} Ai_1 & Ai_2 \\ Bi_1 & Bi_2 \end{bmatrix} \text{ এবং } y \begin{bmatrix} Ai_3 & Ai_4 \\ Bi_3 & Bi_4 \end{bmatrix}$$

এখন x ও y-কে গুণ করে z-এর matrix-এর মধ্যে গুণফল রাখতে হবে। অর্থাৎ

z = x * y

mmulti.c প্রোগ্রামে দুটি 2-d matrix-এর গুণ দেখানো হল

Program **mmulti.c**

```
# include <stdio.h>
main ( ) {
        int x [2] [2] , y [2] [2], z [2] [2];
        int p, q, r, total;
        printf ("Enter 4 value for matrix X \n");
        for (p=0; p<2; ++p)
            for (q=0; q<2; ++q)
```

Continue

Program **mmulti.c** Continue

```
        scanf ("%d", &x [p] [q]);
printf ("\n Enter 4 value for Y \n");
for (p=0; p<2; ++p)
    for (q=0; q<2; ++q)
        scanf ("%d", & y [p] [q]);
printf ("\n Matrix Multiplication of X and Y is : \n");
for (p=0; p<2; ++p)
    {
        for (q = 0; q<2; ++q)
        {
            total = 0;
            for (r=0; r<2; ++r)
            total = total + x [p] [r] * y [r] [q];
            z [p] [q] = total;
            printf ("%d\t", z [p] [q]);
        }
printf ("\n");
    }
getch ( );
}
```

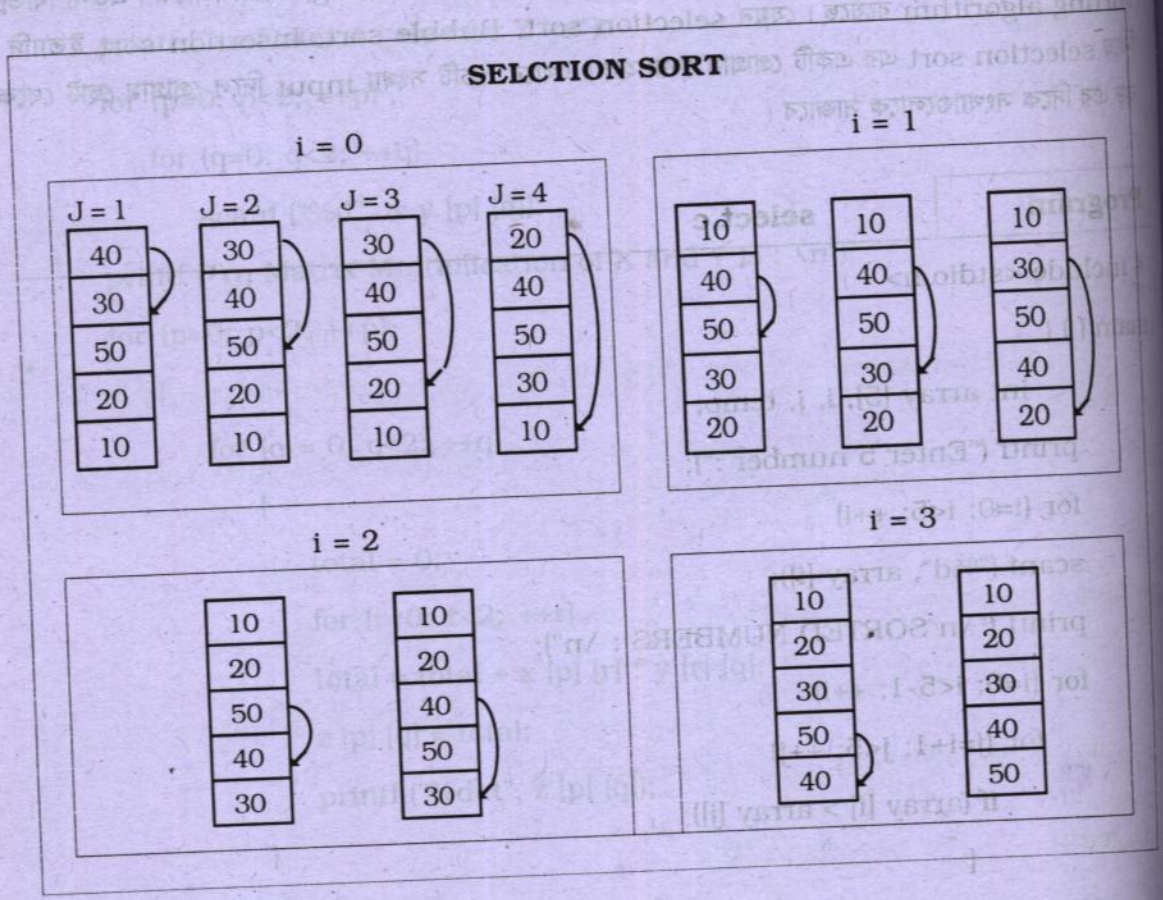
Example :

Selection sort : মনে করি পাঁচটি সংখ্যাকে ছোট থেকে বড় (ascending) করে সাজাবো। এজন্য বিভিন্ন sorting algorithm রয়েছে। যেমন selection sort, Bubble sort, Insertion sort ইত্যাদি। নিম্নে selection sort এর একটি প্রোগ্রাম দেয়া হল। এখানে পাঁচটি সংখ্যা input দিলে প্রোগ্রাম ছোট থেকে বড় এর দিকে সংখ্যাগুলোকে সাজাবে।

Program **select.c**

```
# include <stdio.h>
main ( ) {
    int array [5], i, j, temp;
printf ("Enter 5 number :");
for (i=0; i<5; ++i)
scanf ("%d", array [i]);
printf ("\n SORTED NUMBERS : \n");
for (i=0; i<5-1; ++i)
   for (j=i+1; j<5; ++j)
      if (array [i] > array [j])
       {
            temp = arry [i];
            array [i] = array [j];
            array [j] = temp;
        }
getch ( );
}
```

selction sort প্রোগ্রামে 5 টি মান দুটি লুপের মাধ্যমে যেভাবে ছোট থেকে বড় আকারে সজ্জিত হয়েছে তা নিম্নের চিহ্নে দেখানো হল ঃ

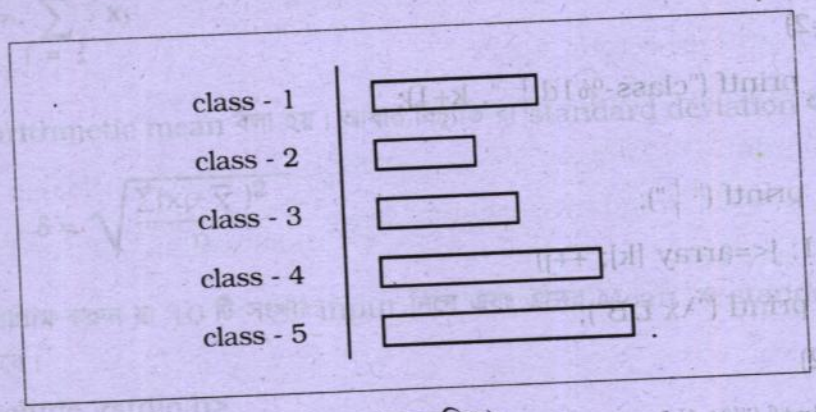


NOTE

Bubble sort এবং Quick sort সম্পর্কে function অধ্যায়ে আলোচনা করা হয়েছে।

Example :

Histgram : নিম্নের প্রোগ্রাম কোন স্কুলের পাঁচটি class-এর প্রতি class-এর ছাত্রদের মোট সংখ্যা input নিবে এবং Histogram তৈরি করে দেখাবে। যেমন ঃ



চিত্র ঃ Histogram (অনুভূমিক)

Program **Histo.c**

```
# include <stdio.h>
main ( ) {
    int array [5];
    int i, j, k;
    for (k=0; k<5; ++k)
      {
        printf ("\n Enter total student Namber in class-%d:", k+1);
        scanf ("%d", &array [k]);
      }
    printf ("\n");
    printf ("      |\n");
    for (k=0; k<5; ++k)
      {
```

Continue

Program **Histo.c** Continue

```
        for (i=1; i <=3, ++i)
    {
if (i==2)
        printf ("class-%1d ¦ ", k+1);
else
        printf (" ¦ ");
for (j=1; j<=array [k]; ++j)
        printf ("\x DB");
if (i==2)
     printf ("[%d] \n", array [k]);
else
       printf ("\n");
    }
       printf (" ¦ \n");
    }
getch ( );
}
```

Q & A :

Q.1. n সংখ্যক গাণিতিক সংখ্যার **mean** কে নিম্নোক্তভাবে প্রকাশ করা যায় ঃ

$$\bar{X} = \sum_{i=1}^{n} x_i$$

এখানে X কে arithmetic mean বলা হয়। আবার বিচ্যুতি বা standard deviation হল ঃ

$$\delta = \sqrt{\frac{\Sigma(x_i - \bar{X})^2}{n}}$$

এমন একটি প্রোগ্রাম করুন যা 10 টি সংখ্যা input নিবে এবং এদের Mean ও standard Deviation বের করে দেখাবে।

উত্তর ঃ

```
# include <stdio.h>
# includ <math.h>
main ( ) {
    int i;
    double num [10];
    double dev, total, total_sqr, mean, std_dev, variance;
    total = 0;
    total_sqr=0;
    printf ("Enter 10 positive values \n");
    for (i=1; i<=10; ++i)
      {
        scanf (% e", & num [i];
        total + = num [i]);
      }
    mean = total /10;
      for (i=1; i<=10; ++i)
      {
```

Continue

```
        dev = num [i] - mean;
        total_sqr + = dev * dev;
    }
variance = total_sqr/10;
std_dev = sqrt (variance);
printf ("\n Mean : %e \n", mean);
printf ("standard Deviation : %e \n", std_dev) ;
getch ( );
}
```

Q.2 কোন **array inttialize** না করে ব্যবহার করলে কি ঘটবে?

উত্তর ঃ array initialize না করে ব্যবহার করলে প্রোগ্রাম compile হবে এবং Run করবে, তবে ফলাফল হবে অপ্রত্যাশিত এবং ভুল যাকে garbage বলা হয়।

Q.3. int array [2] [3] = { {1,2},
{3,4},
{5,6} };

এখানে 3 কোন array element-এ আছে?

উত্তর ঃ array eement array [1] [0]

Q. 4. array এর ক্ষেত্রে **Bound checking** কি?

উত্তর ঃ নিম্নের প্রোগ্রাম code লক্ষ্য করুন

```
int i[5]; // array size is up to 5
i[6] = 1; // exceed array size
```

এখানে array i[5] এর সর্বোচ্চ সীমা 5 কিন্তু i [6] = 1 এ array এর ৭ম element ব্যবহার করা হয়েছে। এখানে array index-এর limit অতিক্রম করা হয়েছে। এ প্রোগ্রামটি compile এবং Run করবে কিন্তু অপ্রত্যাশিত ফলাফল দেখাবে। C/C++ compiler array এর সীমা (index limit) check করতে পারে না অর্থাৎ C/C++ কম্পাইলার array-এর Bound check করতে অক্ষম।

Exercise

1. আমরা জানি পাই radian = 180°। Degree কে Radian-এ রূপান্তরিত করার একটি প্রোগ্রাম করুন।

2. কোন গোলকের ব্যাসার্ধ r হলে তার আয়তন হবে

$$V = \frac{4}{3}\pi r3$$

এই classic সূত্রটি প্রোগ্রাম আকারে লিখুন।

3. C প্রোগ্রামে সর্বোচ্চ কত dimension পর্যন্ত array করা যায়?

4. Two dimension ও Three dimension অ্যারের মধ্যে তুলনা করুন?

5. নিম্নের প্রোগ্রামের ফলাফল কি হবে?

```
int i, a [3];
for (i=0; i<3; ++i)
    {
        a[i] = i;
        printf ("%d", a[i]);
    }
```

6. মনে করি x_i হল ধনাত্মক সংখ্যার সারি—

$$x_o = 1,\ x_{i+1} = \frac{1}{2}\ x_i + \frac{a}{x_i}\ [i = 0,\ 1,\ 2,\ 3.....]$$

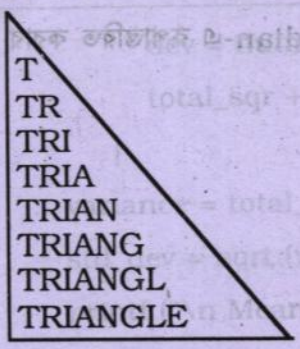
অর্থাৎ, $x_i \to \sqrt{a}$ যেখানে $i \to \alpha$

এই সূত্রটি হল কোন সংখ্যার বর্গমূল (square root) নির্ণয়ের জন্য বিজ্ঞানী Newton-এর প্রদেয় algorith. সূত্রটি প্রোগ্রামে রূপান্তরিত করুন।

7. $e = 1 + \frac{1}{!1} + \frac{1}{!2} + \frac{1}{!3} + \frac{1}{!4} + ...$

e-এর মান নির্ণয়ের জন্য একটি প্রোগ্রাম লিখুন।

8. এমন একটি program লিখুন যার output নিম্নরূপ হবে ঃ



9. নিম্নের statement-এ ভুল কি?

```
int array [5] [ ] [5];
```

10. পাঁচটি সংখ্যা input নিয়ে সবচেয়ে বড় সংখ্যা নির্ণয়ের জন্য একটি প্রোগ্রাম করুন।

11. Selection sort, Bubble sort এবং Quick sort-এর মধ্যে পার্থক্য কি?

12. নিম্নোক্ত array-এর মোট কতগুলো element আছে?

```
int a [5] [5] [5];  // 5*5*5
```

6

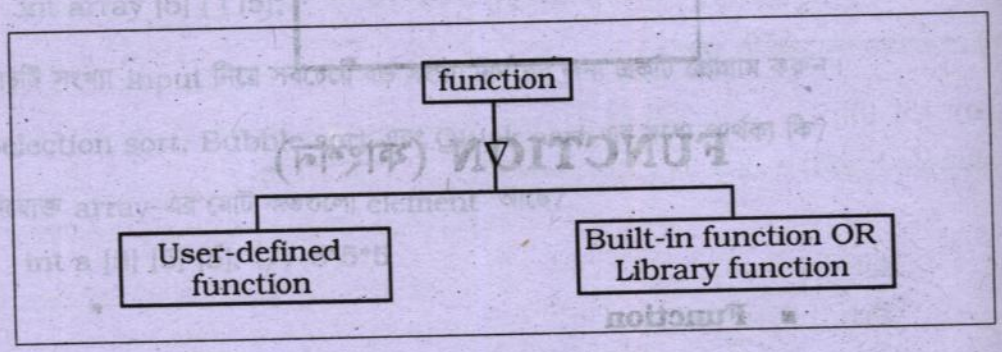
FUNCTION (ফাংশন)

- **Function**
- **Parameter**
- **Return**
- **Variable scope**
- **Structured programming**
- **main ( ) function**

FUNCTION

ফাংশন C প্রোগ্রামের একটি অতিপ্রয়োজনীয় এবং মৌলিক বিষয়। কোন প্রোগ্রাম যখন অনেক বড় হয় তখন সেই প্রোগ্রাম লেখা এবং ব্যবস্থাপনা (manage) করা অনেক কঠিন হয়ে যায়। আবার কখনো কখনো কোন প্রোগ্রামে একই কাজ বার বার করতে হয় এবং এজন্য একই statement একাধিকবার লেখার প্রয়োজন হয়। এসব সমস্যার সমাধানকল্পে প্রোগ্রামে ফাংশন গঠন করা হয়।

সংজ্ঞা ঃ C প্রোগ্রামে এক বা একাধিক statement-এর সমন্বয়ে ফাংশন গঠিত। প্রতিটি ফাংশন এর নির্দিষ্ট নাম থাকে এবং এরা প্রোগ্রামের নির্দিষ্ট কোন সমস্যা সমাধানের জন্য ব্যবহৃত হয়। ফাংশন দুই প্রকার। এই অধ্যায়ে User-defined function নিয়ে আলোচনা করা হয়েছে।



চিত্র ঃ function-এর প্রকারভেদ।

NOTE

Library ফাংশন নিয়ে পরে আলোচনা করা হয়েছে।

USER-DEFINED FUNCTION

আমরা জানি printf ( ), scanf ( ) এগুলো হল Library function. এ ধরনের আরো অনেক Library function রয়েছে c ল্যাংগুয়েজে। এগুলো আমরা যে কোন সময় ব্যবহার করতে পারি। কিন্তু, কখনো কখনো আমাদের নিজেদের function তৈরি করার প্রয়োজন হয়। আমরা নিজেরা নিজস্ব প্রয়োজনের জন্য যে ফাংশন তৈরি (define) করব তাই-ই হল User-defined function.

নিম্নে func1.c প্রোগ্রামে function সমন্ধে একটি সাধারণ ধারণা দেয়া হল।

Program **func1.c**

```
# include <stdio. h>
show ( )
   {
     printf ("This is function show ( )");
   }
main ( )
   {
     show ( );
     getch ( );
   }
```

OUTPUT This is function show ( )

func1.c প্রোগ্রাম বিশ্লেষণ

❑ show ( )

প্রতিটি ফাংশন এর একটি নির্দিষ্ট নাম থাকে। একে বলা হয় function header। এই নাম ধরেই ফাংশনকে শনাক্ত করা যায় এবং main ( ) এর মধ্যে একে ব্যবহার (call) করা যায়। এখানে show ( ) হল ফাংশনের নাম।

❑ show ( )
   {
     printf ("This is function show ( )");
   }

কোন্ ফাংশন কি কাজ করবে তা ফাংশন নাম এর পর '{' এবং '}' এর মধ্যে বলে দিতে হবে। কোন ফাংশন নাম এর পর '{' এবং '}' এর মধ্যে অবস্থিত এক বা একাধিক প্রোগ্রাম statement-কে program body বলে। এখানে show ( ) ফাংশনটি screen-এ একটি বাক্য প্রদর্শন করবে।

❑ main ( )

main ( ) নিজেও একটি ফাংশন। কোন C প্রোগ্রামে আমরা যত ফাংশনই তৈরি করি না কেন, main ( ) ফাংশন অবশ্যই লিখতে হবে এবং এই main ( ) ফাংশন এর মধ্যে অন্য সব ফাংশন ব্যবহার (call) করতে হবে।

এখানে লক্ষণীয় যে, main-এর পূর্বে কোন ফাংশন লিখলেও main ( ) ফাংশন সর্বপ্রথম execute হয়।

❑ main ( ) {

```
    show ( );
    getch ( );
}
```

show ( ) ফাংশনটি main ( ) এর মধ্যে call করা হয়েছে। অর্থাৎ কম্পাইলার যখন show ( ) স্টেটমেন্ট (ফাংশন) execute করবে তখন প্রোগ্রামের নিয়ন্ত্রণ show ( ) ফাংশন-এ প্রতিস্থাপিত হবে এবং main ( ) ফাংশন এর কার্যক্রম স্থগিত হবে। show ( ) ফাংশন এর execution (নির্দিষ্ট কাজ) শেষ হলে প্রোগ্রামের নিয়ন্ত্রণ আবার main ( ) ফাংশন এ ফিরে আসবে এবং যেখান থেকে main ( ) এর excention স্থগিত হয়েছিল তার পরবর্তী statement থেকে প্রোগ্রাম execution শুরু হবে।

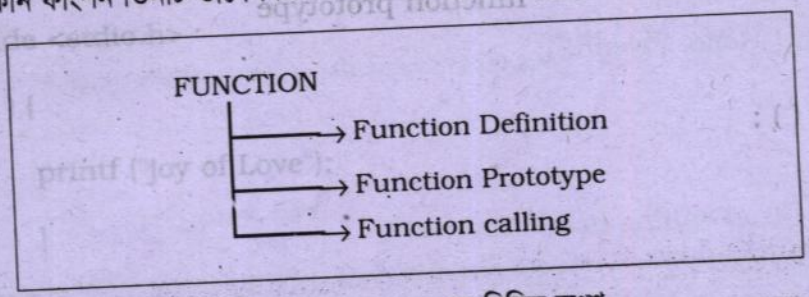
NOTE

কোন প্রোগ্রাম main ( ) ফাংশন না থাকলে কম্পাইলার error message দেখাবে। যেমন ঃ নিম্নোক্ত প্রোগ্রামটি Run করবে না কারণ main ( ) নেই।

```
# include <stdio. h>
show ( ) {
    printf ("show ( ) function");
}
```

ফাংশন-এর গঠন ঃ

C প্রোগ্রামে যেকোন ফাংশন তিনটি ভাগে বিভক্ত। এর প্রতিটি ভাগ নিয়ে নিম্নে বিস্তারিত আলোচনা করা হল।



চিত্র ঃ Function-এর বিভিন্ন অংশ

Function Definition (বা Function body)

কোন ফাংশন কি কাজ করবে, তা Function Definition-এর মধ্যে বর্ণনা করতে হয়। অর্থাৎ function definition-এর মধ্যে program code লিখতে হয়। যেমন ঃ

```
main ( ) {
      show ( ) ;
       }

show ( ) {
      printf ("hello programmar")
      }
```

Function definition

চিত্র ঃ ফাংশন ডেফিনেশন।

NOTE

Function Definition-এর গঠন নিয়ে পরে বিস্তারিত আলোচনা করা হয়েছে।

Function prototype

কোন প্রোগ্রামে ফাংশন ব্যবহার করা হলে, প্রোগ্রামের শুরুতেই সেই ফাংশন সম্পর্কে সংক্ষিত বর্ণনা দেয়া হয় যাতে করে compiler বুঝতে পারে যে এই নামের ও গঠনের ফাংশন পরে কোথাও Define করা হয়েছে। যেমন ঃ

```
# in clude < stdio.h>
show ( );  ←———————— function prototype
main ( ) {
        show ( ) ;
          }
show ( ) {
        printf ("How are you?");
          }
```

* ফাংশন prototype-এর গঠন নিম্নরূপ ঃ

return_type Functin_name (Parameter list....);

এখানে return_type এবং parameter list কি, তা আমরা কিছু পরে জানতে পারবো।

NOTE

উপরের প্রোগ্রামে show ( ) ফাংশনটি যদি main ( ) এর উপরে লিখতাম তাহলে Function prototype না লিখলে ও ভুল হত না।

Function calling

'call' শব্দের অর্থ 'ডাকা' বা 'আহবান করা'। C প্রোগ্রামে call প্রায় একই অর্থে ব্যবহৃত হয়। আমরা যখন কো ফাংশন difine করি তখন তার একটি নাম দেই এবং প্রত্যেকটি ফাংশন কোন নির্দিষ্ট কাজ করার জন্য defi করা হয়। প্রোগ্রামে যখন কোন নির্দিষ্ট কাজ করার প্রয়োজন হয় তখন শুধুমাত্র ফাংশন-এর নাম লিখলেই সে কাজটি করে দেয়।

ফাংশনের নাম লিখলেই, ফাংশন কোন নির্দিষ্ট কাজ করে দেয়, এটাই হল Function calling.

ফাংশন call করার গঠন নিম্নরূপ ঃ

Function_name;

উদাহরণ ঃ

```
# include <stdio.h>
show ( ) {
        printf ("joy of Love");
        }
main ( ) {
        show ( ) ; ←———————— show ( ) ফাংশন call করা হয়েছে।
        }
```

কোন ফাংশনকে একবার define করলে যতবার প্রয়োজন ততবার call করা যায়। যেমন–

```
man ( ) {
        show ( ); /* calling 1st time */
        show ( ) ; /* calling 2nd time */
        show ( ) ; /* calling 3rd time */
        /* you can call 100 times */
        }
```

কোন ফাংশনকে main ( ) এর মধ্যে বা main ( ) এর বাইরে অপর কোন ফাংশন এর মধ্যে call করা যায়। যেমন নিম্নের প্রোগ্রাম প্রথমে main ( ) এর মধ্যে show ( ) call হবে এবং show ( ) এর মধ্যে hello ( ) ফাংশন call হবে।

```
# include <stdio. h>
hello ( ) {
        printf ("Hello, Friend!");
        }
show ( ) {
        hello ( ); /* calling hello function */
        }
main ( ) {
        show ( ); /* calling show function */
          }
```

আমরা জানি main ( ) একটি ফাংশন। এই main ( ) কে অন্য কোন ফাংশন হতে call করা যায়। তবে এখানে একটি বিষয় লক্ষণীয় যে, প্রোগ্রাম execution শুরু হয় main ( ) থেকে। যেহেতু এই main ( ) অপর একটি ফাংশন এর মধ্যে অবস্থিত, তাই এই main ( ) ফাংশনটি ততক্ষণ পর্যন্ত execute হবে না যতক্ষণ পর্যন্ত না main ( ) ফাংশনটি যে ফাংশনের মধ্যে অবস্থিত সেই ফাংশনটি অপর একটি main ( ) এর মধ্যে call না হয়। যেমন–

```
# include <stdio.h>
hello ( ) {
        printf ("Say hello");
        main ( ); ← step – 3   hell ( ) এর মধ্যে main ( ) call হবে
        }
main ( ) { ← step –1   এই main ( ) প্রথমে excute হবে।
        hello ( ); ← step–2   তারপর hello ( ) ফাংশন call হবে।
        }
```

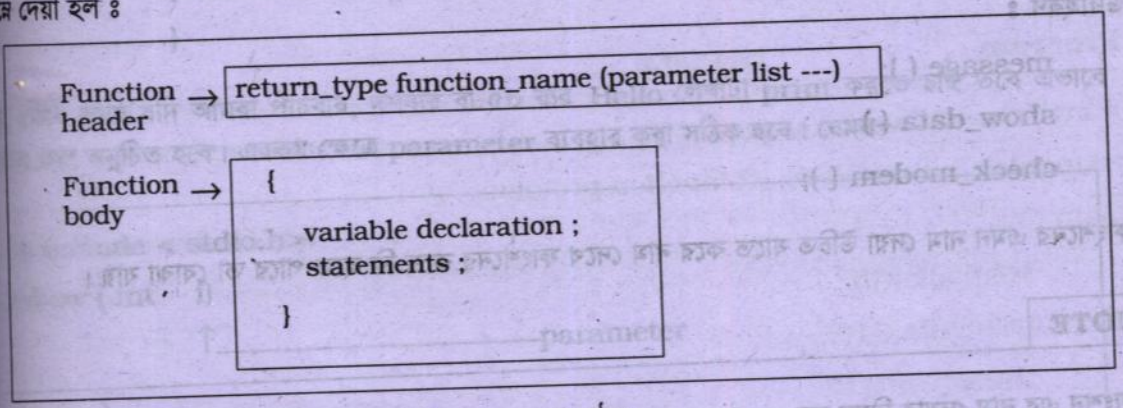
চিত্র ঃ main ( ) এর বাইরে অপর ফাংশন-এ main ( ) calling.

NOTE

কোন C প্রোগ্রামে কতগুলো ফাংশন লেখা যায়, তার কোন নির্দিষ্ট সংখ্যা নেই।

FUNCTION DEFINITION-এর গঠন

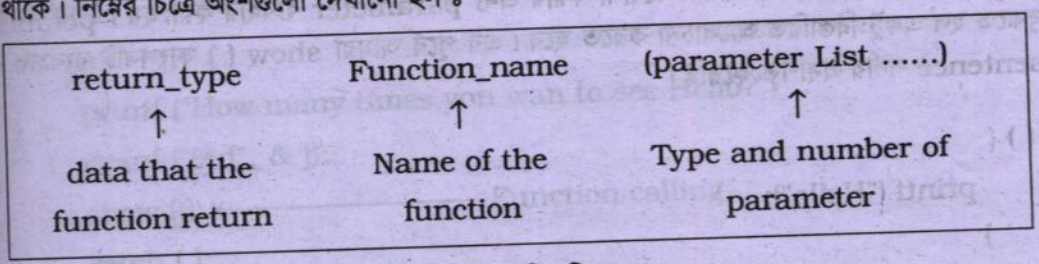
প্রতিটি ফাংশন কোন নির্দিষ্ট কাজ সম্পাদনের জন্য তৈরি করা হয়। ফাংশন কি কাজ করবে তা Function Definition-এর মধ্যে প্রোগ্রাম code আকারে লিখে দেয়া হয়। Function definition-এর নির্দিষ্ট গঠন নিম্নে দেয়া হল ঃ



চিত্র ঃ Function definition-এর গঠন।

FUNCTION HEADER

প্রতিটি ফাংশন definition-এর প্রথম লাইনটি হল Function header. Function header-এর তিনটি অংশ থাকে। নিম্নের চিত্রে অংশগুলো দেখানো হল ঃ



চিত্র ঃ Function header-এর তিনটি অংশ।

Function Name

প্রতিটি ফাংশন এর একটি নাম দিতে হয় এবং এই নামের মাধ্যমেই ফাংশনটিকে পরবর্তিতে call করা হয়।

প্রোগ্রাম কম্পাইল করলে compiler ফাংশন এর নামকে সনাক্ত করে। ফাংশনের নাম লেখার কতগুলো নিয়ম রয়েছে। যেমন ঃ

- প্রথম ক্যারেক্টার অবশ্যই অক্ষর (letter) হতে হবে।
- keyword-এর সাথে নাম মিলতে পারবে না।
- Library ফাংশন এর সাথে নাম মিলতে পারবে না। যেমন– printf ( ) নামে library ফাংশন রয়েছে। তাই আমরা কোন ফাংশন এর নাম printf ( ) দিতে পারবো না।
- ফাংশন নাম এর শেষে ' ( )' দিতে হবে, ইত্যাদি।

উদাহরণ ঃ

```
message ( );
show_data ( ) ;
check_modem ( );
```

ফাংশনের এমন নাম দেয়া উচিত যাতে করে নাম দেখে ফাংশনের কাজ কি হতে পারে তা বোঝা যায়।

NOTE

ফাংশন এর নাম দেয়ার নিয়ম হল variable-এর নাম দেয়ার নিয়মের অনুরূপ

Parameter List

C প্রোগ্রামে ফাংশনকে সর্বোচ্চ ব্যবহার উপযোগী করার জন্য parameter ব্যবহার করা হয়। perameter কি তা বুঝতে হল একটু বিস্তারিত আলোচনা করতে হবে। এর পূর্বে আমরা show ( ) ফাংশনটি ব্যবহার করে একটি sentence পর্দায় প্রদর্শিত করেছি।

```
show ( ) {
        printf ("Hello");
    }
```

ধরুন, আমরা Hello লেখাটি তিনবার লিখব। এজন্য আমরা show ( ) ফাংশনটি তিনবার লিখতে পারি কিংবা show ( ) এর মধ্যে for লুপ ব্যবহার করে ও কাজটি করতে পারি। যেমন–

```
main ( ) {
      show ( );
      show ( );
      show ( ) ;
  }
বা,   show ( ) {
          int i;
          for (i=1; i< = 3; ++i)
                printf ("Hello");
                }
```

কিন্তু, কোন সময় যদি আমরা পাঁচবার, দশবার বা ৫০ বার Hello লেখাটা print করতে চাই তবে এভাবে প্রোগ্রাম করা অনুচিত হবে। এরকম ক্ষেত্রে parameter ব্যবহার করা সঠিক হবে। যেমন–

```
1:  # include < stdio.h>
2:  show ( int    i)
                ↑_____________________parameter
3:        {
4:           int  n;
5:     for (n = 1; n <= i; ++n)
6:              printf ("\n Hello");
7:          } /*end  show ( )*/                    j এর মান i এ copy হবে
8:   main( ) {
9:        int    j;
10:       printf ("How many times you wan to see Hello?")
11:       scanf ("%d", & j);
12.       show (j); ←_____________ Function calling
13:       getch ( );
14:          } /*end main ( )*/
```

চিত্র ঃ ফাংশনে parameter ব্যবহার।

উপরের প্রোগ্রামে প্রথমে জানতে চাওয়া হবে যে মোট কতবার আমরা Hello লেখাটা দেখতে চাই। যদি 1 দেই তবে একবার, যদি 2 দেই তবে ২বার, যদি 20 দেই তবে বিশবার Hello লেখাটা দেখতে পারবো। অর্থাৎ যতবার ইচ্ছে ততবার Hello লেখাটা দেখতে পারবো।

লাইন 12-এ show ( ) ফাংশন এর মধ্যে j ব্যবহার করা হয়েছে। এতে করে j-এর মান লাইন 2-এ show ( ) এর i এর মধ্যে copy হবে। অর্থাৎ i ও j-এর মান একই হবে। লাইন 5-এ for লুপ যেহেতু i-এর মান পর্যন্ত কাজ করবে, সেহেতু আমরা show (j)-এর j তে যে মান দিব ঠিক ততবার Hello লেখাটা প্রিন্ট হবে।

এবার আরো একটি সমস্যা আলোচনা করে parameter বোঝানোর চেষ্টা করা হবে। 1+2+3+.....+n প্রোগ্রামটি এর পূর্বেও আমরা করেছি। আমরা জানি 1+2+3+....+10 = 55, একে Triangular number বলা হয়। প্রোগ্রামে আমাদের কখনো 10 পর্যন্ত, কখনো ২০, ৩০ বা ১০০ বা আরো বেশি সংখ্যার triangular number-এর প্রয়োজন হতে পারে। এজন্য ফাংশন এর মধ্যে parameter ব্যবহার করা অধিক উপযোগী হবে। যেমন ঃ

Program **triangf.c**

```
# include <stdio.h>
triangular (int number); /* fanction prototype */
main ( ) {
        triangular (5);
        triangular (10);
        triangular (15);
        getch ( );
        }

triangular (int number)
      {
      int i, triangular_number;
      triangular_number = 0;
      for (i=1; i <=number ; ++i)
      triangular_number = triangular_number +i;
      printf ("%dth triangular_number is:% d \n", number, triangular_number);
      }
```

OUTPUT

```
5 th triangular number is : 15
10 th triangular number is : 55
15 th triangular number 15 : 120
```

triangt.c প্রোগ্রাম বিশ্লেষণ

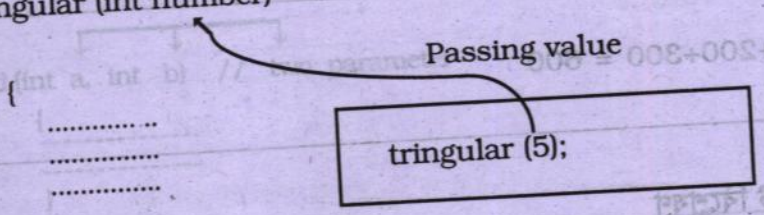
- trianguler (int number);

এখানে fanction prototype ডিকলেয়ার করা হয়েছে।

- triangular (5);
  triangular (10);
  triangular (15);

এখানে তিনবার ফাংশন call করা হয়েছে। এবং প্রত্যেকবার নতুন নতুন মান parameter হিসেবে pass করা হয়েছে। এখানে প্রথমে triangular (5) লেখার ফলে 5 কপি (copy) হয়েছে triangular (int number) এর number-এর মধ্যে।

- triangular (int number)



triangular (5)-এর 5 copy হয়ে number-এ গিয়েছে। সুতরাং number-এর মানও 5.

ফাংশনে একাধিক parameter ব্যবহার

কোন ফাংশনে প্রয়োজনবোধে একাধিক parameter ব্যবহার করা যাবে। একাধিক parameter-এর ক্ষেত্রে প্রতিটি parameter কমা (,) দ্বারা বিচ্ছিন্ন থাকে। নিম্নের mparamf.c

প্রোগ্রামটি লক্ষ করুন ঃ

Pragram **mparamf.c**

```
# include <stdio.h>
sum (int i, int j, ink k); /*prototype declaration*/
main ( ) {
        sum (100, 200, 300); /*function calling*/
        getch ( );
       }
sum (int i, int j, int k) /*function definition*/
      {
       int total;
       total = i + j + k;
       printf ("%d+%d+%d = %d", i, j, k, total); /*print result*/
       }
```

OUTPUT 100+200+300 = 600

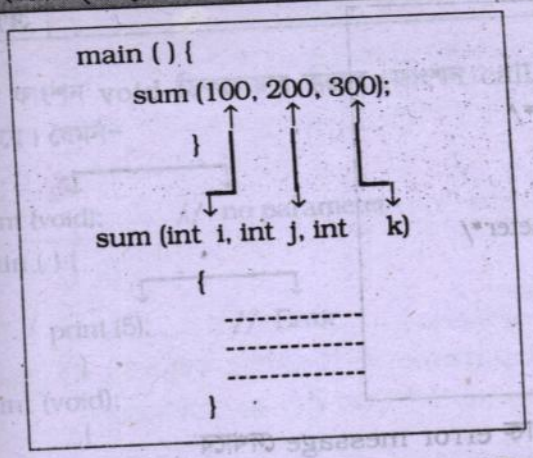
mparamf.c প্রোগ্রামটি বিশ্লেষণ

❑ sum (int i, int j, int k)

sum ফাংশন-এ তিনটি parameter দেয়া হয়েছে যারা কমা (,) দ্বারা একে অপরের থেকে বিচ্ছিন্ন।

❑ sum (100, 200, 300);

এখানে 100 যাকে parameter i-তে, 200 যাবে parameter j-তে এবং 300 যাবে parameter k-তে। নিম্নের চিত্র লক্ষ্য করুন।



❑ এখানে আমরা জদি sum (300,200,100) লিখতাম তবে i=300, j=200 এবং k=100 হত।

NOTE

ফাংশন definition-এ যে কয়টি parameter দেয়া হয়েছে, ফাংশন call করার সময় সেই কয়টি মান দিতে হবে। অন্যথায় কম্পাইলার error দেখাবে। নিচের প্রোগ্রামটি ভুল–

```
main ( ) {
    add (10, 20, 30); // three values
    }
add (int a, int b)  // two parameter
    {
    ----------
    ----------
    }
```

Parameter Type Mismatch

ফাংশন definition এ parameter যে ডাটা টাইপ-এর ডিকলেয়ার করা হয় ফাংশন call করার সময় সেই টাইপের মান দিতে হয়, অন্যথায় কম্পাইলার error দেখায়। নিম্নের প্রোগ্রামে int টাইপ এর স্থলে float টাইপ ডাটা দেয়া হয়েছে ফাংশন call করার সময়–

```
main ( ) {
        add (5.0, 1);    /*  float  type value*/
        }
add (int  a,  int  b)   /*    int  type parameter*/
        {
         int  z;
         z = a + b;
        }
```

উপরোক্ত প্রোগ্রামটি compile করলে কম্পাইলারে নিম্নোক্ত error message দেখাবে–

error : parameter type mismatch

Function without parameter

কোন ফাংশনে parameter না থাকলে কম্পাইলার তা নিজে থেকে বুঝে নেয়। কিন্তু, আমরা যদি কম্পাইলারকে বলে দিতে চাই যে ফাংশনে কোন parameter নেই তবে ফাংশন prototype ডিকলেয়ার করার সময় এবং ফাংশন definition করার সময় parameter লেখার স্থলে Void লিখতে হবে। void একটি ডাটা টাইপ। এটি সাধারণত ফাংশনের ক্ষেত্রে ব্যবহৃত হয়। void কম্পাইলারকে বোলে দেয় যে ফাংশনের কোন parameter নেই। যেমন–

Program **voidf.c**

```
# include <stdio. h>
show (void);
main ( ) {
     show ( ) ;
     getch ( ) ;
        }
show (void) {
     printf ("No parameter, so use void");
        }
```

NOTE

কোন ফাংশন void ডিকলেয়ার করলে, ফাংশন call করার সময় কোন মান দিলে কম্পাইলার error দেখাবে। যেমন–

```
print (void);        // no parameter
main ( ) {
      print (5);     // Error
    }
print (void);
      {
       ---------
       ---------
      }
```

Parameter ও Argument-এর পার্থক্য

ফাংশন header-এর মধ্যে যে ডাটা টাইপ ডিকলেয়ার করা হয় তা হল parameter. Parameter হল একধরনের রূপক (dummy) ভেরিয়েবল যা ঐ ফাংশনের বাইরে ব্যবহার করা যায় না।

ফাংশন calling-এর সময় এর মধ্যে যে মান বা ভেরিয়েবল ব্যবহার করা হয় তা হল argument। যেমন ঃ

```
main ( ) {
      add (10, 20);          // Arguments
    }
add (int i,  int  j)    //   parameter
    {
     int  j;
     z = i + j;
    }
```

উপরের প্রোগ্রামে 10,20 হল argument এবং i, j হল parameter।

NOTE

একাধিকবার কোন ফাংশন call করলে argument ভিন্ন হতে পারে, যেমন–

```
add (5,10);
add (1,2);
```

Classic পদ্ধতিতে Parameter ডিকলেয়ারেশন

C ল্যাংগুয়েজ উৎপত্তির প্রথমদিকে ভিন্নভাবে ফাংশনের parameter ডিকলেয়ার করা হত। এই পদ্ধতিতে parameter ডিকলেয়ার করাকে traditional বা classic পদ্ধতি বলা হয়। এর গঠন নিম্নরূপ–

```
return_type function_name (parameter list.....)
data_type parameter;
data_type parameter;
```

উদাহরণ ঃ

```
add (i, j)
int i;
int j;
 {
   int z;
   z = i + j;
 }
```

ANSI C standard আধুনিক ও classic উভয় পদ্ধতিতে parameter ডিকলেয়ার করাকে সমর্থন করে তবে আধুনিক পদ্ধতিতে ডিকলেয়ার করাকে অধিক support করে।

NOTE

এতক্ষণ আমরা আধুনিক পদ্ধতিতে parameter ডিকলেয়ার করেছি। যেমন– add (int i, int j)

Actual parameter ও Formal parameter-এর পার্থক্য

function definition-এ যে parameter ব্যবহৃত হয় তাই হল formal parameter.

ফাংশন call করার সময় যে argument দেয়া হয় তাই হল actual parameter.

```
main ( ) {
        ↓ ↓ ——————— Actual parameter
    add (1, 2)
    }
        ↓ ↓ ——————— Formal parameters
add (int i,  int  j)  {
        ----------
        ----------
            }
```

ফাংশন থেকে তথ্য পাঠানো (Returning value)

কখনো কখনো কোন ফাংশন call করলে সেই ফাংশন থেকে caller ফাংশনে মান বা তথ্য পাঠানের প্রয়োজন হয়। কোন ফাংশন থেকে অপর কোন ফাংশনে তথ্য পাঠানোর জন্য return শব্দটি ব্যবহৃত হয়। এটি একটি keyword। নিম্নের প্রোগ্রামে cube ফাংশন থেকে main ( )-এ Value পাঠানো (return) হয়েছে।

Program **cube.c**

```
# include <stdio.h>
# define PI 3.1415
int cube (int i);
main ( ) {
        int cube_r, radius, volume;
        printf ("Volume of sphere \n");
    printf ("Enter value for radius :" );
    scanf ("%d", & radius);
    cube_r = cube (radius);
    volume = (4*PI*cube_r)/3
```

continue

Program **cube.c**

```
printf ("\n Volume = %d", volume);
getch ( );
}
int cube (int i) {
        return (i*i*i);
                }
```

INPUT/OUTPUT

```
Volume of sphere
Enter value for radius : 2
Volume = 33
```

cube.c প্রোগ্রাম বিশ্লেষণ

- আমরা জানি বৃত্তের ব্যাসার্ধ যদি r-হয় তবে এর আয়তন হবে,

$$V = 4/3 \pi r^3$$

cube.c প্রোগ্রামে বৃত্তের আয়তন নির্ণয় করা হয়েছে।

- `cube_r = cube (radius);`

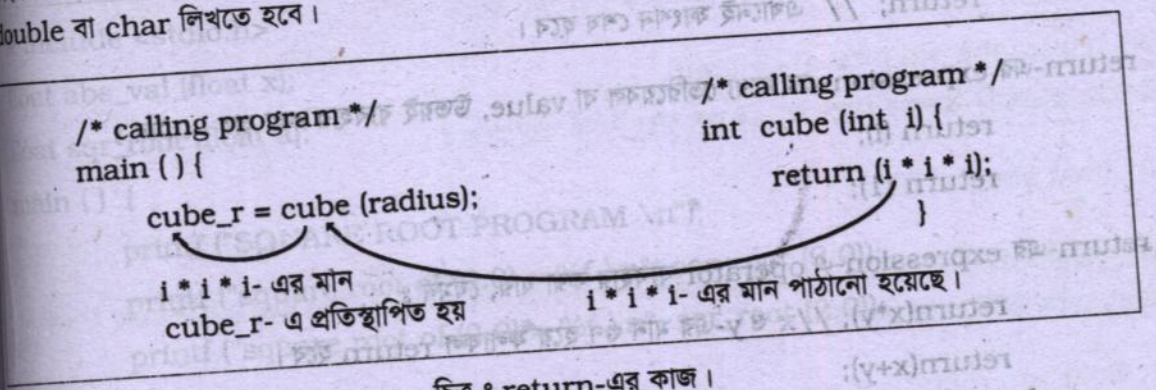
এখানে cube ( ) ফাংশনকে call করা হয়েছে। এর মধ্যে একটি paramet দেয়া হয়েছে। cube (radius)-এর মধ্যে radius এর ঘন (cube) নির্ণয় করা হবে এবং ফলাফল cube_r-এ নির্দেশিত হবে।

-
```
int   cube (int  i)  {
          return  (i * i * i);
}
```

return type

এই cube ফাংশনের মধ্যে i কে তিনবার গুণ করে i এর ঘন (cube) মান নির্ণয় করে main ( ) ফাংশনে পাঠানো হয়েছে। cube ফাংশন থেকে main ( ) এ -এর ঘনমান পাঠানোর জন্য return স্টেটমেন্ট ব্যবহার করা হয়েছে।

এখানে লক্ষণীয় যে cube ( ) ফাংশনের পূর্বে int লেখা হয়েছে। এই int হল cube ( ) ফাংশনের return ডাটা টাইপ। অর্থাৎ, কোন ফাংশন যদি int টাইপ ডাটা return করে তবে ফাংশনের পূর্বে int লিখতে হবে। ফাংশন যদি float, double বা char টাইপ ডাটা return করে তবে ফাংশনের পূর্বে যথাক্রমে float, double বা char লিখতে হবে।



চিত্র ঃ return-এর কাজ।

return keyword

return একটি keyword. প্রোগ্রামে return ব্যবহার করার গঠন। নিম্নরূপ–

```
return (expression);
```

উপরে return স্টেটমেন্টের অর্থ হল ফাংশনটি expression-এর মান calling ফাংশনে পাঠিয়ে দিবে। এখানে parentheses বা '( )' হল optional (না ব্যবহার করলে ভুল হবে না।)।

return-এর কাজ ঃ

i) কোন ফাংশনে return স্টেটমেন্ট exceute হওয়ার সাথে সাথে ঐ ফাংশনের execution বন্ধ হয়ে যায় এবং প্রোগ্রামের control প্রতিস্থাপিত হয় calling ফাংশনে। সুতরাং, return ফাংশন থেকে exit হবার জন্য ব্যবহৃত হয়।

ii) return-এর সাথে কোন expression থাকলে সেই expression-এর মান calling ফাংশনে পাঠানোর জন্য return ব্যবহৃত হয়।

iii) কোন ফাংশন -এ return এরপর কোন statement থাকলে তা execute হবে না। যেমন ঃ

```
if (i = = 0)
    return (i);
++i;
```

উপরের statement গুলিতে i যদি '0' হয় তবে ++i স্টেটমেন্ট execute হবে না এবং এই ফাংশন terminate (শেষ) হবে।

return লেখার নিয়ম ঃ

return-এর সাথে কোন expression ব্যবহার না করলে তা ফাংশনের closing brace (}) হিসেবে কাজ করে। যেমন ঃ

```
return;  // এখানেই ফাংশন শেষ হবে।
```

return-এর expression-এর মধ্যে ভেরিয়েবল বা value, উভয়ই ব্যবহার করা যায়। যেমন ঃ

```
return (i);
return (1);
```

return-এর expression-এ operator ব্যবহার করা যায়, যেমন ঃ

```
return(x*y); //x ও y-এর মান গুণ হয়ে ফলাফল return হবে।
return(x+y);
return ++x;
```

NOTE

কোন ভেরিয়েবল এর মান নির্ধারণের জন্য ফাংশন ব্যবহার করা যায়। যেমন ঃ

```
x = cube (y); // Valid
```

কিন্তু, কোন ফাংশনের মান নির্ধারণের জন্য ভেরিয়েবল ব্যবহার করা যায় না। যেমন, নিম্নের statement টি ভুল ঃ

```
cube (y) = x; // invalid
```

এবার আমরা আরেকটি প্রোগ্রামের মাধ্যমে return statement বোঝার চেষ্টা করবো। কোন সংখ্যা যদি x হয় তবে x-এর চরম (absolute) মান হবে |x|। এবং এই মান সর্বদা ধণাত্মক হবে। কোন সংখ্যার বর্গমূল নির্ণয় করতে হলে absolute মান নির্ণয়ের প্রয়োজন হয়।

Newton-এর বর্গমূল নির্ণয়ের সূত্রানুসারে $x_0 \rightarrow \sqrt{a}$ যেখানে $i \rightarrow \alpha$

এবং $x_0 = 1, \quad x_{i+1} = \frac{1}{2}\left(x_i + \frac{a}{x_i}\right)$

কোন ধনাত্মক সংখ্যার বর্গমূল নির্ণয়ের Newton-Raphson-এর পদ্ধতি নিম্নে প্রোগ্রাম আকারে দেয়া হল ঃ

Program **sqrootf.c**

```
# include <stdio.h>
float abs_val (float x);
float sqr_root (float a);
main ( ) {
        printf ("SQUARE ROOT PROGRAM \n");
        printf ("square root of (9.0) = %f \n", sqr_root (9.0));
        printf ("square root of (2.0) = %f \n", sqr_root (2.0));
        getch ( );
        }
float abs_val (float x) {
            if (x<0)
              x = —x;
          return (x);
                  }
    float sqr_root (float a) {
    float limit = 0.00001;
    float x = 1.0;
    while(abs_val(x * x-a) > = limit)
              x = (x + a/x) /2.0;
    return (x);
                  }
```

OUTPUT

```
SQUARE ROOT PROGRAM
Square root of (9.0) = 3
Square root of (2.0) = 1.4142162
```

❑ x = (x+ a/x) /2.0;

প্রথমে x = 10 ধরা হয়েছে। এটা হল আনুমানিক square root, যে সংখ্যাটির square root নির্ণয় করতে হবে তাকে x দিয়ে ভাগ করে ভাগফল x-এর সহিত যোগ করা হয়েছে। এতে প্রাপ্ত ফলাফলকে ২.০ দ্বারা ভাগ করে ভাগফল x-এ রাখা হয়েছে। এভাবে প্রক্রিয়াটি পুনঃ পুনঃ চলতে থাকবে।

❑ while (abs_val (x*x–a)> = limit)

উপরের প্রক্রিয়াটি কতক্ষণ চলবে তা এই লুপ দ্বারা নির্দিষ্ট করা হয়েছে।

❑ float abs_val (float x);
↑

এটা ফাংশন prototype declaration. এখানে return type float দেয়া হয়েছে কারণ এই ফাংশনটি float type ডাটা return করে।

❑ float sqr_root (float a);
↑

sqr_root ( ) ফাংশনটি float টাইপ ডাটা return করে তাই এর return type দেয়া হয়েছে।

যে ফাংশন কোন মান return করে না (Function returning nothing) ;

কোন ফাংশন যদি কোন Value return না করে তবে সেই ফাংশনের return টাইপ এর স্থলে void লিখতে হবে। যেমন ঃ

```
void show ( ) {
        printf ("Hello");
          }
```

এখানে আমরা যদি void না লিখি তাহলে কম্পাইলার কোন error দেখাবে না। কারণ, ফাংশনের নামের পূর্বে যদি return type উল্লেখ না করা হয় তবে compiler ধরে নেয় যে সেই ফাংশনটি int টাইপ ডাটা return করবে। এটা by default ঠিক করে দেয়া আছে কম্পাইলারের মধ্যে। main ( ) এর মধ্যে return statement ব্যবহৃত না হলে main ( ) এর পূর্বে void লিখতে হয়। যেমন ঃ

```
void main ( ) {
        .....
        .....
          }
```

NOTE

কোন ফাংশন যদি value return করে তবে তার পূর্বে void লিখলে ভুল হবে। যেমন ঃ

```
void cube ( ) {
↑           return (x*x*x);
ভুল         }
```

Nesting of function

C প্রোগ্রামে ফাংশনের argument হিসেবে ফাংশন ব্যবহার করা যায়। যেমন ঃ নিম্নের statemant টি লক্ষ্য করুন ঃ

$$x = \sqrt{(a+b)^3}$$

উপরের expression টি nesting function আকারে নিম্নোক্তভাবে লেখা যায়–

```
x = root (cube (add (a, b)));
```

এখানে, প্রথমে add (a, b) call হবে এবং এর ফলাফল cube ( ) এর argument হিসেবে ব্যবহৃত হবে। এরপর cube ( ) ফাংশনের ফলাফল root ( ) এর argument হিসেবে ব্যবহৃত হয়।

ফাংশনের প্রকার ভেদ (category of Functions)

কোন ফাংশনে parameter এবং argument আছে কি নেই কিংবা ফাংশন কোন মান return করে কি করে না-এর উপর ভিত্তি করে ফাংশনকে নিম্নোক্ত শ্রেণীতে ভাগ করা যায় ঃ

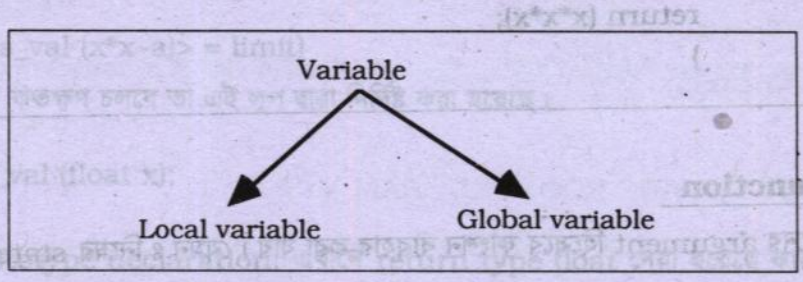
i) Functions with arguments and return values
ii) Functions without arguments and return values
iii) Functions with argumants and no return values
iv) Functions without arguments but have return values

Scope Rules of variables

কোন ফাংশনের মধ্যে যে code লেখা হয় তা অন্যান্য সমস্ত ফাংশন থেকে পৃথক থাকে। কোন ভেরিয়েবল যদি ফাংশনের অভ্যন্তরে ডিকলেয়ার করা হয় তবে সেই ভেরিয়েবল ঐ ফাংশনের বাহিরে ব্যবহার করা যায় না। আবার, কোন ভেরিয়েবল যদি main ( ) এবং অন্যান্য ফাংশনের বাহিরে প্রোগ্রামের শুরুতেই ডিকলেয়ার করা

হয়, তবে সেই ভেরিয়েবল প্রোগ্রামের যে কোন ফাংশনে ব্যবহার করা যায়।। অর্থাৎ এ ধরনের ভেরিয়েবল এর scope সীমিত থাকে না।

প্রোগ্রামে ভেরিয়েবলকে কোথায় কিভাবে ডিকলেয়ার করা হয়, তার উপর ভিত্তি করে ভেরিয়েবলকে দুইভাগে ভাগ করা যায়।



Local variable :

কোন ফাংশনের মধ্যে ভেরিয়েবল ডিকলেয়ার করলে ঐ ভেরিয়েবল ফাংশনের বাহিরে ব্যবহার করা যায় না। এই ধরনের ভেরিয়েবল-কে local ভেরিয়েবল বলে।

local ভেরিয়েবল এর scope শুধুমাত্র নিজস্ব ফাংশনে সীমাবদ্ধ থাকে। যেমন ঃ

Program **locvar.c**

```
# include <stdio. h>
void show (void);
main ( ) {
        show ( );
        printf ("\n Value of i in main is : %d", i);
        getch ( );
        }
void show (void) {
                int i;
                printf ("Value of i in show ( ) is : %d", i);
                }
```

COMPILE প্রোগ্রামটি compile করলে নিম্নোক্ত error message দেখা যাবে।

Error LOCVAR.C 5: Undefined symbol 'i'

এই error message-এর অর্থ হল LOCVAR.C প্রোগ্রামের পঞ্চম লাইনে যে i ভেরিয়েবল ব্যবহার করা হয়েছে তা Undefined (ডিকলেয়ার করা হয়নি)।

□ LOCVAR.C প্রোগ্রামের show ( ) ফাংশনে i ভেরিয়েবল ডিকলেয়ার করা হয়েছে। সুতরাং show ( ) ফাংশনের বাইরে এই ভেরিয়েবল ব্যবহার করা যাবে না।

□ main ( ) ফাংশনের printf ( ) এর মধ্যে i ভেরিয়েবল ব্যবহৃত হয়েছে। কিন্তু, এখানে i ব্যবহার করা যাবে না। কারণ i local variable.

Local Variable's Life time

ফাংশনে যখন কোন local variable ডিকলেয়ার করা হয় তখন stack memory-তে ঐ ভেরিয়েবল এর জন্য স্থান সংরক্ষিত হয়। তবে ফাংশনের কাজ শেষ হওয়া মাত্র local ভেরিয়েবল এর জন্য নির্দিষ্ট স্থান ধ্বংস হয়ে যায়। অর্থাৎ, কোন ফাংশন যতক্ষণ execute হয় ঠিক ততক্ষণ তার local variable-এর জীবন (Life) থাকে।

লোকাল ভেরিয়েবল এর কিছু বৈশিষ্ট্য ঃ

□ একই নামের ভেরিয়েবল একাধিক ফাংশনে ব্যবহার করা যায়। তবে কম্পাইলার প্রতিটি ফাংশনের ভেরিয়েবলকে পৃথকভাবে সনাক্ত করে। যেমন ঃ

Program **locvar1.c**

```
# include <stdio. h>
func1 ( ) ;
func2 ( );
main ( ) {
        func1 ( ) ;
        func2 ( );
        getch ( );
        }
```

continue

Program **locvar1.c** continue

```
func1 ( ) {
      int i = 10;
      printf ("func1 ( ) i = %d", i);
      }
func2 ( ) {
      int i=20;
      printf ("\n func2 ( ) i = %d", i);
      }
```

OUTPUT

```
func1 ( ) i = 10
func2 ( ) i = 20
```

NOTE

কোন প্রোগ্রাম ব্লাক এর মধ্যেও local variabe ডিকলেয়ার করা যায়, যেমন ঃ

```
int i=1 // value of i is 1
if (i) { int i =0; // This is different i
  printf ("i=%d", i);
    }
```

এখানে if condition block-এর মধ্যের i এবং প্রথম i দুটি ভিন্ন ভেরিয়েবল

❑ local ভেরিয়েবল ব্যবহার করে প্রোগ্রাম করলে প্রোগ্রামের data security অক্ষুণ্ন থাকে এবং ফাংশনগুলো more independent থাকে।

Global Variable

প্রোগ্রামের শুরুতেই main ( ) বা অন্য কোন ফাংশনের বাইরে ভেরিয়েবল ডিকলেয়ার করলে, তাকে global variable বলে। Global variable প্রোগ্রামের যে কোন স্থানে ব্যবহার করা যায়। যেমন ঃ নিম্নের glob-var.c প্রোগ্রাম i কে main ( ) এবং show ( ) ফাংশনে ব্যবহার করা হয়েছে।

Program **glob_var.c**

```
# include <stdio.h>
int i = 5;
void show (void) {
        printf ("show ( ) i = %d", i);
            }
void main (void) {
        printf ("main ( ) i =%d \n", i);
        ++i;
        show ( ) ;
        getch ( );
          }
```

OUTPUT

```
main ( ) i = 5
show ( ) i = 6
```

golb_var.c প্রোগ্রাম বিশ্লেষণ

- int i = 5;

এখানে i-কে global ভেরিয়েবল হিসেবে ডিকলেয়ার করা হয়েছে।

- printf ("main ( ) i = %d \n", i);
  ++i;

এখানে main ( ) এর মধ্যে printf ( ) এর সাহায্যে i-এর মান প্রদর্শিত হবে 5. ++i-এর মাধ্যমে i এর মান বেড়ে হবে 6,

- printf ("show ( ) i = %d", i);

এখানে i-এর মান 6 প্রদর্শিত হবে। কারণ i হল global variable. প্রোগ্রামের যেখানেই variable এর মান পরিবর্তন করি না কেন তার প্রতিফলন সম্পূন প্রোগ্রামে পড়বে।

LOCAL Vs GLOBAL Variable

কোন প্রোগ্রাম-এ একই নামের global এবং Local variable ডিকলেয়ার করা যাবে। এক্ষেত্রে ফাংশনের মধ্যে ঐ ভেরিয়েবলটি ব্যবহার করতে চাইলে local variable টিই ব্যবহৃত হবে। তবে main ( ) এর মধ্যে global variable টি ব্যবহৃত হবে। যেমন ঃ

Progsam **loc vs glb.c**

```
# include <stdio. h>
int i = 1; /* global variable */
void show (void) {
        int i=2; /* local variable */
        printf ("\n show ( ) i = %d", i);
            }
void main (void) {
        printf ("main ( ) i = %d", i);
        show ( );
        getch ( );
            }
```

OUTPUT

```
main ( ) i = 1
show ( ) i = 2
```

locvsglb.c প্রোগ্রাম বিশ্লেষণ

❑ show ( ) এর printf ( ) এর মধ্যের i টি হল local variable. প্রোগ্রামে local এবং global variable এক হলে ফাংশনের মধ্যে সর্বদা local variable ব্যবহৃত হয়।

Call by Value

প্রোগ্রামে ফাংশনকে দুইভাবে call করা যায়। এর মধ্যে একটি হচ্ছে call by value। এটা c প্রোগ্রামের default পদ্ধতি। এ পদ্ধতিতে কোন ফাংশনকে call করা হলে ফাংশনের actual parameter-এর

formal parameter-এ copy হয়। এক্ষেত্রে formal parameter-এর মানকে পরিবর্তন করলে actual parameter-এর মান পরিবর্তন হয় না।

callbyv.c প্রোগ্রামটি লক্ষ্য করুন ঃ

Program **callby v.c**

```
# include <stdio.h>
void swap_value (int i, int j);
void main ( void) {
        int p = 5, q = 10;
        printf ("In main ( ) p = %d, q=%d", p,q);
        swap_value (p,q);
        printf ("\nIn main ( ) p=%d, q =%", p,q);
        getch ( ) ;
          }
void swap_value (int i, int j) {
          int temp;
          temp = i;
          i = j;
          j = temp;
printf ("\n In swap_value i=%d, j=%d", i,j);
          }
```

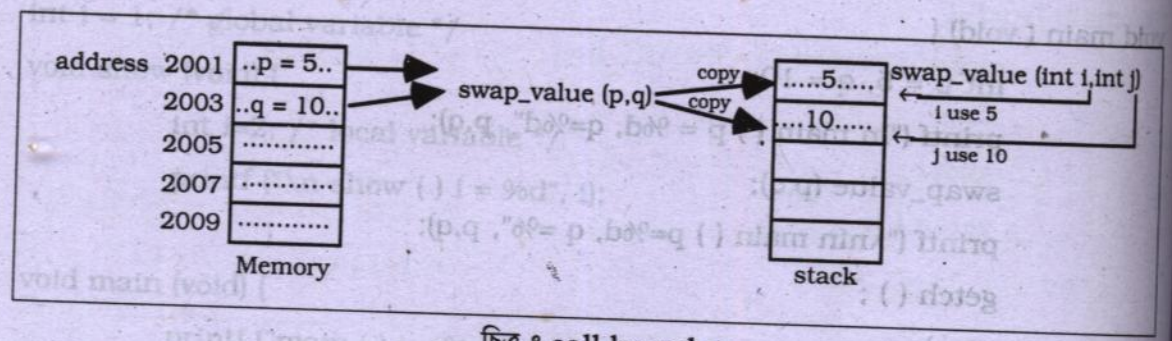
OUTPUT

```
In main p = 5, q = 10
In swap_value p=10, q=5
In main p = 5, q = 10
```

Callbyv.c প্রোগ্রাম বিশ্লেষণ

- swap_value ( ) ফাংশনের মধ্যে i-এর মান j-তে এবং j-এর মান i-তে নেয়া হয়েছে। কিন্তু, p ও q এর মান অপরিবর্তিত রয়েছে।
- swap_value (p, q);

এই statement-এর মাধ্যমে p ও q-এর মান memory 'র stack অংশে সংরক্ষিত হয় এবং swap-value ফাংশন এই stack-এ সংরক্ষিত মান নিয়ে কাজ করে। যার ফলে p ও q-এর মান memory-তে অপরিবর্তিত থাকে। নিম্নের চিত্রটি লক্ষ্য করা যাক–

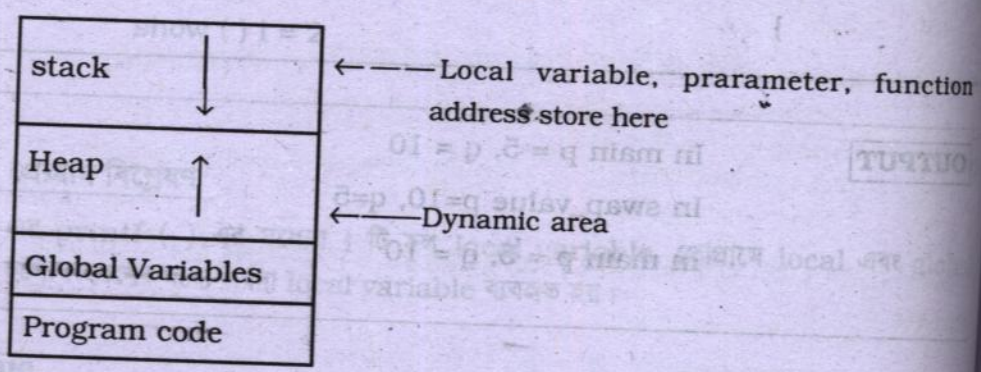


চিত্র ঃ call by value

NOTE

STACK : stack হল মেমোরীর এমন একটি অংশ যেখানে প্রোগ্রাম চলাকালিন সময়ে ফাংশনের argumant, local ভেরিয়েবল ও function-এর address সংরক্ষিত হয়।

একটি সম্পূর্ণ প্রোগ্রামের বিভিন্ন অংশ প্রোগ্রাম চলাকালিন সময়ে মেমোরী কোথায় কোথায় সংরক্ষিত হয় তা নিচে দেখানো হল ঃ



চিত্র ঃ memory-তে প্রোগ্রামের অবস্থান।

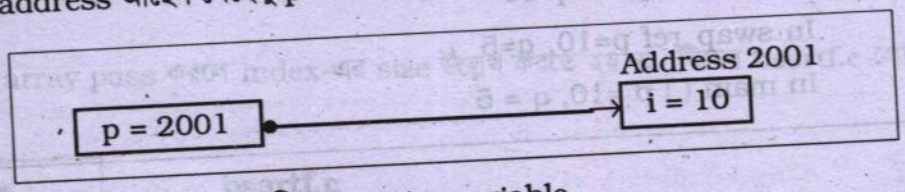
Call by Reference

এ পদ্ধতিতে কোন ফাংশনকে call করলে actual parameter-এর address formal paramcter-এ copy হয়। ফাংশন এ ক্ষেত্রে actual parameter-এর adefuess-এ যে মান থাকে তার উপর কাজ করে।

call by reference বুঝতে হলে আগে আমাদের বুঝতে হবে pointer variable কি। pointer variable হল এমন এক ধরনের ভেরিয়েবল যা অন্য কোন variable-এর address রাখতে পারে। যেমন ঃ

```
int i = 10;
int *p;
p = & i;
```

এখানে p হল int type pointer ভেরিয়েবল যে অপর কোন int type ভেরিয়েবল এর address রাখতে পারবে। p = & i-এ p, i-এর address রেখেছে। i-এর address-এ 10 সংরক্ষিত আছে। আবার p এর মধ্যে i এর address আছে। সেহেতু p-এর মাধ্যমে 10 access করা যাবে।



চিত্র ঃ pointer variable

নিম্নের callbyr.c প্রোগ্রামটি লক্ষ্য করা যাক ঃ

Program **callb yr.c**

```
# include <stdio.h>
void swap_ref (int * i, int * j);
void main (void) {
        int p =5, q = 10;
    pintf ("\n main ( ) p=%d, q=%d", p,q);
    swap_ref (&p, &q);
    printf ("\n In main ( ) p=%d, q=%d", p,q);
    getch ( );
    }
```

continue

Program **callb yr.c** continue

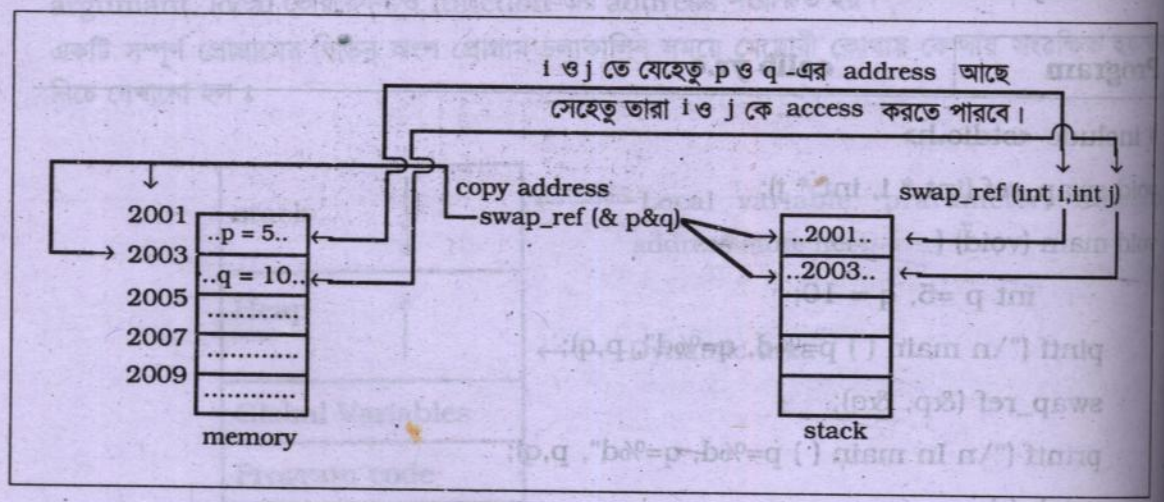
```
void swap_ref (int * i, int * j) {
        int temp;
        temp = * i;
        * i = *j;
        * j = temp;

printf ("\n In swap_ref i = %d, j=%d", *i, *j);
        }
```

OUTPUT

```
In main ( ) p=5, q=10
In swap_ref p=10, q=5
In main ( ) p =10, q = 5
```

উপরের callbyr.c প্রোগ্রামে *i ও *j যেহেতু p ও q-এর address-এ রক্ষিত value নিয়ে কাজ করেছে তাই p ও q -এর মান interchange হয়েছে। নিম্নের চিত্রটি লক্ষ্য করুন ঃ



চিত্র ঃ call by reference.

Returning More than One value

এতক্ষণ আমরা বিভিন্ন ফাংশনে value return করেছি। যেমন ঃ

```
return (p);
return (a+b);
```

এসব ক্ষেত্রেই একটি value return হয়েছে। কিন্তু, আমরা যদি একই সাথে একাধিক value return করতে চাই তবে তা array বা pointer-এর মাধ্যমে করতে হবে। এই অধ্যায়ে ফাংশনে array ব্যবহার নিয়ে আলোচনা করা হয়েছে।

Array as Function parameter

কোন ফাংশনের parameter হিসেবে array ব্যবহার করা যায়। parameter হিসেবে array-কে মোট তিনভাবে pass করা যায়। তিনটি পদ্ধতি নিম্নে আলোচনা করা হল ঃ

পদ্ধতি ঃ 1

এ পদ্ধতিতে array pass করলে index-এর size উল্লেখ করতে হয় না। নিম্নের bsortf.c প্রোগ্রামটি লক্ষ্য করা যায় ঃ

Program **bsartf.c**

```
# include <stdio.h>
void bubble_sort (int n, int num [ ] );
void main (void) {
        int i;
    static int score [5] = {20, 50, 10, 80, 5};
    printf ("scores Before sort \n");
    for (i=0; i<5, ++i)
        printf ("%d", score [i]);
    printf ("\n");
    bubble_sort (5,score);
    printf ("scores After sort \n");
    for (i = 0; i<5; ++i)
    printf ("%d,", score [i]);
    getch ( ); }
```

continue

Program **bsartf.c** continue

```
void bubble_sort (int n, int num [ ] ) {
        int i, j, temp;
    for (i=1; i<=n-1; ++i)
      for (j=1; j<=n-i; ++j)
        if (num [j-1]> = num [j])
          {
            temp = num [j-1];
            num [j-1] = num [j];
            mun [j] = temp;
          }
    }
```

OUTPUT

```
Score Before sort
20, 50, 10, 80, 5
Marks After sort
5, 10, 20, 50, 80.
```

bsortf. c প্রোগ্রাম বিশ্লেষণ

- bubble_sort (5,score);

এখানে ফাংশনের parameter হিসেবে array ব্যবহার করা হয়েছে। score-এর address coopy হয়ে ফাংশন-এ pass হবে।

- void bubble_sort (int n, int num [ ])

```
{
.....
.....
}
```

এখানে num [ ] হল size বিহীন array। যেহেতু, c তে boundary checking করা যায় না সেহেতু এখানে array-এর প্রকৃত size অপ্রাসঙ্গিক।

পদ্ধতি ঃ 2

এ পদ্ধতিতে ফাংশন def_inition-এ array-এর size বলে দেয়া হয়। যেমন ঃ

Program **arsizerf.c**

```
#include <stdio.h>
void show (int num [5]);
main ( ) {
    int ar [5], i;
    for (i=0; i<10; ++i)
      ar [i] = i;
    show (ar);
    getch ( );
     }
void show (int num [5]) {
    int j;
    for (j=0; j <10; ++j)
    printf ("%d", num [j]);
     }
```

arsizef.c প্রোগ্রাম বিশ্লেষণ

- void show (int num [5])
  {
  .....
  .....
  }

এখানে num হল int array যার মধ্যে 5 টি elements রয়েছে। C এই num-কে int টাইপ pointer এ রূপান্তরিত করে।

এতে করে num অ্যারে ar-এর address সংরক্ষণ করতে পারে। মনে রাখতে হবে যে একটি সম্পূর্ণ array কখনো pass করা সম্ভব নয়।

পদ্ধতি ঃ 3

এই পদ্ধতিতে ফাংশন definition-এ parameter হিসেবে pointer variable ডিকলেয়ার করা হয়। যেমন–

```
void show (int * num)
        {
        ......
        ......
        }
```

এখানে num হল int টাইপ pointer variable যে array-এর address সংরক্ষণ করতে সক্ষম।

Recursive Functions

কোন ফাংশন অপর কোন ফাংশনকে call করতে পারে। কিন্তু, কোন ফাংশন যখন নিজেকে call করে তখন তাকে recursive function বলে।

recursive function একটি নির্দিষ্ট অবস্থা পর্যন্ত নিজেকে নিজে call করতে থাকে এবং এর ফলে একটি শিকল (chain of process)-এর সৃষ্টি হয়।

Recursive ফাংশন এর একটি অন্যতম উদাহরণ হল factorial ফাংশন। আমরা জানি–

factorial n = n (n–1) (n–2).....1.

অর্থাৎ !5 = 5*4*3*2*1

এই factorial-এর recursive ফাংশন -এর প্রোগ্রাম নিম্নে দেখানো হল ঃ

Program **facfunc.c**

```
# include <stdio.h>
long int factorial (int n);
void main (void) {
        int i;
    for (i=0; i<=5; ++i)
        printf ("%2d!=%ld \n", i, factorial (i));
        }
```

continue

continue

Program **facfunc.c**

```
long int factorial (int n) {
   long int total;
    if (n==0)
       total = 1;
  else
     total = n* factorial (n–1);
  return (total);
    }
```

OUTPUT

```
0! =1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

ফাংশনের argument হিসেবে ফাংশন ব্যবহার ঃ

C প্রোগ্রামে ফাংশনের argument-এ ফাংশন ব্যবহার করা যায়। আমরা জানি বৃত্তের ক্ষেত্রফল হল ঃ

πr^2 [$\pi = 3.1415$, r = ব্যাসার্ধ]

নিম্নের প্রোগ্রামে area ( ) ফাংশনের argument হিসেবে square ( ) ফাংশন ব্যবহার করা হয়েছে ঃ

Program **area1.c**

```
# include <stdio h>
int square (int i)
       {
          return i*i;
       }
```

continue

Program **area1.c** continue

```
float area (int sqr, float p)
        {
       return sqr*p;
        }
main ( ) {
        float sqr_area, pi = 3.1415;
        sqr_area = area (square (10), pi);
        printf ("Area of square is : %f", sqr_area);
        getch ( );
        return (0);
        }
```

OUTPUT area of square is : 314.149994

area 1.c প্রোগ্রাম বিশ্লেষণ ঃ

❑ sqr_area = area (square (10), pi);

এখানে প্রথমে square ( ) ফাংশন execute হয় এবং 10×10 = 100 return হয়। পরে area ( ) ফাংশনটি area (100, pi) argument নিয়ে call হয় এবং মান sqr-area-তে নির্দিষ্ট হয়।

A Fumction with variable number of arguments :

কোন ফাংশনে যতগুলো parameter define করে দেয়া হয় ঐ ফাংশন call করার সময় ঠিক ততগুলো argument ব্যবহার করা যায়। কিছু কিছু Library function আছে যেগুলোতে ইচ্ছেমত argument ব্যবহার করা যায়। যেমন ঃ

```
printf ("%d+%d = %d", i,j,k); // Three arguments
printf ("Value of i is : %d", i); // One argument
```

এখানে printf ( ) ফাংশনে প্রথমে তিনটি ও পরেরবার একটি argument ব্যবহার করা হয়েছে। অর্থাৎ একই ফাংশন বিভিন্ন সময় বিভিন্ন সংখ্যক argument নিয়ে কাজ করতে পারে।

C প্রোগ্রামের "stdarg.h" হেডার ফাইলে রক্ষিত কিছু tools ব্যবহার করে আমরা এই ধরনের ফাংশন তৈরি করতে পারি। এই ধরনের ফাংশন লেখার নিয়ম নিম্নরূপ ঃ

```
return_type func_name (mandatory parameter, .....)
        {                    ↓                    ↓
                          বাধ্যতামূলক          ঐচ্ছিক parameter
        va_list     ptr_to_arg;
va_start (ptr_to_arg, mamdatory paremeter);
va_arg (ptr_to_arg, mamdatory parameter type);
va_end (ptr_to_arg);
        }
```

এই ধরনের ফাংশন তৈরিতে কিছু macro (বা tools) ব্যবহৃত হয় যার কাজ নিচে বর্ণনা করা হল ঃ

macrs/data type	কাজ
va_list	ইহা একটি pointor data type
va_start ( )	একটি macro যা argument list-এর initialization-এর জন্য ব্যবহৃত হয়।
va_arg ( )	ইহা একটি macro যা va_list টাইপ ডাটাগুলোর address একটির পর একটি গ্রহণ করতে থাকে।
va_end ( )	সবগুলো argument নিয়ে কাজ করার পরে মেমোরীর stack অংশ restore (পুনরুদ্ধার) করার কাজে va_end ( ) ব্যবহৃত হয়। va_end ( ) ব্যবহার না করলে program crash (ধ্বংস) হতে পারে।

এ ধরনের ফাংশন ডিকলেয়ার করার সময় mandatory arguments গুলো প্রথমে লিখতে হবে এবং পরে ellipsis (...) লিখতে হবে। তবে একটি কথা মনে রাখতে হবে যে এ ধরনের ফাংশনে অন্ততপক্ষে একটি momdatory (বাধ্যতামূলক) argument থাকতে হবে। নিম্নে এর একটি উদাহরণ দেয়া হল ঃ

Program **stdargf.c**

```
# include <stdarg.h>
main ( ) {
    show (1, 2, 10, 20);
    show (2, 3, 'a', 'b', 'c');
```

continue

Program **stdargf.c** continue

```
    show (3, 4, 1.0, 2.0, 3.0, 4.0);
    getch ( );
    }
show (int type, int num) {
    int i, j;
    char ch;
    float fl;
    va_list argptr;
    va_start (argptr, num);
    switch (type)
      {
    case 1 : for (j=1; j<=num; ++ j)
      {
        i = va_arg (argptr, int);
        printf ("%d", i);
    }
    break;
case 2 :
    for (j=1; j<= num; ++j)
    {
      ch=va_arg (argptr, char);
      printf ("%c", ch);
    }
    break;
```

Program **stdargf.c**

```
case 3 :
    for (j=1; j<=num; ++j)
        {
            fl = (float) va_arg (argptr, double);
            printf ("%f", fl);
        }
}
}
```

stdargf.c প্রোগ্রাম বিশ্লেষণ ঃ

❑ show (1, 2, 10, 20)

এখানে, 1 হল data type অর্থাৎ ফাংশন parameter হিসেবে কোন্ টাইপের ডাটা পাঠানো হবে তার নির্দেশনা হল 1.2 হল কতটি argument পাঠানো হবে তার সংখ্যা। এখানে দুটি argument (10, 20) পাঠানো হয়েছে। তাই 2 লিখতে হয়েছে। নিম্নের প্রোগ্রামটি সর্ববৃহৎ সংখ্যা নির্ণয় করবে ঃ

Program **stdargmx.c**

```
# include <stdarg.h>
main ( ) {
    int maximum;
    maximum = max (4, 15, 10, 20, 5);
    printf ("\n Maximum number is : %d", maximum);
    maximum = max (5, 3, 9, 6, 12, 100);
    printf ("\n Maximum number is : %d", maximum);
    getch ( );
    }
max (int num) {
    int store, i, max;
```

Program **stdargmx.c** continue

```
    va_list argptr;
    va_start (argptr, num);
    max = va_arg (argptr, int);
for (i=1; i<num; ++i)
    {
      store = va_arg (argptr, int);
      if (store>max)
      max = store;
    }
  return (max);
  }
```

main ( ) ফাংশন

কোন প্রোগ্রামে একটি মাত্র ফাংশন থাকলে তা হবে main ( ) ফাংশন এবং প্রোগ্রামের execution এই main ( ) ফাংশন থেকে শুরু হয়। প্রোগ্রাম Run করলে program code-এর মাধ্যমে কম্পিউটারকে বিভিন্ন instruction (নির্দেশ) দেয়া হয় এবং কম্পিউটার সেই অনুযায়ী কাজ করে।

কম্পিউটারের সমস্ত নিয়ন্ত্রণ operating system-এর মাধ্যমে পরিচালিত হয়। তাই কোন c প্রোগ্রাম Run করে কম্পিউটারকে কোন কাজের নির্দেশ দিতে হলে তা অবশ্যই operating system-এর মাধ্যমে করতে হবে। অর্থাৎ "C programs run under operationg system."

main ( ) ফাংশন-এর মধ্যে return ব্যবহার করলে operating system প্রোগ্রাম execution terminate (শেষ) করে। DOS, OS/2 সহ অন্য অনেক operating system প্রোগ্রামের main ( ) ফাংশনে return (0) ব্যবহার করা হলে ঐ প্রোগ্রামকে nomally trminate করে। return এর মধ্যে '0' ব্যতীত অন্য কোন সংখ্যা লিখলে প্রোগ্রাম execution-এ কোন error (ভুল) হয়েছে বলে ধরে নেয়া হয়।

```
int       main ( ) {
↑         ..........
return    ..........
type      ..........
          return (0); ←return value
              }
```

main ( ) ফাংশনে কোন value return না করলে return type অবশ্যই void লিখতে হবে। যেমন ঃ

```
void main ( ) {
        .....
        .....
        .....
        }
```

main ( ) ফাংশনে কোন value return না করলে এবং main ( ) এর পূর্বে return type না উল্লেখ করলে Turbo C একটি অজ্ঞাত মান return করে এবং main ( ) এর return type হিসেবে int ধরে নেয়।

```
int      main ( ) {
↑          ..........
Default    ..........
value      ..........
           return (unknown value);
             }
```

এই অধ্যায়ে বিভিন্ন ফাংশনে parameter বা ও argument নিয়ে আলোচনা করা হয়েছে। main ( ) ফাংশনেও argument ব্যবহার করা যায়। এখন এই বিষয়ে আলোচনা করা হবে–

Command-Line Arguments

DOS operating system-এ কাজ করতে হলে বিভিন্ন ধরনের command-এর মাধ্যমে তা করতে হয়। যেমনঃ কোন ফাইলকে copy করতে হলে নিম্নোক্ত commond লিখতে হয়-

```
c:\> copy source_file    destination_file
```

এখানে copy হল execute ফাইল যার কাজ হচ্ছে source ফাইলকে destination ফাইলে copy করা। এখানে source file এবং destination file হল copy প্রোগ্রামে pass করা argument. কোন প্রোগ্রামে এ ধরনের argument pass করতে হলে তা অবশ্যই main ( ) এর মাধ্যমে করতে হবে কারণ প্রোগ্রাম execution শুরু হয় main ( ) থেকে।

C প্রোগ্রামে main ( ) ফাংশন এর মধ্যে argument ব্যবহার করা যায়। মনে রাখতে হবে main ( ) ফাংশনে তিনটির অধিক argument ব্যবহার করা যায় না। main ( ) ফাংশনে argamet ব্যবহার করার নিয়ম নিম্নরূপ হবে ঃ

```
main ( int argc, char * argv [ ], char * cnv [ ] )
    {
    ...
    ...
    }
```

main ( ) এর সহিত ব্যবহৃত argument গুলোর কাজ কি তা নিম্নে বর্ণনা করা হল ঃ

main ( ) এর argument	কাজ
argc	arge-কে Argument count বলা হয়। command line-এ ব্যবহৃত argument-এর সংখ্যা নির্ণয় করা এর কাজ। যেমন ঃ c:\> show Hello এখানে argc = 2 (show এবং Hello) অর্থাৎ argc = প্রোগ্রামের নাম + argument এর সংখ্যা
* argv [ ]	argv হল argument vector command line-এ যে command লিখবো তা এই argv [ ] এর মধ্যে থাকবে। * argv [ ] হল pointer to an array of string। c:\> show Hello Maria এখানে, argv [0] এর মধ্যে show অবস্থান করবে। argv [1] এবং argv [2]-এ যথাক্রমে Hello ও Maria অবস্থান করবে।
* env [ ]	env কে environment বলা হয়। এটা অবশ্যই argc ও argv-এর পরে ব্যবহার করতে হয়। env [ ] অপারেটিং সিস্টেমের বিভিন্ন environmental তথ্য দেখায়।

main ( ) ফাংশনে argument হিসেবে argc, argv ও env নাম ব্যাবহার না করে অণ্য নামও ব্যবহার করা যায়। এতে compiler error দেখাবে না। কারণ কম্পাইলার কোন্ argument-এর পর কোন্টা ব্যবহার হয়েছে তার উপর ভিত্তি করে কাজ করে, argument-এর নাম এখানে গুরুত্বহীন। কিন্তু, যেহেতু এই নামগুলো চালু হয়ে গিয়েছে, তাই argument-এ এই নামগুলো ব্যবহার করা উত্তম।

নিম্নের প্রোগ্রাম command line-এ লেখা বাক্য print করবে।

Program **cmdline1.c**

```
# include <stdio.h>
main ( int argc, char * argv [ ])
    {
      if (argc ! = 2)
        {
          printf (" Enter your name \n");
          return (1);
        }
      printf ("%s", argv [1]);
      getch ( );
      return (0);
      }
```

INPUT/OUTPUT

```
c:\> cmdline1 TAREQ <enter প্রেস করুন>
TAREQ
```

cmdline1.c প্রোগ্রাম বিশ্লেষণ

❑ main (int arge....);

command line-এ মোট কতটি word লেখা হয়েছে।

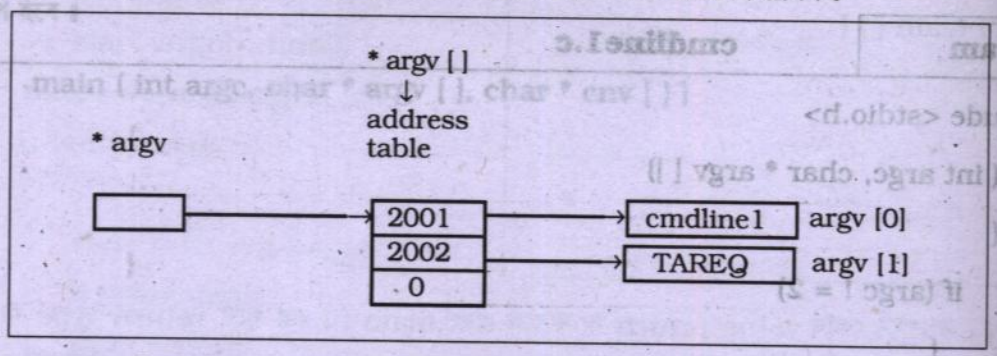
argc তা গণনা করবে। যেমন ঃ

c:\> cmdline.1 (1) TAREQ (2)

এখানে argc = 2

❑ main (....., char * argv [ ]);

argv [ ] এর মধ্যে command line-এ লেখা word গুলো রক্ষিত থাকবে। যেমন ঃ



চিত্র ঃ argv এক বা একাধিক string-এর address সংরক্ষণ করে।

❑ if (argc! = 2)

```
{
.....
.....
}
```

command line-এ একটি word লিখলে তা হবে প্রোগ্রামের নাম। যেহেতু এখানে arg [1] প্রিন্ট করা হয়েছে, তাই কিছু লিখতেই হবে। অন্যথায় প্রোগ্রাম কোন কিছু দেখাবে না।

NOTE

c:\> cmdline.1 TAREQ

এখানে cmdline.1 হল প্রোগ্রামের exeutable ফাইল এর নাম। প্রোগ্রাম Run করলে তার একটি execute ফাইল তৈরি হয়। কিন্তু, কোথায় এই ফাইল তৈরি হয় তা দেখতে হলে TC3 এর মেনু থেকে option-এর directories এ যেতে হবে। এখানে output যদি D:\ থাকে তবে প্রোগ্রামের executable ফাইল D drive-এ তৈরি হবে। তখন নিম্নের conmand লিখতে হবে

c:\> cmdline.1 TAREQ

বা

c:\> D:\cmdline.1 TAREQ

এবার main ( ) ফাংশনে env-এর ব্যবহার দেখা যাক। নিম্নের প্রোগ্রামটি কম্পিউটারের অপরেটিং সিসটেমের বিভিন্ন environmental তথ্য প্রদর্শন করবে। environmental তথ্য বলতে বিভিন্ন .bat বা .sys ফাইলকে বুঝানো হচ্ছে।

Program **env.c**

```
# include <string.h>
main (int argc, char * argv [ ], char * env [ ])
    {
        int i;
        for (i=0; env [i]; ++i) {
            if (strstr (env [i], argv [1]))
                printf ("%s\n", env [i]);
                                }
    getch ( ) ;
    return (0);
    }
```

INPUT/OUTPUT

```
c:\> env TEMP <enter প্রেস>
Tmp = c : \WINDOWS\TEMP
TEMP = c:\ TEMP
(Re-run)  c:\> env PATH
PATH = c:\ PROGRA~1 \APPLET~1 \JDK\BIN;
PATH = c:\ WINDOWS;
CLASSPATH = c:\java\LIB\CLASSES. ZIP;
```

env.c প্রোগ্রাম বিশ্লেষণ

❑ এখানে command line এ TEMP লিখলে TEMP সংক্রান্ত তথ্য প্রদর্শিত হবে। strstr ( ) ফাংশন এর জন্য string.h হেডার ফাইল include করতে হবে। strstr (str1, str2) ফাংশন str2 এর মধ্যে দেয়া string-এর জন্য str1-কে search করে।

❑ এই প্রোগ্রামে argc ও argv -কে ব্যবহার করা হয়নি কিন্তু, main ( ) এ এগুলো লিখতে হয়েছে। কারণ argv কে না লিখে env লেখা যায় না।

NOTE

string হল কতগুলো ক্যারেক্টারের সমষ্টি।

character→ 'A', 'B', 'C'

string→ "ABC"

Example :

নিম্নের প্রোগ্রামটি 10 টি সংখ্যার সবচেয়ে ছোট সংখ্যাটি খুজে বের করে দেখাবে ঃ

Program **minimum.c**

```
# include <stdio. h>
int min_val (int val [10]);
main ( ) {
        int num [10], i;
        printf ("Enter 10 Numbers\n");
        for (i=0; i<10; ++i)
            scanf ("%d", & num [i]);
        printf ("\n Lowest number is : %d", min_val (num));
        }
int min_val (int val [10])
        {
            int min_num, i;
        min_num = val [0];
        for (i=1; i<10; ++i)
            if (val [i] <min_num)
                min_num = val [i];
        return (min_num);
        }
```

Example :

নিম্নের প্রোগ্রামটি কোন্ সংখ্যা কতকবার input করা হয়েছে তা গুনে দেখাবে।

Program **countnum.c**

```
# include <stdio.h>
main ( ) {
    int rat_count [11], i, val;
    for (i =1; i<=10; ++i)
        rat_count [i] = 0;
    pintf ("Enter 20 numbers \n");
    for (i=1; i<=20; ++i)
        {
        scanf ("%d", & val);
        if (val <1 || val> 10)
            printf ("Illigal Number : %d\n", val);
        else
        ++ rat_count [val];
        }
    printf ("\n NUMBERS     INPUT THIS NUMBER HOW MANY TIMES \n");
    printf  ("..........\n");
    for (i=1; i<=10; ++i)
        printf ("%5d%15d \n", i, rat_count [i]);
}
```

Q & A :

Q.1 এমন একটি প্রোগ্রাম লিখুন যা x এবং n এর মান input নেয় এবং নিম্নের সূত্রটির মান নির্ণয় করে

$$y = x^n$$

উত্তর ঃ

```
main ( ) {
    float x, n;
    float power (float, float);
    printf ("Enter Base and power");
    scanf ("%f%f", & x, & n);
    printf ("\n% f to power %f is % f \n", x, n, power (x,n));
    getch ( );
      }
    float power (float x, float n) {
    float total;
    total = 1.0;
    if (n>=0)
        while (n--) /*positive power calculation*/
            total * = x;
    else
        while (n++) /*negative power calculation*/
            total /=x;
    return (total);
    }
```

Q.2. একটি ফাংশন মোট কতটি value return করতে পারে?

উত্তর ঃ ফাংশন সাধারণত একটি মান return করে। তবে pointer-এর মাধ্যমে একাধিক মান return করা যায়।

Q.3. ফাংশন যদি কোন value return না করে তবে সেই ফাংশনের return type কি হবে?

উত্তর ঃ void

Q.4. main ( ) ফাংশন প্রোগ্রামের কোথায় লিখতে হয়?

উত্তর ঃ প্রোগ্রামের যে কোন স্থানেই main ( ) লেখা যায়। তবে প্রোগ্রামের শুরুতেই main ( ) লেখা উচিত। তবে প্রোগ্রামের যে কোন স্থানেই main ( ) লেখা হোক না কেন, এই main ( ) ফাংশন সর্ব প্রথম execute হয়।

Exercise

1. ফাংশন কি এবং কেন প্রোগ্রাম ফাংশন আকারে লেখা হয়?
2. Recursive ফাংশন কি?
3. নিম্নের algebric সূত্রটি Recursive ফাংশন আকারে লিখুন।

$$b = 1-a+a^2/2 - x^3/6 + a^4/24 +.....+ (-1)^n a^n/n!$$

4. নিম্নের ফাংশনে ভুল কি?

```
int square (int x) {
        return ((float) x*x);
                }
```

5. নিম্নের ফাংশনে ভুল কোথায়?

```
void add (int a, int b);
    {
    return (a+b);
    }
```

6. আমরা জানি $R = \frac{V}{I}$ যেখানে, R হল resistance, V হল potential difference এবং I হল current. সূত্রটি ফাংশন আকারে লিখুন।
7. Local এবং global ভেরিয়েবল এর পার্থক্য কি?
8. Actual ও Formal parameter কি?
9. parameter ও Argument-এর মধ্যে পার্থক্য কি?
10. Nested ফাংশন কি?
11. ফাংশনের Parameter হিসেবে array কে কতভাবে pass করা যায়?
12. pointer কি?
13. call by value ও Call by reference কি?
14. এমন একটি ফাংশন লিখুন যা কোন ইংরেজি সংখ্যাকে Roman সংখ্যায় পরিণত করবে।

English Number	Roman Number
1	i
5	V
10	X
50	L

15. Return statement-এর কাজ কি?
16. Stack কি?

7

MULTIFILE PROGRAM এবং STORAGE CLASS

- একাধিক ফাইল নিয়ে একটি প্রোগ্রাম করা।
- হেডার ফাইল তৈরি।
- **Storage class**
- **auto, static, extern** এবং **register** নিয়ে আলোচনা

একাধিক ফাইলে C প্রোগ্রাম করা

C প্রোগ্রামে বিভিন্ন রকম buit-in ফাংশন ব্যবহার করা হয়। যেমন ঃ printf ( ), scanf ( ), getch ( ) ইত্যাদি। এই সকল ফাংশন বিভিন্ন ফাইলে define করা থাকে। এই ফাইলগুলো C প্রোগ্রামে সংযোগ করার জন্য # include preprosessor ব্যবহার করা হয়। যেমন ঃ

Program **multif 1.1**

```
# include <stdio.h>
main ( ) {
    printf (" ");
    }
```

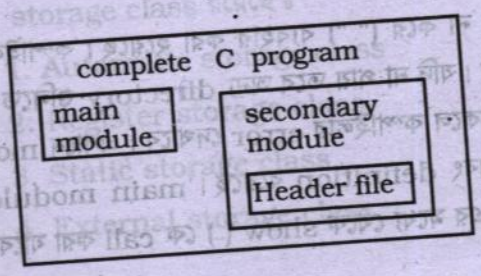
multif 1.c প্রোগ্রামে printf ( ) ফাংশন ব্যবহার করা হয়েছে কারণ এর prototype ডিকলেয়ার করা আছে stdio.h হেডার ফাইলে।

ঠিক এই ভাবে কোন ফাইলে কিছু ফাংশন লিখে আমরা অপর একটি ফাইলে তা include করতে পারি। এভাবে একটি প্রোগ্রামের source code একাধিক ফাইলে লিখে প্রোগ্রাম করাকে modular programming বলে।

এই ধরনের প্রোগ্রামিং-এর সবচেয়ে বড় সুবিধা হল, কোন প্রয়োজনীয় ফাংশন একবার লিখে তা যে কোন সময় যে কোন প্রোগ্রামে # include করা যায় এবং বার বার লেখার প্রয়োজন হয় না।

মডিউলার প্রোগ্রামিং techniques

প্রতিটি C প্রোগ্রামে একটি মাত্র main ( ) ফাংশন থাকে। যে module-এ main ( ) ফাংশন থাকে তাকে main module বলে। এই main module-এর সহিত অন্য কোন module ব্যবহার করলে তাকে secondary modules বলে। Secondary module-এর সহিত সাধারণত header ফাইল সংযুক্ত করা হয়।



modular programming-এর উদাহরণ নিম্নের প্রোগ্রামে দেখানো হল।

Program **main_mod.c**

```
# include <stdio.h>
# include <conio.h>
# include "module.h"
main ( ) {
        int i = 5;
        printf ("%d", show (i));
        }
```

Program **module.h**

```
int show (int i);
int show (int i)
   {
   return (i);
   }
```

OUTPUT 5

main_mod.c প্রোগ্রাম বিশ্লেষণ

```
# include "module.h"
```

এখানে # include-এর পর (< >) ব্যবহার না করে (" ") ব্যবহার করা হয়েছে। কম্পাইলার প্রথমে বর্তমান ডিরেকটরিতে module.h নামে ফাইল খুজবে। যদি না পায় তবে অন্য directory গুলিতে এই নামের ফাইল খুজতে থাকবে। এই নামের কোন ফাইল না থাকলে কম্পাইলার error দেখাবে। এখানে module.h ফাইলের মধ্যে show ( ) ফাংশনের prototype এবং definition রয়েছে। main module-এ module.h ফাইলটি # include করার ফলে main ( ) এর মধ্যে থেকে show ( ) কে call করা যাবে।

STORAGE CLASS

বিভিন্ন টাইপের ভেরিয়েবল নিয়ে পূর্বে আলোচনা করা হয়েছে। প্রতিটি ভেরিয়েবলবে যেমন বিভিন্ন টাইপের ডিকলেয়ার করা যায় তেমনই এদেরকে নির্দিষ্ট strage class-এর অন্তর্ভুক্ত করা যায়।

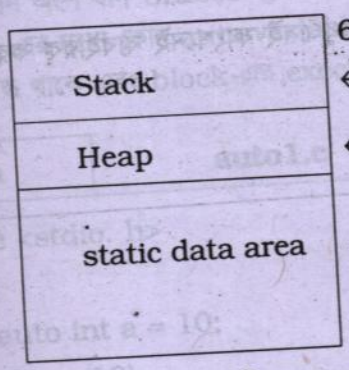
আমরা জানি, ভেরিয়েবল ডিকলেয়ার করলে কম্পাইলার ঐ ভেরিয়েবল এর টাইপ অনুযায়ী মেমোরীতে নির্দিষ্ট স্থান allocate (দখল) করে। কিন্তু, কখনো কখনো ভেরিয়েবল এর জন্য মেমোরীতে জায়গা না নিয়ে CPU register-এ জায়গা allocate (দখল) করার প্রয়োজন হয়। এক্ষেত্রে ভেরিয়েবল এর টাইপ এর পূর্বে storage class উল্লেখ করতে হয়।

storage class কোন ভেরিয়েবলকে দুটি বৈশিষ্ট্য প্রদান করে–

ক. **Lifetime :** কোন ভেরিয়েবল তার মধ্যে রক্ষিত value-কে যতসময় সংরক্ষণ করতে পারে, তাই-ই হল ঐ ভেরিয়েবল-এর life time.

খ. **Scope :** কোন ভেরিয়েবলকে প্রোগ্রামের কোথায় ব্যবহার করা যায় বা যায় না তা ঐ ভেরিয়েবল-এর scope-এর উপর নির্ভর করে।

Intel microprocessor সমৃদ্ধ কম্পিউটারে DOS অপারেটিং সিসটেম memory-কে বিভিন্ন অংশে ভাগ করে। এই প্রতিটি অংশকে বলা হয় segment। কোন্ segment-এ সর্বোচ্চ 65,536 byte স্থান থাকে। C প্রোগ্রামের সমস্ত ভেরিয়েবল ও ফাংশন data segment-এ কিভাবে সংরক্ষিত হয় তা নিম্নের চিত্রে দেখানো হল ঃ



চিত্র ঃ C এ ব্যবহৃত data segment

C-তে মোট চারটি storage class রয়েছে ঃ

1. Automatic storage class
2. Register storage class
3. Static storage class
4. External storage class

Automatic storage class

Auto ভেরিয়েবল মেমোরীতে সংরক্ষিত হয় (রেজিস্টারে নয়)। এটি কোন ফাংশন বা block-এর মধ্যে ডিকলেয়ার করা হয়। যখন কোন ফাংশন call হয় তখন এই ভেরিয়েবল তৈরি হয় আবার ফাংশন execution শেষ হলে এই ভেরিয়েবল ধ্বংস হয়।

Automalic variable-কে local বা internal ভেরিয়েবলও বলা হয়। যে ফাংশন বা block-এ এই ভেরিয়েবল ডিকলেয়ার করা হয়, সেই ফাংশন বা block-এর জন্য ঐ ভেরিয়েবল private অর্থাৎ বাইরে থেকে এ ভেরিয়েবল ব্যবহার করা যাবে না।

Automatic ভেরিয়েবল ডিকলেয়ার করার গঠন নিম্নরূপ ঃ

```
auto data_type variable_name.
```

ফাংশনের মধ্যে কোন ভেরিয়েবল auto ছাড়া ডিকলেয়ার করলেও তা automatic হিসেবে আচরণ করে। যেমনঃ

```
function ( ) {
        int i; ←—— automatic variable.
        }
```

automatic ভেরিয়েবল যে ফাংশনে ডিকলেয়ার করা হয় শুধু সেই ফাংশনেই ব্যবহারক করা যায়। নিম্নের উদাহরণটি লক্ষ্য করুন ঃ

Program **auto.c**

```
main ( ) {
        auto int a = 1;
        function ( ) ;
        printf ("a=%d", a)'
        }
function ( ) {
        auto int a = 10;
        function 1 ( );
        printf ( "function ( ) : a = %d", a);
        }
```

continue

Program auto.c

```
function1 ( ) {
        auto int a = 100;
        printf ("function 1 ( ): a = % d", a);
          }
```

OUTPUT

```
100
 10
  1
```

auto.c প্রোগ্রাম বিশ্লেষণ

❑ এখানে main ( ), function ( ) এবং function 1 ( )-এ a ভেরিয়েবলটি ডিকলেয়ার করা হয়েছে। কম্পাইলার এই তিনটি ফাংশনের তিনটি a- কে তিনটি ভিন্ন ভেরিয়েবল হিসেবে গ্রহণ করবে।

যে ফাংশনের মধ্যে a-এর মান যত দেয়া হয়েছে, printf ( ) a-এর মান তত প্রিন্ট করবে।

প্রোগ্রামের কোন অংশ যদি braces ({.....}) দ্বারা আবদ্ধ রাখা হয় তখন প্রোগ্রামের সেই অংশকে block বলে। Block-এর মধ্যে কোন auto variable ডিকলেয়ার করলে সেই ভেরিয়েবল-এর কাজ ঐ block-এর মধ্যেই সীমাবদ্ধ থাকে এবং block-এর execution শেষ হলে ঐ ভেরিয়েবল ধ্বংস হয়ে যায়।

Program auto1.c

```
# include <stdio. h>
main ( ) {
        auto int a = 10;
        if (a==10)
        {
auto int a = 100;
          {
          auto int a =1000;
              printf ("%d", a);
          }
```

continue

Program auto1.c continue

```
printf ("\n%d", a);
    }
        printf ("\n%d", a);
    }
```

OUTPUT

```
1000
100
10
```

auto1.c প্রোগ্রাম বিশ্লেষণ

এই প্রোগ্রামে তিনবার a ডিকলেয়ার করা হয়েছে। কম্পাইলার তিনটি a-কে তিনটি ভিন্ন ভেরিয়েবল হিসেবে গণ্য করেছে। একারণেই block থেকে বের হওয়া মাত্র a-এর মান পরিবর্তিত হয়েছে।

NOTE

local variabl- কে কম্পাইলার সব সময় auto হিসেবে গণ্য করে। তাই auto1.c প্রোগ্রামে ভেরিয়েবল ডিকলেয়ার করার সময় auto না লিখলেও ভেরিয়েবলগুলো automatic হিসেবে গণ্য হবে।

auto ভেরিয়েবল-এর মান initialize না করে ব্যবহার করলে garbage (আবর্জনা) মান পাওয়া যায়। যেমন ঃ

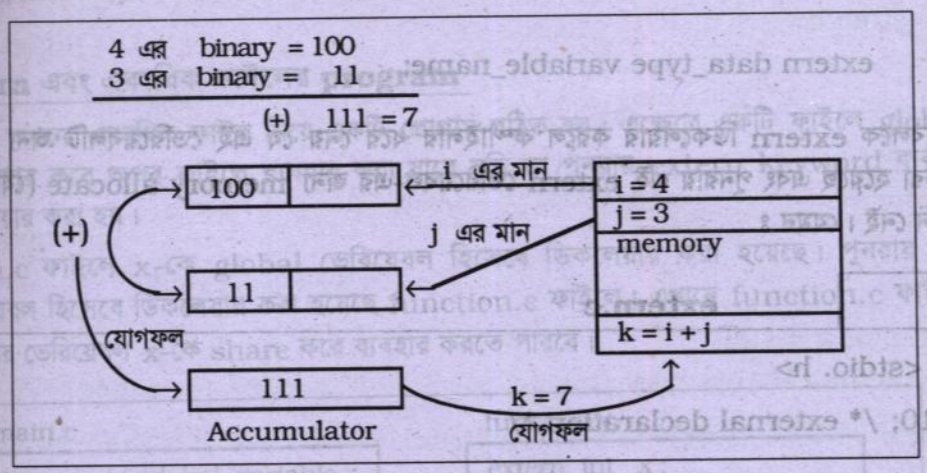
```
auto int a;
printf ("%d", a) // output may be any number
```

Register storage class

Register ভেরিয়েবল মেমোরীতে অবস্থান না করে CPU register-এ অবস্থান করে। কম্পিউটারের Central Processing Unit (CPU)-এ কিছু স্থান আছে যেখানে ডাটা রেখে যোগ, বিয়োগ ইত্যাদি করা হয়। CPU এর এই স্থানগুলোকে register বলা হয়।

Register ভেরিয়েবল ছাড়া অন্যান্য ভেরিয়েবলগুলো memory-তে স্থান নেয়। এদের ডাটার উপর গাণিতিক কাজ করতে হলে ডাটাকে memory থেকে register-এ প্রতিস্থাপিত করতে হয় এবং কাজ শেষে ফলাফল

পুনরায় memory-তে স্থানান্তরিত করা হয়। এতে করে প্রচুর সময় ব্যয় হয়। কিন্তু, register ভেরিয়েবলগুলো CPU-এর register-এ স্থান নেয়ায় এদের উপর দ্রুত গাণিতিক কাজ করা যায়।



চিত্র ঃ মেমোরী ও register-এর data প্রতিস্থাপন।

লুপের ভেরিয়েবলগুলোর মান বারবার বৃদ্ধি বা হ্রাস করতে হয়। তাই loop control ভেরিয়েবলকে register ডিকলেয়ার করলে লুপ execution দ্রুত হয়। যেমন ঃ

```
register int r;
for (r=0; r<=1000; ++r)
    printf ("%d\n", r);
```

যে সব CPU-এর register-এর দৈর্ঘ্য 2 byte (16 bit) সে সব কম্পিউটারে float, double ও long ডাটা টাইপের ভেরিয়েবলকে register টাইপ হিসেবে ডিকলেয়ার করা যায় না। কারণ এসব ভেরিয়েবল এর জন্য 2 byte-এর অধিক memory-এর প্রয়োজন। যেমন ঃ

```
register double a;
```

এখনে a ভেরিয়েবলকে কম্পাইলার auto হিসেবে গণ্য করবে এবং কোন error message দেখাবে না।

Auto ভেরিয়েবলকে initialize না করে ব্যবহার করলে garbage মান দেখা যাবে।

Extern storage class :

main ( ) বা অপর কোন ফাংশনের বাহিরে কোন ভেরিয়েবল ডিকলেয়ার করলে তাকে external variable বলে। একে global ভেরিয়েবলও বলা হয়।

External variable ডিকলেয়ার করার সাথে সাথে কম্পাইলার এর মান শূন্য intialize করে।

কোন ফাংশন যখন external ভেরিয়েবল ব্যবহার করে তখন ঐ ফাংশনের মধ্যে ভেরিয়েবলটি পুনরায় extern keyword ব্যবহার করে ডিকলেয়ার করা উচিত। কোন ভেরিয়েবলকে extern টাইপ ডিকলেয়ার করার গঠন নিম্নরূপ ঃ

```
extern data_type variable_name;
```

কোন ভেরিয়েবলকে extern ডিকলেয়ার করলে কম্পাইলার ধরে নেয় যে এই ভেরিয়েবলটি অন্য কোথাও ডিকয়েলার করা হয়েছে এবং পুনরায় এই extern ভেরিয়েবল-এর জন্য memory allocate (যোগানোর) করার প্রয়োজন নেই। যেমন ঃ

Program extern.c

```
# include <stdio. h>
int ext = 10; /* external declaration */
show ( );
main ( ) {
    extern int ext; /* optional declaration */
    printf ("%d \n", ext);
    show ( ) ;
    getch ( );
    }
show ( ) {
    extern int ext; /* optional declaration */
    printf ("%d", ext);
    }
```

OUTPUT

```
10
10
```

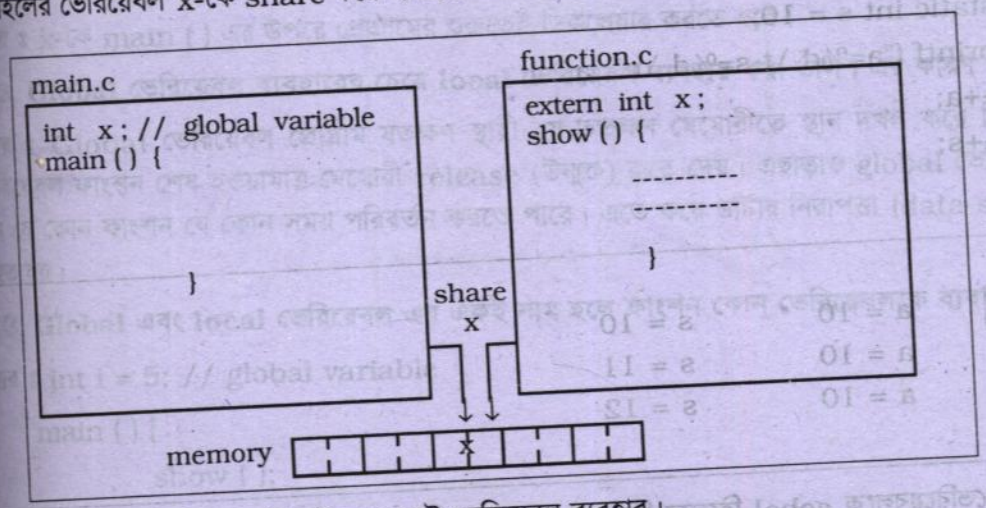
extern. c প্রোগ্রামে ext-কে external (global) ভেরিয়েবল হিসেবে ডিকলেয়ার করে এর মান initialize করা হয়েছে। পুনরায় main ( ) ও show ( )-এর মধ্যে extern হিসেবে ডিকলেয়ার করা হয়েছে। এখন main ( )-এর ext এবং show ( ) এর ext-এর জন্য কোন মেমোরী allocate হয়নি।

extern.c প্রোগ্রামে main ( ) এবং show ( )-এর মধ্যে যদি extern ব্যবহার না করা হত, তবে কম্পাইলার main ( ) এর show ( ) এর ext ভেরিয়েবলকে auto হিসেবে ব্যবহার করত এবং এদের জন্য ভিন্ন ভিন্ন memory ব্যবহার করত।

extern এবং একাধিক ফাইলের program

কখনো কখনো একাধিক ফাইল নিয়ে একটি প্রোগ্রাম গঠিত হয়। এক্ষেত্রে একটি ফাইলে global ভেরিয়েবল ডিকলেয়ার করে অপর ফাইলে ব্যবহার করা যাবে যদি তা পুনরায় extern keyword ব্যবহারের মাধ্যমে ডিকলেয়ার করা হয়।

main.c ফাইলে x-কে global ভেরিয়েবল হিসেবে ডিকলেয়ার করা হয়েছে। পুনরায় একে extern ভেরিয়েবল হিসেবে ডিকলেয়ার করা হয়েছে function.c ফাইলে। এখানে function.c ফাইলটি main.c ফাইলের ভেরিয়েবল x-কে share করে ব্যবহার করতে পারবে।



চিত্র ঃ দুটি ফইলের একই ভেরিয়েবল ব্যবহার।

NOTE

ফাংশনের সাথেও extern keyword ব্যবহার করা যায়। যেমন ঃ

```
extern show ( ) ;
```

Static storage class

static ভেরিয়েবল মেমোরীতে স্থান নেয় এবং কম্পাইলার এর মান শূন্য নির্ধারণ করে যখন এই ভেরিয়েবল ডিকলেয়ার করা হয়।

static ভেরিয়েবল auto ভেরিয়েবল এর মত যে ফাংশনে ডিকলেয়ার করা হয় তার বাহিরে ব্যবহার করা যায় না। তবে ফাংশন execution শেষ হলেও static ভেরিয়েবল এর মান ধ্বংস হয়ে যায় না বরং main ( ) প্রোগ্রাম যতক্ষণ স্থায়ী হয় static ভেরিয়েবলও ততক্ষণ টিকে থাকে।

নিম্নের প্রোগ্রামের মাধ্যমে auto এবং static ভেরিয়েবল এর পার্থক্য দেখানো হল ঃ

Program **static.c**

```
# include <stdio.h>
main ( ) {
        show ( ) ;
        show ( ) ;
        show ( );
        }
show ( ) {
        auto int a = 10;
        static int s = 10;
        printf ("a=%d \t s=%d \n", a, s);
        ++a;
        ++s;
        }
```

OUTPUT

```
a = 10      s = 10
a = 10      s = 11
a = 10      s = 12
```

কোন static ভেরিয়েবলকে gobal হিসেবে ডিকলেয়ার করলে তাকে external static ভেরিয়েবল বলে। একাধিক ফাইলের প্রোগ্রামে external static ভেরিয়েবলকে extern keyword ব্যবহার করে use (ব্যবহার) করা যায় না।

storage class	কোথায় ডিকলেয়ার করা হয়	Lifetime	scope
auto	ফাংশনের মধ্যে	ফাংশন	ফাংশনে
register	ফাংশনের মধ্যে	ফাংশন	ফাংশনে
extern	ফাংশনের বাহিরে	প্রোগ্রাম	একাধিক ফাইলে
static	ফাংশনের মধ্যে	প্রোগ্রাম	ফাংশনে
static (external)	ফাংশনের বাহিরে	প্রোগ্রাম	একটি ফাইলে

চিত্র ঃ storage class-এর সারাংশ।

Q & A :

Q.1 নিম্নের প্রোগ্রামে ভুল কোথায়?

```
main ( ) {
        x=1;
        }
int x;
show ( ) {
        printf ("%d", x);
        }
```

উত্তর ঃ x-কে main ( ) এর উপরে প্রোগ্রামের শুরুতেই ডিকলেয়ার করতে হবে।

Q.2. Global ভেরিয়েবল ব্যবহারের চেয়ে local ভেরিয়েবল ব্যবহার করা ভাল। এর কারণ কি?

উত্তর ঃ Global ভেরিয়েবল প্রোগ্রাম যতক্ষণ স্থায়ী হয় ততক্ষণ মেমোরীতে স্থান দখল করে কিন্তু local ভেরিয়েবল ফাংশন শেষ হওয়ামাত্র মেমোরী release (উন্মুক্ত) করে দেয়। এছাড়াও global ভেরিয়েবল এর মান যে কোন ফাংশন যে কোন সময় পরিবর্তন করতে পারে। এতে করে ডাটার নিরাপত্তা (data security) ব্যহত হয়।

Q.3. Global এবং local ভেরিয়েবল এর একই নাম হলে ফাংশন কোন্ ভেরিয়েবলকে ব্যবহার করবে?

উত্তর ঃ

```
int i = 5; // globai variable
main ( ) {
        show ( );
        }
show ( ) {
        int i = 10; // local variable
        printf ("%d", i); // print local value of i
        }
```

OUTPUT 10→ লোকাল variable-এর মান।

ফাংশন local ভেরিয়েবলকে ব্যবহার করবে।

Q.4. একাধিক ফাইলে কোন variable-কে share করতে হলে তার storage class কি হবে?

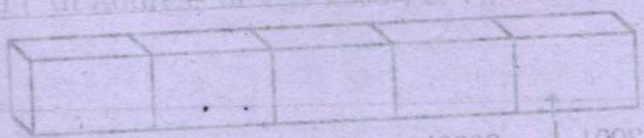
উত্তর ঃ extern.

Exercise

1. auto এবং static ভেরিয়েবলের পার্থক্য কি?
2. register ভেরিয়েবল কখন ব্যবহার করা হয়?
3. ভেরিয়েবল-এর scope বলতে কি বোঝায়?
4. External static ভেরিয়েবলকে একাধিক ফাইলে share করা যায় কি?
5. গ্লোবাল ভেরিয়েবল ডিকলেয়ার করার সাথে সাথে এর মান কত intialize হয়?
6. Storage class-এর উপর নির্ভর করে ভেরিয়েবল মেমোররীর বিভিন্ন স্থানে অবস্থান করে কি?
7. Global এবং extern ভেরিয়েবল এর পার্থক্য কি?
8. রেজিস্টার কি?
9. CPU কি?
10. long টাইপ ভেরিয়েবলকে register-এ store করা যায় কি?

8

Pointers (পয়েন্টার)



- **Pointer**
- **Pointer এবং array**
- **Pointer এবং function**
- **Examples**

POINTERS

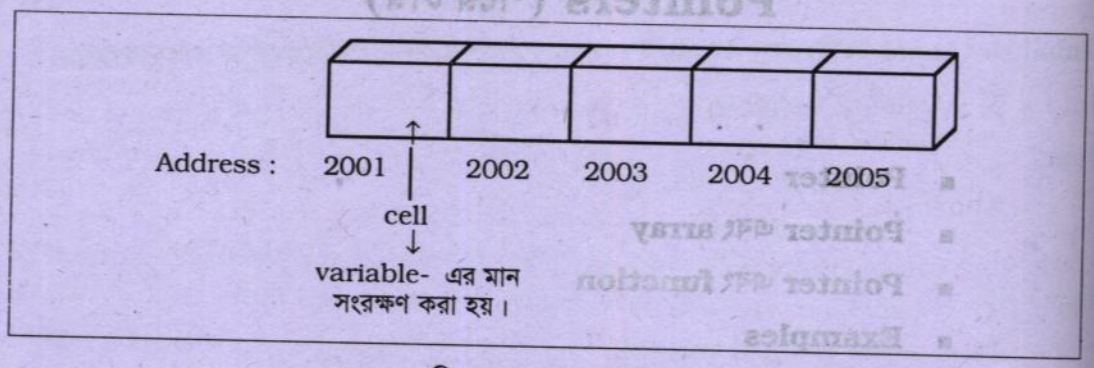
Pointer C প্রোগ্রামের একটি জটিল পদ্ধতি। আলোচ্য অধ্যায়ে মেমোরী address, pointer, array ও pointer, pointer ও ফাংশন, ইত্যাদি বিষয়গুলো নিয়ে বিস্তারিত আলোচনা করা হয়েছে।

Pointer সম্বন্ধে জানতে হলে প্রথমেই variable এবং memory নিয়ে আলোচনা করতে হবে।

ভেরিয়েবল এবং মেমোরী (Variable associated address) ঃ

মেমোরী অসংখ্য কম্পিউটারের cell বা ছোট ছোট ঘর নিয়ে গঠিত। এই cell গুলোর প্রতিটির নির্দিষ্ট address থাকে।

প্রোগ্রামে variable ডিকলেয়ার করলে compiler-এ ভেরিয়েবল এর জন্য নির্দিষ্ট মেমোরী cell দখল নেয় এবং ঐ cell-এর address-কে ভেরিয়েবল এর সাথে সংযুক্ত করে দেয়। এরপর প্রোগ্রামের কোথাও variable ব্যবহার করলে compiler সাথে সাথে ঐ varible-এর জন্য দখল করা cell-এর address-এ পৌছায়। এখানে লক্ষণীয় যে, আমরা প্রোগ্রামে variable ব্যবহার করি কিন্তু ঐ variable-এর জন্য নির্দিষ্ট করা address নিয়ে আমাদের চিন্তা করতে হয় না। বরং compiler এই বিষয়গুলো নিয়ন্ত্রণ করে। কিন্তু, pointer পদ্ধতি ব্যবহার করে ভেরিয়েবল এর address নিয়ে সরাসরি কাজ করা যায়।



চিত্র ঃ memory cell

নিম্নের variable ডিকলেয়ারেশন এবং এর মান নির্ধারণ লক্ষ্য করুন ঃ

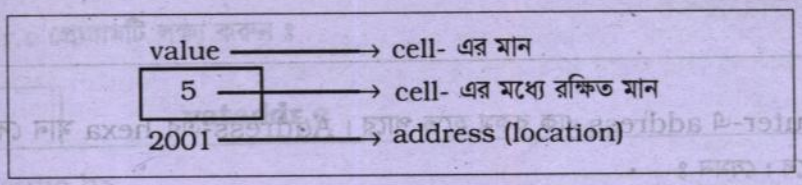
```
int value = 5;
```

এই statement-এর জন্য compiler নিম্নোক্ত কাজ করবে–

(ক) int টাইপ মান রাখার জন্য মেমোরীতে জায়গা নেয়া হয়।

(খ) ভেরিয়েবল এর নাম value-এর সাথে মেমোরী address টি সংযুক্ত (associate) করা হয়।

(গ) value-এর জন্য রক্ষিত মেমোরী address-এ 5 রাখা হয়।



চিত্র ঃ মেমোরীতে variable

কোন ভেরিয়েবল-এর address দেখার জন্য ভেরিয়েবল-এর পূর্বে & অপারেটর ব্যবহার করা হয়। নিম্নের প্রোগ্রাম variable-এর মান এবং address উভয়ই দেখাবে ঃ

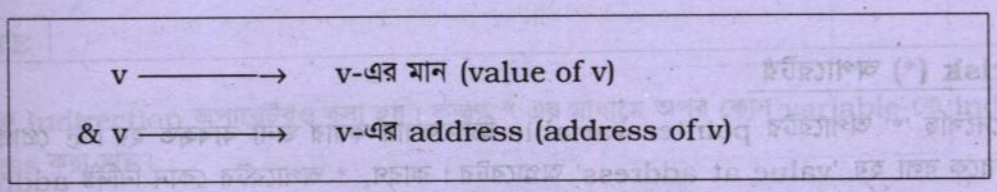
Program **addr.c**

```
# include <stdio.h>
main ( ) {
        int V = 5;
        printf ("Value of V is : %d", V);
        printf ("\n Address of V is : %u", & V);
        }
```

OUTPUT

```
Value of V is : 5
Address of V is : 2001
```

উপরোক্ত addr.c প্রোগ্রামে printf ( )-এর মধ্যে &v ব্যবহার করার ফলে V-এর address দেখা সম্ভব হয়েছে।



NOTE

এক এক computer-এ address এক রকম হতে পারে। Address-এর hexa মান দেখতে হলে %x ব্যবহার করতে হবে। যেমন ঃ

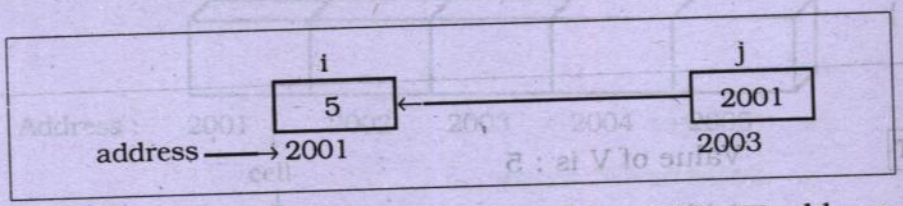
```
printf ("Address of V is : %x", & V);
```

Pointer Variable :

Pointer variable অপর কোন variable-এর address সংরক্ষণ করে (Pointer variable contains address of another variable) pointer variable ডিকলেয়ার করার জন্য * (asterisk) অপারেটর ব্যবহার করা হয়। যেমন ঃ

```
int i = 5;      //initialize variable
int *j;         // declare pointer variable
j = &i;         // j contains address of i
```

এই statement গুলোকে নিম্নোক্ত চিত্রের মাধ্যমে দেখানো যায় ঃ



চিত্র ঃ pointer variable-এর মান হল অপর কোন variable-এর address

এখানে i একটি int টাইপ ভেরিয়েবল এবং এর মান হল 5 এবং address 2001. j এখানে int টাইপ pointer vaviable বার মধ্যে i-এর address সংরক্ষিত আছে (2001) এবং নিজস্ব এর address 2003। অর্থাৎ pointer variable j-এর মান হল i-এর address। এখানে লক্ষণীয় যে, j-এর মান যেহেতু i-এর address, সেহেতু j-এর মাধ্যমে i-এর address-এ পৌছে i-এর মান দেখা যাবে।

Asterisk (*) অপারেটর

আমরা দেখেছি '*' অপারেটর pointer variable ডিকলেয়ার করার জন্য ব্যবহৃত হয়। c প্রোগ্রামে * অপারেটরকে বলা হয় 'value at address' অপারেটর। কারণ, * অপারেটর কোন নির্দিষ্ট address-এ রক্ষিত মানকে দেখানোর জন্য ব্যবহৃত হয়।

নিম্নের vataddr.c প্রোগ্রামটি লক্ষ্য করুন ঃ

Program **vataddr.c**

```
# include <stdio.h>
main ( ) {
        int V = 5;
        printf ("Address of V = %u", & V);
        printf ("\n Value of V=%d", V);
        printf ("\n Value at address V = %d:", * (& V));
        }
```

OUTPUT

```
Address of V = 2001
Value of V = 5
Value at address V = 5
```

এখানে vataddr.c প্রোগ্রামে printf ( )-এর মধ্যে * (&v) ব্যবহার করার ফলে v-এর address-এ রক্ষিত v-এর মান দেখা সম্ভব হয়েছে। অর্থাৎ * (&v) এবং v একই expression-এর দুটি ভিন্ন রূপ।

v	⟶	v-এর মান (value of v)
& v	⟶	v-এর address (address of v)
*(&v)	⟶	v-এর address-এর মান (value at address of v)

NOTE

'*' কে indirection অপারেটরও বলা হয়। কারণ * এর মাধ্যমে অপর কোন variable-কে indirectly access করা যায়।

Pointer ডিকলেয়ারেশন

Pointer ভেরিয়েবল ব্যবহার করার পূর্বে অবশ্যই ডিকলেয়ার করতে হবে। Pointer variable ডিকলেয়ার করার নিয়ম নিম্নরূপ ঃ

```
data_type * variable_name
```

pointer variable-এর নামের পূর্বে অবশ্যই '*' ক্যারেক্টার লিখতে হবে। তবে মনে রাখতে হবে '*' ক্যারেক্টারটি variable-এর নামের ঠিক পূর্বে লিখলেও '*' মূলত data_type-এর অংশ। যেমন ঃ

```
int *v;
```

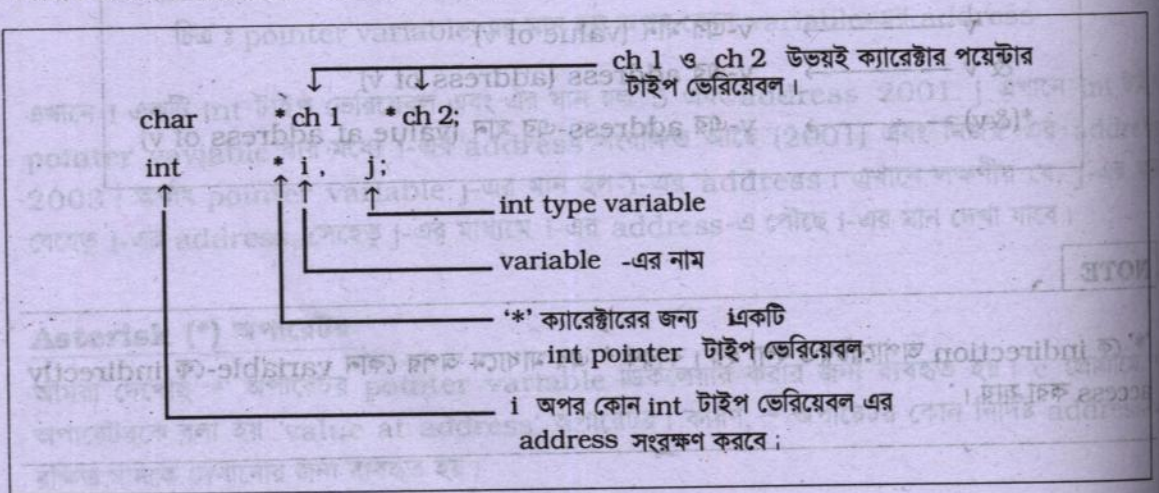
এখানে, v হল int ডাটা টাইপ pointer-এর একটি variable। এ কথার অর্থ হল, v এমন একটি int টাইপ pointer ভেরিয়েবল যে অপর কোন int টাইপ variable-এর address সংরক্ষণ করতে পারবে। c-তে যে কোন ডাটা টাইপ-এর pointer variable ডিকলেয়ার করা যায়। যেমন ঃ

```
int *i;
char * ch;
float *fl;
```

NOTE

কোন Pointer Variable-কে যে ডাটা টাইপ-এর ডিকলেয়ার করা হবে, সে শুধুমাত্র সেই ডাটা টাইপের variable-এর address সংরক্ষণ করতে পারবে।

নিম্নোক্তভাবেও Pointer ডিকলেয়ার করা যায় ঃ



চিত্র ঃ Pointer declaration

INITIALIZING POINTERS (Pointer-এর মান নির্ধারণ)

কোন Pointer variable ব্যবহারের পূর্বে তার মান নির্ধারণ করতে হবে। Uninitialized pointer ব্যবহার করলে প্রোগ্রামের ফলাফল অপ্রত্যাশিত হবে। Pointer variable নিম্নোক্তভাবে initialize করতে হবে ঃ

```
Pointer = & variable;
```

যেমন ঃ
```
int i, *p;
p = & i;
```

Pointer variable ডিকলেয়ার করার সময় initialize করা যায়। যেমন ঃ

```
int i, *p = & i;
```

যে ভেরিয়েবল-এর address pointer variable-এ রাখবো, সেই variable-কে অবশ্যই pointer variable-এর পূর্বে declare করতে হবে। যেমন নিম্নের statementগুলো ভুল ঃ

```
int * p = & i, i;
int * j = & q
int q;
```

Pointer variable যে টাইপ-এর, সে ঠিক সেই টাইপের variable-এর address সংরক্ষণ করতে পারবে। যেমন নিম্নের statements ভুল ঃ

```
char * ch;
int i;
ch = &i; //Error, different type.
```

pointer variable-কে initialize না করে এর মান নির্ধারণ করলে কম্পিউটারের মেমোরীর যে কোন অজানা অংশে এই মান সংরক্ষিত হয়। যেমন—

```
* i = 10; // doin't use like this
```

তবে char pointer-কে সরাসরি নিম্নোক্তভাবে মান নির্ধারণ করা যায় ঃ

```
char * string = "Hello! Ricky Martin";
```

Pointer-এর মাধ্যমে Variable-এর মান দেখা ঃ

Pointer variable-এর মধ্যে অপর কোন variable-এর address রাখলে Pointer variable-এর পূর্বে indirection operator (*) ব্যবহার করে ঐ variable-কে access (ব্যবহার) করা যাবে। যেমন ঃ

```
int x = 10;
int *p;
p = &x;
printf ("x = %d", *p)
```

এখানে *p ব্যবহার করার ফলে output 10 দেখা যাবে কারণ x = 10 এবং *P-এর মধ্যে x-এর address রয়েছে এবং *P-এর মাধ্যমে x-কে access করা হয়েছে।

নিম্নের প্রোগ্রামটি লক্ষ্য করুন ঃ

Program **varacces.c**

```
# include <stdio.h>
# include <conio.h>
main ( ) {
        char ch, * pch;
        ch = 'T';
        pch = & ch;
        printf ("Address of ch stored in pch is : %u\n", pch);
        printf ("Using pointer value of ch is : %c\n", *pch);
        printf ("pch's own address is : %u", & pch);
        getch ( ) ;
        }
```

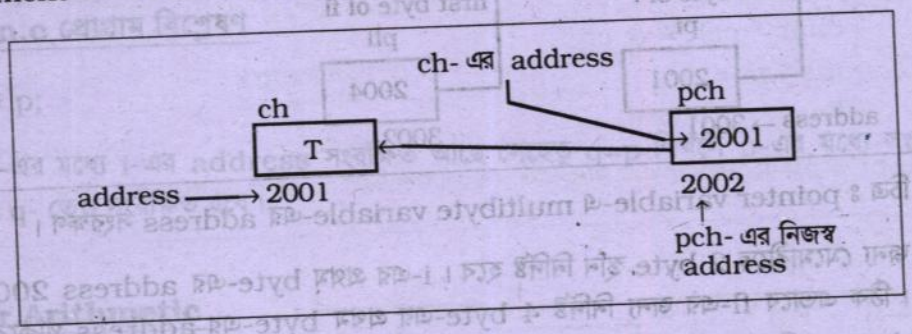
OUTPUT

```
Address of ch stored in pch is : 2001
(Using pointer) value of ch is : T
pch's own address is : 2002
```

varacces.c প্রোগ্রাম বিশ্লেষণ

❑ char ch, * pch;
ch = 'T';
pch = & ch;

এই statement গুলোকে নিম্নোক্ত চিত্রের মাধ্যমে দেখানো যায় ঃ



❑ printf ("....", pch);

এখানে pch লেখার জন্য pch-এর মধ্যে রক্ষিত ch-এর address 2001 দেখা যাবে।

❑ printf ("....", * pch);

এখানে printf ( )-এর মধ্যে pch-এর পূর্বে * ব্যবহারের ফলে ch-কে access (ব্যবহার) করা যাবে। *pch লেখার ফলে ch-এর মান T দেখা যাবে।

❑ printf ("....", & pch);

pch-এর পূর্বে address operator ব্যবহারের ফলে pch-এর নিজস্ব address দেখা যাবে।

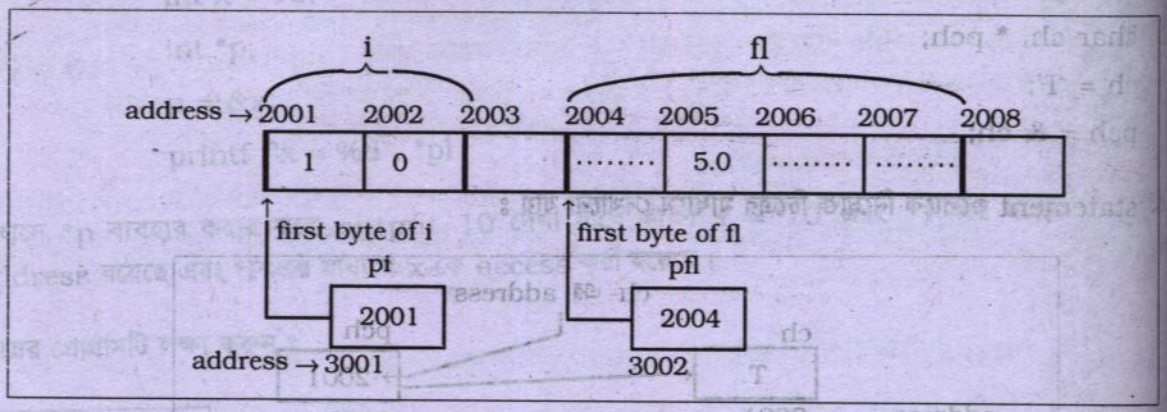
Pointer এবং Variable type

কোন ভেরিয়েবল int টাইপ হলে 2 byte, char টাইপ হলে 1 byte এবং float টাইপ হলে 4 byte যায়গা মেমোরীতে দখল করে। এখন প্রশ্ন জাগতে পারে যে, যে সকল ভেরিয়েবল মেমোরীতে একাধিক byte দখল করে তাদের address pointer variable কিভাবে সংরক্ষণ করবে।

মূলত pointer variable যে ভেরিয়েবলকে point করে, তার প্রথম (সর্বনিম্ন) byte-এর address-কে সে সংরক্ষণ করে। যেমন ঃ

```
int i = 10;
float fl = 5.0;
int *pi = &i;
float * pfl = & fl;
```

এই statement গুলোকে নিম্নোক্ত চিত্রের মাধ্যমে দেখানো হল ঃ



চিত্র ঃ pointer variable-এ multibyte variable-এর address সংরক্ষণ।

এখানে i-এর জন্য মেমোরীতে 2 byte স্থান নির্দিষ্ট হবে। i-এর প্রথম byte-এর address 2001 থাকবে pi-এর মধ্যে। ঠিক এভাবে fl-এর জন্য নির্দিষ্ট 4 byte-এর প্রথম byte-এর address থাকবে pfl-এর মধ্যে (2004)।

Pointer Assignments

কোন pointer variable-এর মান হিসেবে অপর কোন pointer variable ব্যবহার করা যেতে পারে। অর্থাৎ, assignment statement-এর বাম দিকের কোন pointer variable-এর মান নির্ধারণের জন্য assignment statement-এর ডান দিকে pointer variable ব্যবহার করা যাবে। নিম্নের প্রোগ্রামটি লক্ষ্য করুন ঃ

Program **ptrasgn.c**

```
# include <stdio.h>
main ( ) {
        int i = 10;
        int *p, *q;
        p = & i;
        q = p;
        printf ("Address of i = %u", &i);
        printf ("\n Address of i in p = %u", p);
        printf ("\n Address of i in q = %u", q);
        }
```

OUTPUT

```
Address of i = 2001
Address of i in p = 2001
Address of i in q = 2001
```

ptrasg n.c প্রোগ্রাম বিশ্লেষণ

❑ `q = p;`

যেহেতু, p-এর মধ্যে i-এর address সংরক্ষিত আছে সেহেতু q=p লিখলে p-এর মধ্যে সংরক্ষিত i-এর address q- তেও সংরক্ষিত হবে।

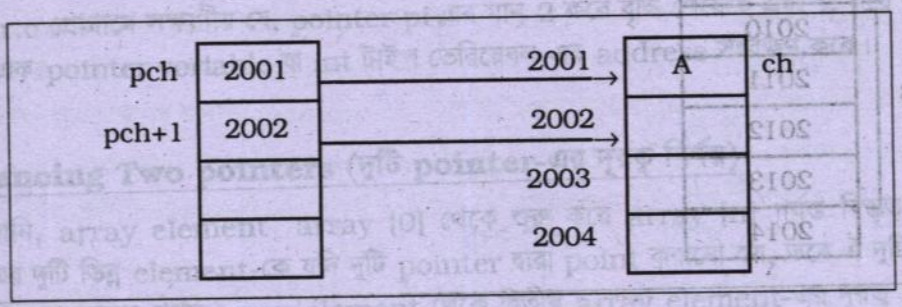
Pointer Arithmetic

pointer-এর উপর শুধুমাত্র দুটি গাণিতিক কাজ করা যায়। এবং তা হল যোগ এবং বিয়োগ (বা increment এবং decrement)। এখানে উল্লেখ্য যে, pointer-এর উপর গুণ বা ভাগের কাজ করা যায় না।

আমরা জানি pointer variable অপর কোন variable-এর address সংরক্ষণ করে। তাই যখন কোন pointer variable-এর সহিত 1 যোগ করা হয় তখন তার মধ্যে সংরক্ষিত মান বৃদ্ধি পায় যা পরবর্তি মেমোরী location- কে point করে। যেমন ঃ

```
char    ch, * pch;
        ch = 'A';
        pch = & ch;
        ++pch; // pch+1
```

উপরের statementগুলো নিম্নোক্ত চিত্রের মাধ্যমে দেখানো হল ঃ

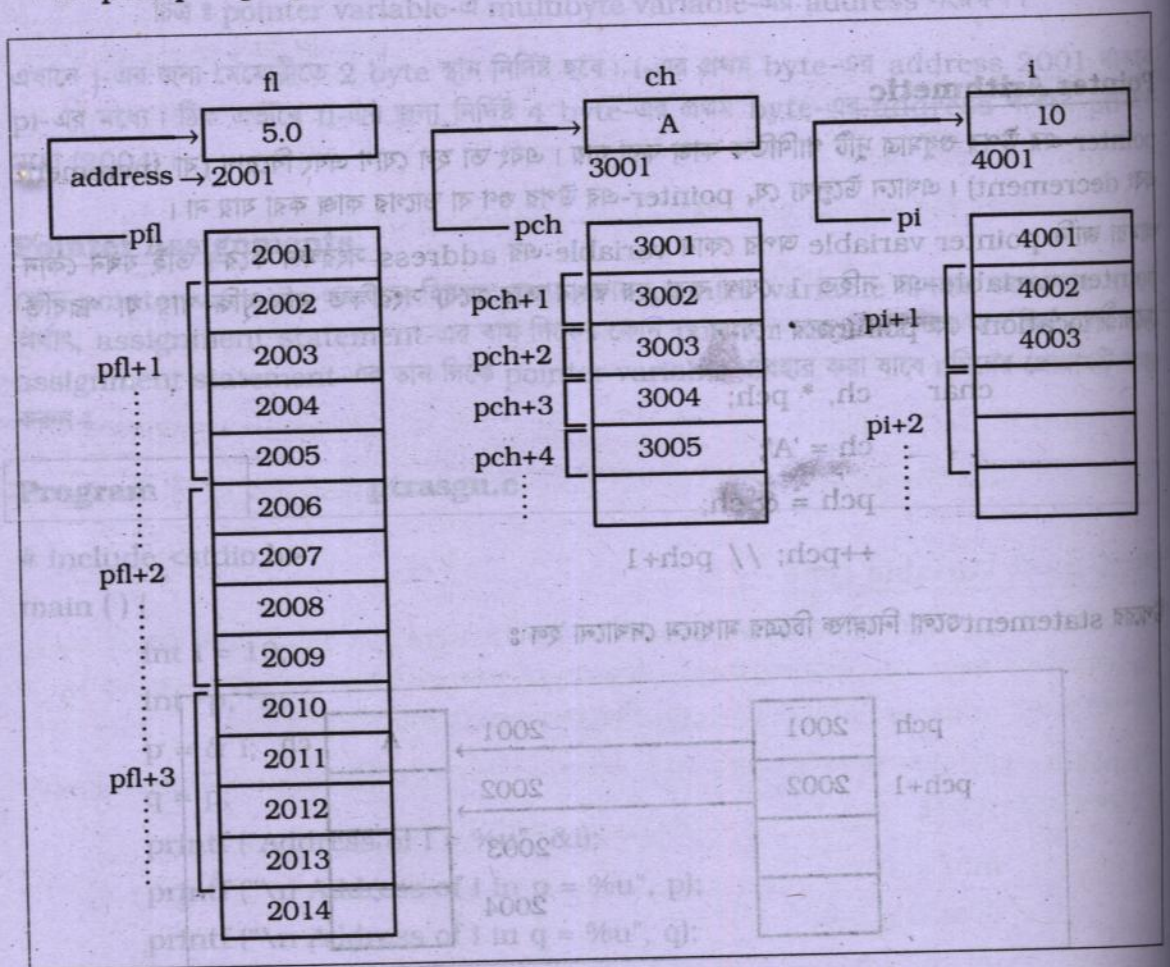


চিত্র ঃ pointer increment

কোন pointer-কে increment (বৃদ্ধি) করলে তার মান বৃদ্ধি পায় base data type অনুযায়ী, অর্থাৎ আমরা জানি, int টাইপ ডাটা সংরক্ষণের জন্য 2 byte মেমোরীর প্রয়োজন। pointer-এর মধ্যে int টাইপ ভেরিয়েবল-এর address রাখার পর ঐ pointer-এর সহিত 1 যোগ করলে তার মান দুই বৃদ্ধি পাবে। char ডাটা টাইপের বেলায় 1 ও float-এর বেলায় 4 করে বৃদ্ধি পাবে। যেমন ঃ

```
float fl = 5.0;
char ch = 'A';
int i = 10;
char * pch = & ch
int * pi = & i;
float *pfl = & fl;
```

এখানে pch, pi ও pfl pointer-এর মান বৃদ্ধি করলে যথাক্রমে 1, 2 ও 3 করে বৃদ্ধি পাবে।



চিত্র ঃ pointer-এর মান বৃদ্ধি হয় ডাটা টাইপ অনুযায়ী।

pointer-এর মান যেভাবে বাড়ানো যায় ঠিক একই নিয়মে (- -) অপারেটর ব্যবহার করে কমানোও যায়। যেমন ঃ

```
--pi;
--pch;
```

নিম্নের প্রোগ্রামটি লক্ষ্য করুন ঃ

Program **incrdecr.c**

```
# include <stdio.h>
main ( ) {
    int i = 1;
    int *pi = & i;
    printf ("Now address is : %u", pi);
    printf ("\n After Increment address is : %u", ++pi);
    printf ("\n After decrement address is : %u", --pi);
    }
```

OUTPUT

```
NOW address is : 2000
After Increment address is : 2002
After decrement address is : 2000
```

incrdecr.c প্রোগ্রামে লক্ষ্যণীয় যে, pointer pi-এর মান 2 করে বৃদ্ধি পেয়েছে এবং 2 করে কমেছে। কারণ pi এমন এক pointer voriable যা int টাইপ ভেরিয়েবল-এর address সংরক্ষণ করে।

Differencing Two pointers (দুটি pointer-এর দূরত্ব নির্ণয়)

আমরা জানি, array element array [0] থেকে শুরু করে array [n] পর্যন্ত বিস্তৃত থাকে। একই array-এর দুটি ভিন্ন element-কে যদি দুটি pointer দ্বারা point করানো হয়, তবে ঐ দুটি pointer-কে পরস্পর বিয়োগ করলে প্রথম array element থেকে দ্বিতীয় array element-এর দূরত্ব বের করা যায়। যেমন ঃ

```
char ch [10]
char * pch1, *ch2;
pch1 = ch; // points to base address of array
pch2 = pch 1+5; /* pch1=&ch[0] is equvalent to pch1 = ch */
```

এখন কোন statement-এ যদি pch2—pch1 ব্যবহার করি তবে output 5 হবে। কারণ ch [0] থেকে ch [5]-এর দুরত্ব 5 ঘর।

```
printf ("%d", pch2-pch1);
```

Pointer comparisons (pointer ভেরিয়েবলের তুলনা)

একই array-এর দুটি ভিন্ন element-কে যদি দুটি pointer দ্বারা point করা হয়, তবে ঐ দুটি pointer কে পারস্পরিক তুলনা করা যায়। যেমন ঃ

```
if (pch2>pch1) {.....}
if (pch2! = pch1) {.....}
if (pch1 <=pch2) {.....}
```

Pointer Expressions

Pointer variable-এর মাধ্যমে expression তৈরি করা যায়। যেমন ঃ

```
int a, b, c;
int *p, *q;
int total = 0;
a = 10;
b = 20;
c = 30;
    p = & a;
    q = & b;
    c = * p**q; // (*p) * (*q)
    total = a+(*p);
    total + = *q;
```

Example :

Direct access/Indirect access : নিম্নের প্রোগ্রামে i-এর মানকে প্রথমে সরাসরি এবং পরে Pointer variable-এর মাধ্যমে access করা হয়েছে ঃ

Program **dirindir.c**

```
# include <stdio.h>
main ( ) {
        int i = 10;
        int *ptr = &i;
        printf ("Value of i (Direct access) : %d", i);
        printf ("\n Value of i (indircct access) : %d", * ptr);
        }
```

POINTER & ARRAY

আমরা জানি, সাধারণ ভেরিয়েবল নিয়ে কাজ করার সময় pointer ব্যবহার করলে প্রোগ্রামের উপযোগিতা বৃদ্ধি পায়। কিন্তু, array নিয়ে প্রোগ্রাম করার সময় আমরা যদি pointer ব্যবহার করি, তবে program-এর efficiency অনেক বেড়ে যায়। Pointer ব্যবহার করে array-এর কোন নির্দিষ্ট element-কে access করা যায়। নিম্নে pointer এবং array নিয়ে বিস্তারিত আলোচনা করা হল ঃ

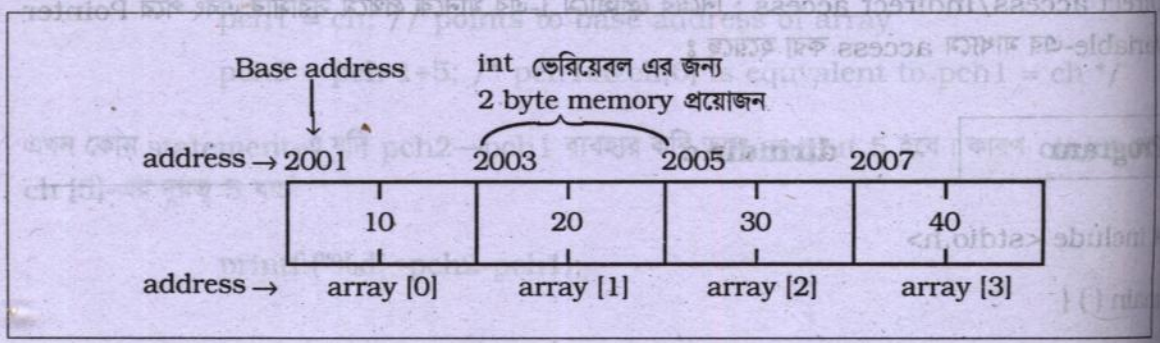
Array-এর pointer

যখন কোন array ডিকলেয়ার করা হয়, তখন কম্পাইলার array-এর প্রথম element (array [0])-এর জন্য মেমোরীতে স্থান নেয় এবং এর address array-এর নাম এর সহিত সংযুক্ত করে। array-এর প্রথম element-এর পরেরগুলোর জন্যও memory-তে পরস্পর সংলগ্ন জায়গা নির্দিষ্ট করা হয়।

কোন array-এর প্রথম element-এর address-কে base address বলা হয়। কোন array ডিকলেয়ার করলে কম্পাইলার এই base address-কেই array-এর নামের সাথে সংযুক্ত করে। যেমন ঃ

```
int array [4] = {10,20,30,40};
```

এখানে, array-এর array [0]-এর address যদি 2001 হয় তবে একে নিম্নোক্ত চিত্রের মাধ্যমে দেখানো যায় ঃ



চিত্র ঃ int array-এর base address

উপরের চিত্রানুযায়ী

```
& array [0] = 2001          // address of array [0] is 2001
& array [1] = 2003
& array [2] = 2005
& array [3] = 2007
```

প্রোগ্রামে যদি কোন element ব্যবহার না করে শুধুমাত্র নাম ব্যবহার করা হয় তবে তা হবে array-এর প্রথম element-এর address। যেমন—

```
array = & array [0] = 2001
```

এখানে array হল একটি constant pointer যা array [0]-কে point করবে।

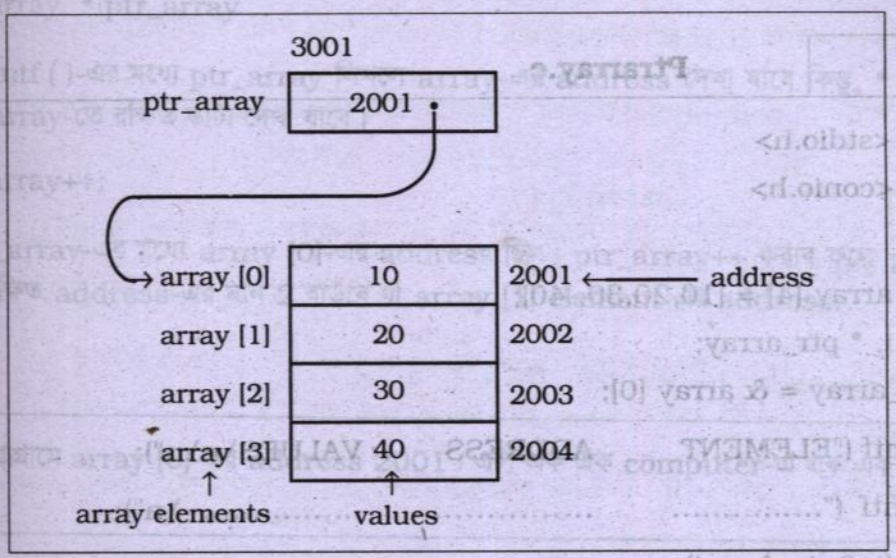
এখন ptr_array নামে কান pointer ডিকলেয়ার করলে এই pointer-এর মাধ্যমে নিম্নোক্তভাবে array-কে point করা যাবে।

```
int *ptr_array;
ptr_array = array;
```

এখানে ptr-array অ্যারের প্রথম element (array [0])-এর address সংরক্ষণ করবে। একে নিম্নোক্তভাবেও লেখা যায়—

```
ptr_array = & array [0];
```

উপরোক্ত statement গুলোকে নিম্নোক্ত চিত্রের মাধ্যমে দেখানো যায় ঃ



চিত্র ঃ Pointer-এর মাধ্যমে array কে Point করা।

Pointer-এর মাধ্যমে array-কে ব্যবহার ঃ

pointer-এর মধ্যে array-এর প্রথম element-এর address রেখে pointer-এর মাধ্যমে array-এর বিভিন্ন element কে ব্যবহার করা যায়। যেমন ঃ

ptr_array = & array [0]; ——→ address 2001
ptr_array+1 = & array [1]; ——→ address 2003
ptr_array+2 = & array [2]; ——→ address 2005
ptr_array+3 = & array [3]; ——→ address 2007

উপরে আমরা দেখলাম pointer কিভাবে কোন নির্দিষ্ট array element-কে নির্দেশ (point) করে। এবার আমরা দেখবো pointer-এর মাধ্যমে কিভাবে কোন array-এর ডাটা বা তথ্যকে ব্যবহার করা যায় ঃ

*ptr_array = array [0]-এর ডাটা (10)
* (ptr_array +1) = array [4]-এর ডাটা (20)
* (ptr_array +2) = array [2]-এর ডাটা (30)
* (ptr_array +3) = araay [3]-এর ডাটা (40)

অর্থাৎ ptr-array যদি array [0]-কে point করে তবে array-এর যেকোন element-এর ডাটাকে ptr_array-এর মাধ্যমে access (ব্যবহার) করা যাবে।

নিম্নের প্রোগ্রামে pointer-এর মাধ্যমে কিভাবে array-এর address এবং ডাটাকে ব্যবহার করা যায় তা দেখানো হল ঃ

Program **Ptrarray.c**

```
# include <stdio.h>
# include <conio.h>
main ( ) {
        int array [4] = {10,20,30, 40};
        int i, * ptr_array;
        ptr_array = & array [0];
        printf ("ELEMENT        ADDRESS      VALUES\n\n");
        printf ("..............        ..............  .............. \n");
        for (i=0; i <4; ++i)
                {
                printf ("array [%d]     %u     %d \n", i, ptr_array, * ptr_array);
                ptr_array++;  /* increment pointer to point to the next array
                                location */
                }
        getch ( );
        }
```

OUTPUT

ELEMENT	ADDRESS	VALUE
array [0]	2001	10
array [1]	2003	20
array [2]	2005	30
array [3]	2007	40

ptrarray.c প্রোগ্রাম বিশ্লেষণ

- ptr_array, * ptr_array

এখানে printf ( )-এর মধ্যে ptr_array লিখলে array-এর address দেখা যাবে কিন্তু, * ptr_array লিখলে ঐ array-তে রক্ষিত ডাটা দেখা যাবে।

- ptr_array++;

প্রথমে ptr_array-এর মধ্যে array [0]-এর address ছিল। ptr_array++ করার ফলে ptr_array-এর মধ্যে রক্ষিত address-এর মান 2 বাড়বে যা array [1] element-এর address.

NOTE

উপরোক্ত প্রোগ্রামে array [0]-এর address 2001। এটা এক এক computer-এ এক এক রকম হতে পারে।

কোন array element-এর memory address নিম্নোক্তভাবে নির্ণয় করা যায় ঃ

address of array [n] = base address + (n * scale factor of data type)

যেমন ঃ address of array [5] = 2001+ (5&2)

এখানে scale factor হল কোন ডাটা টাইপের জন্য memory-তে কত byte স্থানের প্রয়োজন, তার সংখ্যাগত মান। যেমন ঃ

scale factor of int = 2
scale factor of char = 1
scale factor of float = 4
scale factor of double = 8

pointer ব্যতীত '*'-এর মাধ্যমে array ব্যবহার

আমরা জানি যে, শুধুমাত্র array-এর নাম হল array-এর base address. যেমন ঃ

int number [5] = { 10, 20, 30, 40, 50 };

element ⟶	[0]	[1]	[2]	[3]	[4]
Value ⟶	10	20	30	40	50
address ⟶	2001	2003	2005	2007	2009

এখানে আমরা যদি শুধু number উল্লেখ করি তবে number [0]-এর address পাবো। কিন্তু, *number লিখলে number [0]-এর মধ্যে সংরক্ষিত মান 10-কে access করতে পারবো। তেমনিভাবে *(number+1) লিখলে 20 দেখতে পাবো।

তাহলে দেখা যাচ্ছে যে pointer ছাড়া শুধুমাত্র '*' অপারেটর ব্যবহার করে array নিয়ে কাজ করা যায় ঃ

Program **astersk.c**

```
# include <stdio.h>
main ( ) {
    int number [ ] = {10, 20, 30, 40};
    int i;
    for (i=0; i <4; ++i)
        {
        printf ("\n Address =%u", & number [i]);
        printf ("Value = %d", number [i]);
        printf ("%d", * (number +i));
        printf ("%d", * (i+number));
        printf ("%d", i [number]);
        }
}
```

OUTPUT

```
Address = 2001 Value = 10 10 10 10
Address = 2003 Value = 20 20 20 20
Address = 2005 Value = 30 30 30 30
Address = 2007 Value = 40 40 40 40
```

Pointers to Pointers

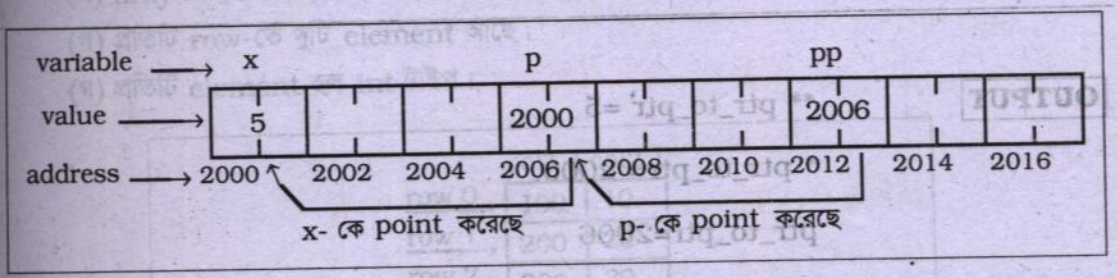
কোন Pointer variable অপর কোন variable-এর address সংরক্ষণ করে। যেমন ঃ

```
int x = 5, *p;
p = &x; // p-এর মধ্যে x-এর address সংরক্ষিত হল।
```

pointer to pointer ভেরিয়েবল অপর কোন pointer variable-এর address সংরক্ষণ করে। যেমন ঃ

```
int x = 5, *p, **pp;
p = &x; // p-এর মধ্যে x-এর address সংরক্ষিত হল
pp = &p; // pp-এর মধ্যে p-এর address সংরক্ষিত হল
```

উপরের statement গুলোকে নিম্নোক্ত চিত্রের মাধ্যমে দেখানো যায় ঃ



চিত্র ঃ p-এর মধ্যে x-এর address এবং pp-এর মধ্যে p-এর address রয়েছে।

এবার আমরা যদি x-এর মধ্যে 10 রাখতে চাই তবে লিখতে হবে ঃ

```
* p = 10; বা **pp = 10;
```

এখানে, *pp = 10 লিখলে p-এর মান 10 হবে যা memory-এর অজানা অংশকে point করবে।

নিম্নের ptrtoptr.c প্রোগ্রাম লক্ষ্য করুন ঃ

Program **ptrtoptr.c**

```
# include <stdio.h>
# include <conio.h>
```

continue

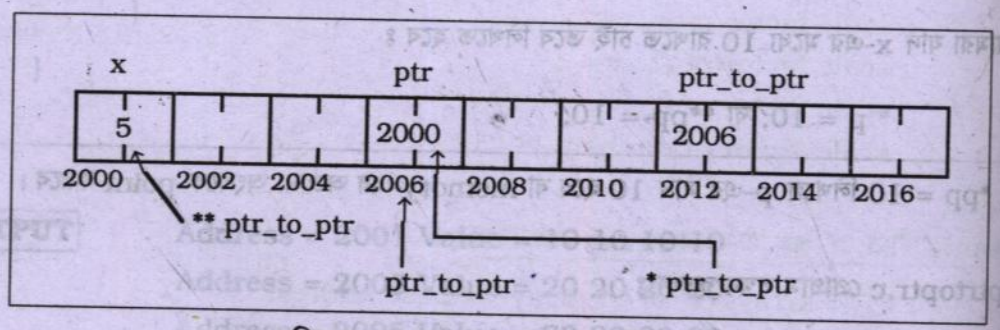
Program **ptrtoptr.c** continue

```
main ( ) {
        int x = 5;
        int * ptr = & x;
        int ** ptr_to_ptr = & ptr;
        printf ("**ptr_to_ptr=%d", **ptr_to_ptr);
        printf ("\n * ptr_to_ptr=%u", *ptr_to_ptr);
        printf ("\n ptr_to_ptr =%u", ptr_to_ptr);
        getch ( );
        }
```

OUTPUT

```
** ptr_to_ptr =5
* ptr_to_ptr= 2000
ptr_to_ptr=2006
```

নিম্নের চিত্র ভালভাবে লক্ষ্য করুন ঃ



চিত্র ঃ Pointer to Pointer বিশ্লেষণ।

Pointer এবং Two Dimensional Arrays :

ponter-এর মাধ্যমে 2-d array-কে ব্যবহার করা যায়। প্রথমে আমরা দেখব 2-d array কিভাবে গঠিত। যেমন ঃ

```
int      array  [5]  [2]  =  {
 ↗        ↗      ↗    ↗
type    name   row  column      { 100, 10 },
                                 { 200, 20 },
                                 { 300, 30 },
                                 { 400, 40 },
                                 { 500, 50 }
                                }
```

উপরোক্ত ডিকলেয়ারেশনকে নিম্নোক্তভাবে বর্ণনা করা যায় ঃ

(ক) 2-d array-এর নাম array

(খ) aray-টির 5 টি row (সারি) আছে।

(গ) প্রতিটি row-তে দুটি element আছে।

(ঘ) প্রতিটি element হল int টাইপ।

	col1 ↓	col2 ↓
row 0 →	100	10
row 1 →	200	20
row 2 →	300	30
row 3 →	400	40
row 4 →	500	50

প্রকৃতপক্ষে 2-d array-এর প্রতিটি row হল এক একটি one Dimensional array। আমরা পূর্বে দেখেছি array [i]-কে (array+i) লেখা যায়। ঠিক এভাবে Two dimensional array-কে নিম্নোক্তভাবে লেখা যায় ঃ

* (array [0] + 0) = array [0] [0]-এর ডাটা (100)
* (array [0] +1) = array [0] [1]-এর ডাটা (10)
* (array [1]+0) = array [1] [1]-এর ডাটা (200)

আবার, Two dimensional array-তে নিম্নোক্তভাবে ডাটা access করা যায় ঃ

* (*(array +0)+0) হল *(array [0]+0)
*(*array+0)+1) হল * (array [0][1])
((array+1)+0) হল * (array [1][0])

নিম্নের প্রোগ্রামটি লক্ষ্য করুন ঃ

Program **ptr2darr.c**

```
# include <stdio.h>
main ( ) {
        int array [5] [2] = {
                              { 100,10 },
                              { 200,20 },
                              { 300,30 },
                              { 400,40 },
                              { 500,50 }
                            };
        int i, j;
        for (i=0; i <=4; ++i)
            {
              printf ("\n");
              for (j=0; j<=1; ++j)
              printf ("array [%d] [%d] = %d", i, j, *(*(array+i)+j));
            }
}
```

OUTPUT

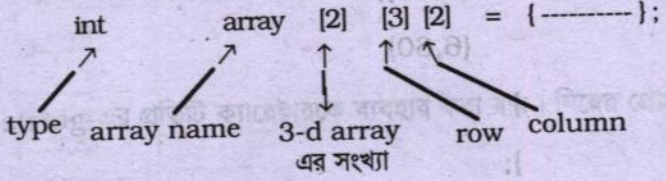
```
array [0] [0] = 100        array [0] [1] = 10
array [1] [0] = 200        array [1] [1] = 20
array [2] [0] = 300        array [2] [1] = 30
array [3] [0] = 400        array [3] [1] = 40
array [4] [0] = 500        array [4] [1] = 50
```

NOTE

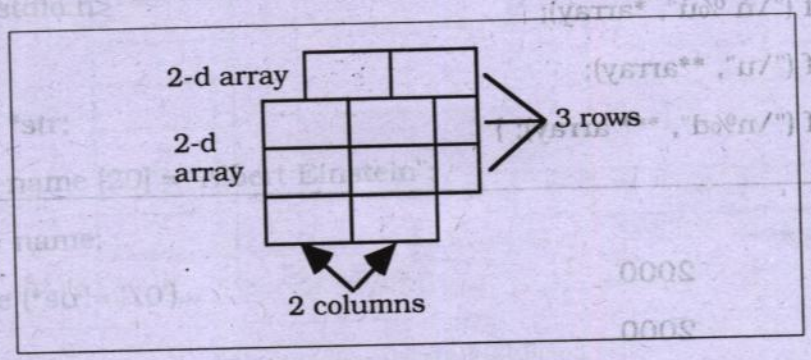
Two dimensional array সম্পর্কে array অধ্যায়ে বিস্তারিত আলোচনা করা হয়েছে।

Pointer এবং Three Dimensional Array :

Three dimensional array মূলত একাধিক 2-d array নিয়ে গঠিত। যেমন ঃ



একে নিম্নোক্ত চিত্রের মাধ্যমে দেখানো যায় ঃ



চিত্র ঃ 3-d array.

Program **ptr3darr.c**

```
# include <stdio.h>
main ( ) {
    int array [2] [3] [2] = {
                            {
                            {1, 10},
                            {2,20},
```

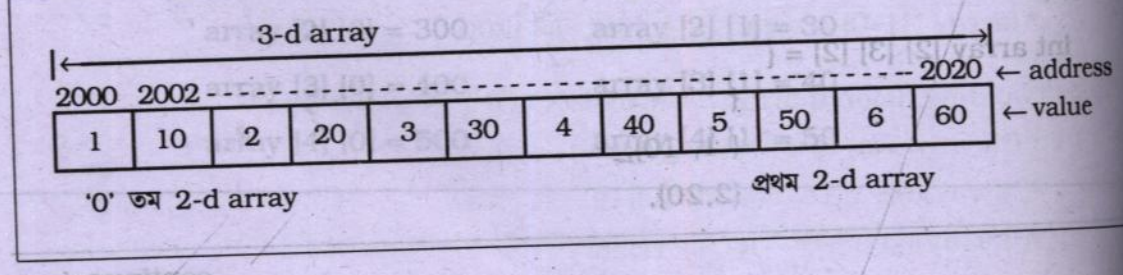
Program **ptr3darr.c** continue

```
            {3,30}
        },
        {
            {4,40},
            {5,50},
            {6,60}
        }
    };
printf ("\n %u", array);
printf ("\n %u", *array);
printf ("\u", **array);
printf ("\n%d", *** array); }
```

OUTPUT

```
2000
2000
2000
1
```

উপরোক্ত প্রোগ্রামের array-কে নিম্নের চিত্রের মাধ্যমে দেখানো হল ঃ



NOTE

address এক এক computer-এ এক এক রকম হতে পারে।

Pointer এবং character string

String হল একাধিক সন্নিহিত ক্যারেক্টারের সমষ্টি। যেমন ঃ 'R', 'E', 'D' হল তিনটি ক্যারেক্টার। কিন্তু "RED" হল string।

Pointer-এর মাধ্যমে string-এর প্রতিটি ক্যারেক্টারকে ব্যবহার করা যায়। নিম্নের প্রোগ্রামটি লক্ষ্য করুন ঃ

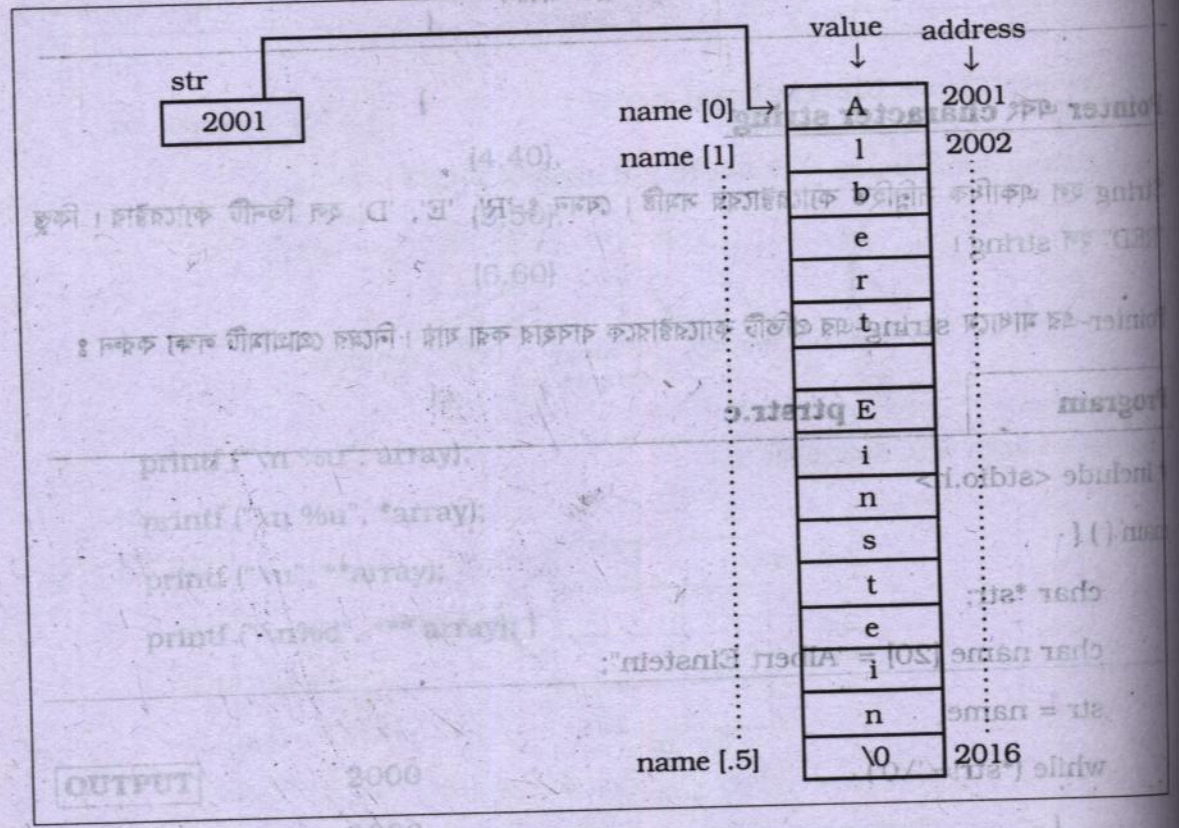
Program ptrstr.c

```
# include <stdio.h>
main ( ) {
    char *str;
    char name [20] = "Albert Einstein";
    str = name;
    while (*str!= '\0')
      {
      printf ("%c", *str);
      str++;
      }
  }
```

NOTE

প্রোগ্রাম output দেখতে হলে ALT-F5 key একসাথে প্রেস করুন।

OUTPUT Albert Einstein



চিত্র ঃ ক্যারেক্টার string-এর pointer।

Array of Pointers (pointer-এর array)

Array of pointer হল pointer ভেরিয়েবল-এর array. অর্থাৎ সাধারণ কোন ভেরিয়েবল-এর যেমন array করা যায়, ঠিক তেমনই pointer ভেরিয়েবলেরও array তৈরি করা যায়। যেমন ঃ

```
char * array [5] = {"red", "green", "blue", "white", "black"};
```

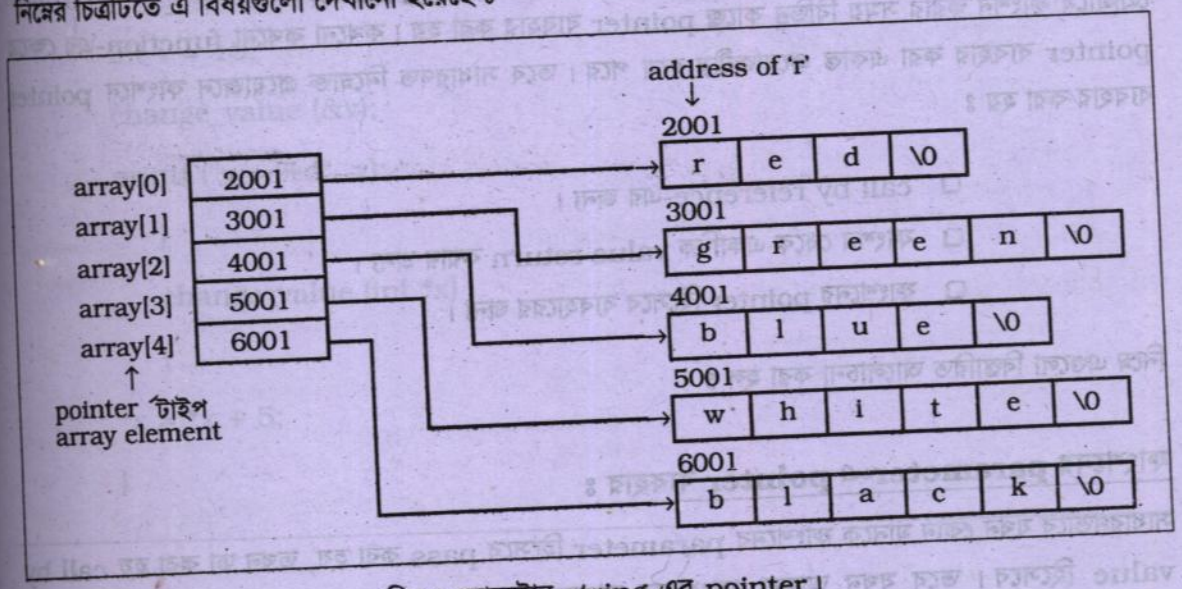
উপরোক্ত pointer-এর array ডিকলেয়ারেশনকে নিম্নোক্তভাবে বর্ণনা করা যায় ঃ

- এখানে array নামে 5 টি array element তৈরি হয়েছে এবং array-এর প্রতিটি element হল char টাইপ পয়েন্টার।

- এই ডিকলেয়ারেশনের ফলে memory-তে স্থান নেয়া হয়েছে এবং পাঁচটি string সেখানে সংরক্ষণ করা হয়েছে। প্রতিটি string-এর শেষে NULL ক্যারেক্টার ('\0') যোগ করা হয়েছে।
- array-এর প্রতিটি element-কে প্রতিটি string-এর প্রথম ক্যারেক্টারের address দ্বারা initialize করা হয়েছে। যেমন ঃ

array [0]-এর মধ্যে red-এর প্রথম ক্যারেকটারের address রাখা হয়েছে।

নিম্নের চিত্রটিতে এ বিষয়গুলো দেখানো হয়েছে ঃ



চিত্র ঃ ক্যারেক্টার string-এর pointer।

Program

```
# include<stdio.h>
# include <conio.h>
main ( ) {
        char * array [3] = {"Alexei", "Maxim", "Gorky"};
        int i;
        for (i=0; i <3; i++)
        printf ("%s", array [i]);
        getch ( );
        }
```

OUTPUT Alexei Maxim Gorky.

Pointer & Function

প্রোগ্রামে ফাংশন করার সময় বিভিন্ন কাজে pointer ব্যবহার করা হয়। কখনো কখনো function-এর ক্ষেত্রে pointer ব্যবহার করা একান্ত প্রয়োজনীয় হয়ে পরে। তবে সাধারণত নিম্নোক্ত প্রয়োজনে ফাংশনে pointer ব্যবহার করা হয় ঃ

- ❑ call by reference-এর জন্য।
- ❑ ফাংশন থেকে একাধিক value return করার জন্য।
- ❑ ফাংশনের pointer হিসেবে ব্যবহারের জন্য।

নিম্নে এগুলো বিস্তারিত আলোচনা করা হল ঃ

ফাংশনের parameter-এ pointer ব্যবহার ঃ

সাধারণভাবে যখন কোন মানকে ফাংশনের parameter হিসেবে pass করা হয়, তখন তা করা হয় call by value হিসেবে। তবে যখন আমরা ফাংশনের মাধ্যমে address pass করি তখন ফাংশনের যে parameter গুলো এই address গ্রহণ করে, তাদেরকে অবশ্যই pointer টাইপ-এর ডিকলেয়ার করতে হবে।

Parameter হিসেবে pointer pass করার জন্য যখন কোন ফাংশনকে call করা হয়, তখন তাকে call by reference বলা হয়।

call by referonce-এর মাধ্যমে কোন ফাংশনকে call করে ঐ ফাংশনের argument হিসেবে যে ভেরিয়েবল-এর address pass করা হয়, ঐ ফাংশনের মাধ্যমে সেই variable-এর মান স্থায়ীভাবে পরিবর্তিত হতে পারে।

নিম্নের change_value ( ) ফাংশনটি main ( )-এর মধ্যে ডিকলেয়ার করা v-এর মানকে স্থায়ীভাবে পরিবর্তন করেছে ঃ

Program **change.c**

```
# include <stdio. h>
main ( ) {
        int v = 10;
        change_value (&v);
        printf ("v = %d", v);
          }
          change_value (int *x)
          {
        *x = *x + 5;
        }
```

OUTPUT V = 15

ফাংশন থেকে Pointer return করা

কোন ফাংশন থেকে যদি একাধিক value return করতে হয় তবে pointer ব্যবহার করা হয়। এছাড়া অন্যান্য কাজেও ফাংশন থেকে pointer return করা হয়।

যে ফাংশন কোন pointer return করে সেই ফাংশনের prototype ডিকলেয়ারেশন এবং definition উভয় ক্ষেত্রেই ফাংশন নামের পূর্বে '*' ব্যবহার করতে হবে। যেমন ঃ

```
return_type * function_name (parameter list);
```

নিম্নের প্রোগ্রামের greater ( ) ফাংশনটি pointer return করেছে ঃ

Program **retptr.c**

```
# include <stdio.h>
int *greater (int *a, int *b);
main ( ) {
        int x = 10, y = 20, * max;
        max = greater (&x, &y);
        printf ("Large Value = %d", * max);
        getch ( ) ;
        return (0);
        }
int *greater (int *a, int *b) {
        if (*b>*a) return b;
        return a;
        }
```

OUTPUT Large Value = 20

প্রোগ্রাম বিশ্লেষণ

- int * greater (int * a, int * b);

ফাংশন নামের পূর্বে * লিখে বুঝানো হয়েছে যে ফাংশনটি pointer raturn করবে।

- int x = 10, y = 20*, *max;

max-কে int টাইপ pointer ডিকলেয়ার করা হয়েছে কারণ max greater ( ) ফাংশন থেকে return করা pointer গ্রহণ করেছে।

- if (*b>*a) return b;
  return a;

b-এর মান যদি a-এর থেকে বড় হয় তবে b-এর address return করবে এবং ফাংশন শে (terminate) হবে নতুবা ফাংশন a-এর address return করবে।

Pointer এবং function

ফাংশনের pointer (pointer to functions)

বিভিন্ন ভেরিয়েবল-এর মত ফাংশনের pointer গঠন করা যায়। প্রোগ্রাম Run করলে ফাংশন memory-তে স্থান করে নেয় এবং memory-তে এর একটি প্রাথমিক address (starting address) থাকে। এই প্রাথমিক address-কে বলা হয় ফাংশনের entry point. pointer to funcfion-এর মধ্যে ফাংশনের এই entry point-এর address সংরক্ষিত হয়।

প্রোগ্রাম চলাকালীন (running) সময়ে ফাংশনকে call করলে, মূলত এই entry point-এর মাধ্যমে ফাংশনকে execute করা হয়।

ফাংশনের pointer ডিকলেয়ার করার নিয়ম নিম্নরূপ ঃ

return data_type (*pointer_to_function) (parameter list);

উদাহরণ ঃ int (*func) (inta a);

এখানে func কে এমন এক ফাংশনের pointer হিসেবে ডিকলেয়ার করা হয়েছে, যার return টাইপ int এবং যার একটি int টাইপ parameter রয়েছে।

ফাংশন pointer-এর মান নির্ধারণ (initializing pointer to function)

ফাংশন pointer-এর মান নির্ধারণের পূর্বে অবশ্যই-এর pototype ডিকলেয়ার করতে হবে। ফাংশন pointer-এর মান নির্ধারণের গঠন নিম্নরূপ ঃ

```
float function (float a); // ফাংশনের prototype ডিকলেয়ারেশন
float (*ptr_func) (float a); // ফাংশন pointer-এর prototype ডিকলেয়ারেশন
ptr_func = function; // function ( )-এর address রাখা হয়েছে ptr_func-এ
```

ptr_func-এর মধ্যে function-এর address রাখার সময় function-এ '( )' ব্যবহার করা হয় না। ফাংশন pointer-এর মাধ্যমে নিম্নোক্তভাবে ফাংশন call করা যায় ঃ

```
Value = ptr_func (a);
```

কোন ফাংশনের pointer to function তৈরি করতে হলে উভয়ের গঠন এক হতে হবে নতুবা কম্পাইলার error message দেখাবে। যেমন ঃ

```
int function (float a); // ফাংশন
float (*ptr_func) (float a); // ফাংশনের pointer ডিকলেয়ার
```

উপরে ফাংশনের return টাইপ int কিন্তু ফাংশন pointer-এর return টাইপ float। তাই function ( )-এর address ptr_func-এর মধ্যে রাখা যাবে না।

ফাংশনের pointer-এর নামকে parentheses ('( )') দ্বারা ঘিরে রাখতে হয়। যেমন ঃ

```
int (*func) (int a);
```

কিন্তু, ফাংশনের pointer-এর নামকে যদি parmtheses দ্বারা ঘিরে না রাখা হয় তবে তার অর্থ অন্যরকম দাড়ায়। যেমন ঃ

```
int * func (int a);
```

উপরে func-কে '( )' দ্বারা ঘিরে রাখা হয়নি। এখন-এর অর্থ দাড়ায় func ( ) এমন এক ফাংশন যে int টাইপ-এর pointer raturn করবে। '( )'-এর precedence '*'-এর অধিক তাই ফাংশনের pointer-এর নামকে '( )' দ্বারা ঘিরে রাখতে হবে। Precedence সম্পর্কে operator এবং expression অধ্যায়ে আলোচনা করা হয়েছে।

Program **callptr.c**

```
# include<stdio.h>
# include<conio.h>
float cube (float a);
float (*ptr) (float a);
main ( ) {
    ptr = cube; /* cube ( )-এর address ptr-এর মধ্যে রাখা হল */
    printf ("Cube = %f", ptr (2.0));
    getch ( );
      }
    float cube (float a);
    {
return a*a*a;
}
```

OUTPUT 8.000000

callptr.c প্রোগ্রামে ফাংশনের pointer-এর মাধ্যমে ফাংশনকে call করা হয়েছে।

ফাংশনের parameter-এ ফাংশন pointer ব্যবহার

যে ফাংশনের parameter হিসেবে ফাংশন pointer ব্যবহার করা তার গঠন নিম্নরূপ ঃ

return type function (return type (*pointer_to_function) (parameter));

যেমন ঃ void show (void (*p) (void));

এখানে show ( ) ফাংশনের parameter হিসেবে ফাংশনের pointer *p ব্যবহার করা হয়েছে।

Program **argfptr.c**

```
#include <stdio.h>
# include <conio.h>
void show (void (*p) (void));
void red (void);
void blue (void);
void unknown (void);
main ( ) {
        void (*ptr) (void);
        char ch;
        printf ("Type R or B :");
        scanf ("%c", &ch);
        switch (ch) {
                case 'R' : { ptr = red; break; }
                case 'B' : { ptr = blue; break;}
                default : { ptr = unknown; break;}
                }
```

continue

Program retptr.c continue

```
    show (ptr);
    getch( ); }
    void show (void (*p) (void))
      {
      p ( );
      }
void red (void)
    {
    printf ("\n RED");
    }
void blue (void)
    {
    printf ("\n BLUE");
    }
void unknown (void)
    {
    printf ("\n Unknown character");
    }
```

OUTPUT

```
Type R or B : R
RED
(re-run)  Type R or B : P
          Unknown character
(re-run)  Type R or B : B
          BLUE
```

NOTE

Structure-এর pointer নিয়ে structure এবং Union অধ্যায়ে আলোচনা করা হয়েছে।

Q & A :

Q. 1. array of pointer-এর সীমাবদ্ধতা কি?

উত্তর ঃ two dimensional array ডিকলেয়ার করার সময় এর মান initialize করা যায় কিংবা scanf ( )-এর মাধ্যমেও এর মধ্যে মান রাখা যায়। কিন্তু, array of pointer ডিকলেয়ার করলে, scanf ( )-এর মাধ্যমে এর মান initialize করা যাবে না। ডিকলেয়ার করার সময় এর মানinttialize করতে হবে। নিম্নের statement গুলো ভুল ঃ

```
char * str [5];
for (i=0; i<5; ++i)
scanf ("%s", str [i]);
```

Q. 2. int i = 5, *p;

```
p =&i;
++p;
*p++;
```

এখানে ++p ও *p++ দ্বারা কি বোঝানো হচ্ছে?

উত্তর ঃ ++p দ্বারা p-এর মধ্যে রক্ষিত address বাড়ানো হচ্ছে এবং * p++ দ্বারা i-এর মান বাড়ানো হচ্ছে।

Q.3. pointer ভেরিয়েবল-এর পূর্বে & ব্যবহার করলে কি হবে?

উত্তর ঃ pointer ভেরিয়েবল এর address পাওয়া যাবে।

Q. 4. নিম্নের প্রোগ্রামের ভুল কি?

```
main ( ) {
        char a [ ] = "I am a conquarer";
        printf ("%c", *a);
        a++;
        printf ("\n%c", *a);
        }
```

উত্তর ঃ a দ্বারা array-এর প্রথম element-এর address-কে বোঝানো হয়। তাই a++ লিখলে কম্পাইলার array-এর base address হারিয়ে ফেলবে। error message : Levalue required.

Q. 5. নিম্নের প্রোগ্রামের Output কি?

```
main ( ) {
        char a [ ] = "HIMALAY";
```

```
char *ch;
ch = &a [2]-2;
while (*ch)
printf ("%c", *ch++);
}
```

OUTPUT HIMALAY

Exercise

1. & এবং * অপারেটরের পার্থক্য কি?
2. pointer কি?
3. Indirection কি?
4. প্রোগ্রামে pointer কেন ব্যবহার করা হয়?
5. নিম্নের প্রোগ্রামের output কি হবে?

```
main ( ) {
    int a [ ] = {1, 2, 3, 4};
    int i;
    for (i=0; i <4; ++i)
    printf ("%d", a+j);
    }
```

6. Pointer to pointer এবং array of pointer-এর মধ্যে পার্থক্য কি?
7. Pointer arithmatic কি?
8. নিম্নের প্রোগ্রামের Output কি হবে?

```
main ( ) {
    int arr [ ] = {1, 3, 5, 7, 9};
    int i;
    for (i=0; i<4; ++i)
```

```
    {
    *(arr+i) = arr [i] + i [arr];
    printf ("%d", * (i+arr));
    }
}
```

9. নিম্নের প্রোগ্রামের output কি হবে?

```
main ( ) {
    int n [ ] = {0,1,2,3};
    int *p [ ] = {n, n+1, n+2, n+3};
    int **ptr;
    ptr=p;
    **ptr++;
    *++*ptr;
    ++**Ptr;
    printf ("\n%d%d%d", ptr-p,*ptr-n,**ptr);
    }
```

10. Three dimensional array-এর pointer কিভাবে তৈরি করতে হয়।

11. pointer to function কি?

12. ফাংশনের parameter হিসেবে pointer to function কিভাবে ব্যবহার করা যায়।

9

String-এর ব্যবহার

- **String**
- **String I/O ফাংশন**
- **String-এর গাণিতিক ব্যবহার**

STRING

string হল কতগুলো ক্যারেক্টারের সমষ্টি। C-তে string নামে কোন ডাটা টাইপ নেই। ক্যারেকটার array এর মাধ্যমে string তৈরি করা হয়। প্রোগ্রামে শব্দ বা বাক্য ব্যবহারের জন্য string ব্যবহার হয়। যেমন ঃ

HANSI CRONYE

MARADONA

String constant

string হল array of character এবং প্রতিটি string-এর শেষে NULL ক্যারেক্টার থাকে। C প্রোগ্রামে string লেখা হয় ডবল কোটেশনের (" ") মধ্যে। যেমন ঃ

" Hello Tareq"

" Programmming in C"

" "

"T"

" 1 2 3"

" # @ % ?"

" Sky"

নিম্নের string-এর গঠন ভুল ঃ

'AMERICAN BEAUTY'

" INVALID

String ভেরিয়েবল ডিকলেয়ারেশন

string ভেরিয়েবল-এর নাম হিসেবে যে কোন সঠিক ভেরিয়েবল নাম ব্যবহার করা যায় এবং একে অবশ্যই array হিসেবে ডিকলেয়ার করতে হয়। string ডিকলেয়ার করার নিয়ম নিম্নরূপ ঃ

char string name [SIZE];

string- টি কয়টি ক্যারেক্টারের সমন্বয়ে গঠিত হবে তা এই SIZE দ্বারা নির্ণয় করা হয়। string ভেরিয়েবল-এ কোন মান রাখলে কম্পাইলার string-এর শেষে নাল কারেক্টার ('\0') যোগ করে। সুতরাং SIZE অবশ্যই string-এ যে কয়টি ক্যারেক্টার থাকবে তার চেয়ে এক বেশি হবে। অর্থাৎ

SIZE = ক্যারেক্টারের সংখ্যা +১

```
char    address [60];
char    line [79];
```

এখানে name হল ক্যারেক্টার টাইপ array যার মধ্যে 40 টা (0-39) ক্যারেকটার রাখা যাবে এবং শেষ ক্যারেক্টারটি হবে নাল ক্যারেক্টার ('\0')।

Initializing Strings (String-এর মান নির্ধারণ)

string ভেরিয়েবল ডিকলেয়ার করার সময় এর মান নির্ধারণ করা যায়। যেমন ঃ

```
char name [20] = " MICHAEL JACKSON";
char city [5] = "DHAKA";
```

এখানে DHAKA শব্দটি মেমোরীতে নিম্নোক্তভাবে সংরক্ষিত হবে ঃ

city [0]	D
city [1]	H
city [2]	A
city [3]	K
city [4]	A
city [5]	\0 ←——NULL ক্যারেক্টার

city-এর মান DHAKA দেয়ার সাথে সাথে কম্পাইলার string-এর শেষে একটি নাল ক্যারেক্টার যোগ করবে। এই নাল ক্যারেক্টারের ASCII মান শূন্য।

String-এর প্রতিটি element-এর মান আলাদাভাবেও initialize করা যায় ঃ

```
char name [8] = {'M', 'A', 'R', 'A', 'D', '0', 'N', 'A', '\0' };
```

এখানে আমাদেরকে NULL ক্যারেক্টার দিয়ে দিতে হবে। আবার ক্যারেকটার array-এর size উল্লেখ না করলে কম্পাইলার ক্যারেকটার হিসেবে করে size নির্ণয় করবে। যেমন ঃ

```
char    string [ ] = "AMERICA";
char    country [ ] = { 'B', 'A', 'N', 'C', 'L', 'A', 'D', 'E', 'S', 'H', '\0' };
```

string ভেরিয়েবল ডিকলেয়ার করার পরও এর মান দেয়া যায়। যেমন ঃ

```
char str [3];
str = "ice";
```

নিম্নোক্তভাবে array element-এর মান নির্ধারণ করা যায় ঃ

```
char str [3];
str [0] = 'i';
str [1] = 'c';
str [2] = 'e';
str [3] = '\0';
```

NUUL ক্যারেকটার ('\0')

String-এর শেষে যদি নাল ক্যারেকটার না থাকে তবে তা মূলত string নয় বড়ং ক্যারেক্টারের সমষ্টি। নাল ক্যারেকটার \ এবং 0 এর সমন্বয়ে গঠিত। এর ASCII মান শূন্য। \0 হল c-এর অন্যতম escape sequence। string-এর শেষে যদি \0 না থাকে তবে ঐ string-এর শেষ কোথায় তা বুঝা যায় না। যেমন ঃ

Program **str1.c**

```
# include <stdio. h>
main ( ) {
    char name [ ] = "TULIP";
    int i = 0;
    while (name [i] ! = '\o')
        {
        printf ("%c", name [i]);
        ++i;
        }
    }
```

string ভেরিয়েবল ডিকলেয়ার করার পরও এর মান দেয়া যায়। যেমন ঃ

```
char str [3];
str = "ice";
```

নিম্নোক্তভাবে array element-এর মান নির্ধারণ করা যায় ঃ

```
char str [3];
str [0] = 'i';
str [1] = 'c';
str [2] = 'e';
str [3] = '\0';
```

NUUL ক্যারেকটার ('\0')

String-এর শেষে যদি নাল ক্যারেকটার না থাকে তবে তা মূলত string নয় বড়ং ক্যারেক্টারের সমষ্টি। নাল ক্যারেকটার \ এবং 0 এর সমন্বয়ে গঠিত। এর ASCII মান শূন্য। \0 হল c-এর অন্যতম escape sequence। string-এর শেষে যদি \0 না থাকে তবে ঐ string-এর শেষ কোথায় তা বুঝা যায় না। যেমন ঃ

Program **str1.c**

```
# include <stdio. h>
main ( ) {
    char name [ ] = "TULIP";
    int i = 0;
    while (name [i] ! = '\o')
        {
        printf ("%c", name [i]);
        ++i;
        }
    }
```

OUTPUT TULIP

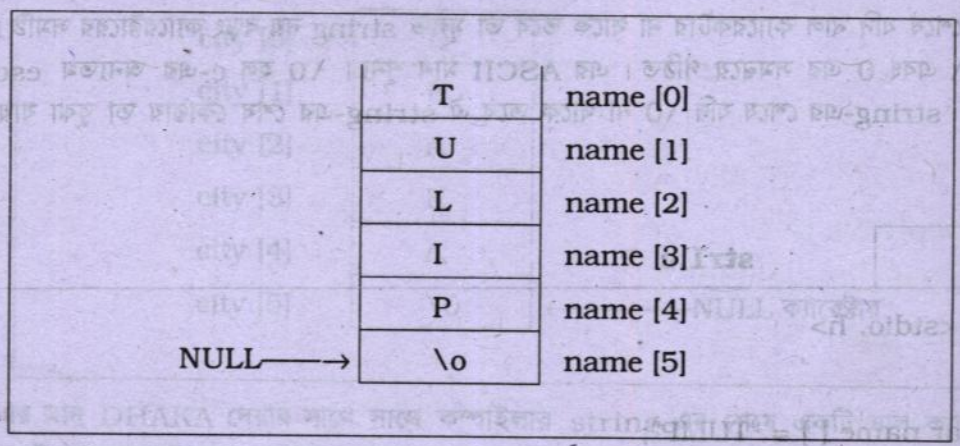
str1.c প্রোগ্রাম বিশ্লেষণ

- char name [ ] = "TULIP";

এখানে name-এর মান " TULIP" দেয়ার সাথে সাথে কম্পাইলার এর শেষে নাল ক্যারেকটার সংযুক্ত করেছে।

- while (name [i]! = '\0')

name-এর প্রতিটি element এর মধ্যে এক একটি ক্যারেক্টার আছে। কোন element-এর মান নাল না হওয়া পর্যন্ত প্রতিটি ক্যারেক্টার প্রিন্ট হবে।



Pointer-এর মাধ্যমে String তৈরি

Array-এর পরিবর্তে pointer ব্যবহার করে string গঠন করা যায়।

যেমন ঃ

```
char    * str = "Programmer can do anything";
char    * name = "Maltucoccus";
```

Array-এর address (প্রথম element-এর) pointer-এ রেখে তার মাধ্যমে string ব্যবহার করা যায়।

যেমন ঃ

Program **str2.c**

```
# include <stdio.h>
main ( ) {
        char str [ ] = "Night is black but day is white";
        char *p;
        p = str;
        while (*p! = '\0' )
          {
          printf ("%c", *p);
          p++;
          }
    }
```

OUTPUT Night is black but day is white

String I/0 ফাংশন

keyboard থেকে Input নেয়া এবং পর্দায় (screen) তার output দেখানোর জন্য C-তে বেশ কয়েকটি string সম্পর্কিত I/0 ফাংশন রয়েছে। যেমন ঃ

OUTPUT ফাংশন ঃ printf ( ), puts ( ), putchar ( )

INPUT ফাংশন ঃ scanf ( ) , gets ( ), getchar ( ), getch ( ), getche ( )

printf ( )/ scanf ( ) ফাংশন ঃ

printf ( ) এবং scanf ( ) ফাংশনে string ব্যবহারের জন্য %s (format specification) ব্যবহৃত হয়।

string ভেরিয়েবল ডিকলেয়ার করার পরও এর মান দেয়া যায়। যেমন ঃ

```
char str [3];
str = "ice";
```

নিম্নোক্তভাবে array element-এর মান নির্ধারণ করা যায় ঃ

```
char str [3];
str [0] = 'i';
str [1] = 'c';
str [2] = 'e';
str [3] = '\0';
```

NUUL ক্যারেকটার ('\0')

String-এর শেষে যদি নাল ক্যারেকটার না থাকে তবে তা মূলত string নয় বড়ং ক্যারেক্টারের সমষ্টি। নাল ক্যারেকটার \ এবং 0 এর সমন্বয়ে গঠিত। এর ASCII মান শূন্য। \0 হল c-এর অন্যতম escape sequence। string-এর শেষে যদি \0 না থাকে তবে ঐ string-এর শেষ কোথায় তা বুঝা যায় না। যেমন ঃ

Program **str1.c**

```
# include <stdio. h>
main ( ) {
        char name [ ] = "TULIP";
        int i = 0;
        while (name [i] ! = '\o')
                {
                printf ("%c", name [i]);
                ++i;
                }
        }
```

OUTPUT TULIP

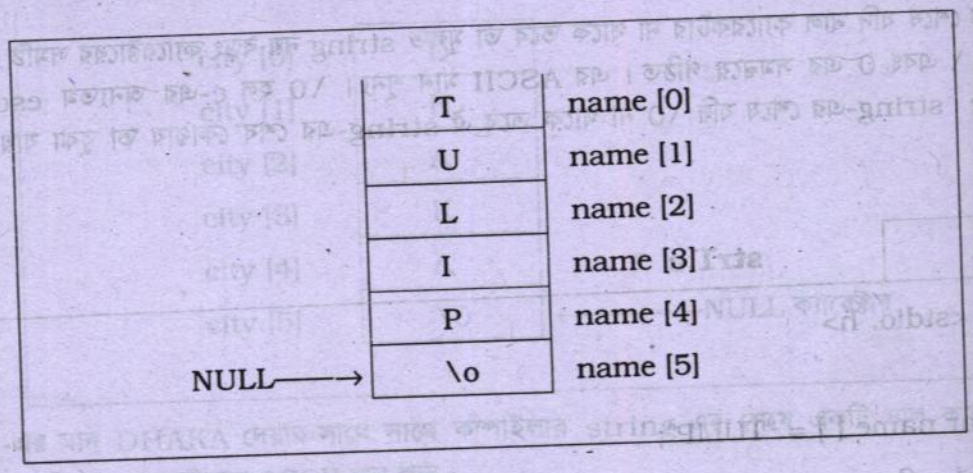
str1.c প্রোগ্রাম বিশ্লেষণ

- char name [ ] = "TULIP";

এখানে name-এর মান " TULIP" দেয়ার সাথে সাথে কম্পাইলার এর শেষে নাল ক্যারেক্টার সংযুক্ত করেছে।

- while (name [i]! = '\0')

name-এর প্রতিটি element এর মধ্যে এক একটি ক্যারেক্টার আছে। কোন element-এর মান নাল না হওয়া পর্যন্ত প্রতিটি ক্যারেক্টার প্রিন্ট হবে।



Pointer-এর মাধ্যমে String তৈরি

Array-এর পরিবর্তে pointer ব্যবহার করে string গঠন করা যায়।

যেমন ঃ

```
char    * str = "Programmer can do anything";
char    * name = "Maltucoccus";
```

Array-এর address (প্রথম element-এর) pointer-এ রেখে তার মাধ্যমে string ব্যবহার করা যায়।

যেমন ঃ

Program **str2.c**

```
# include <stdio.h>
main ( ) {
        char str [ ] = "Night is black but day is white";
        char *p;
        p = str;
        while (*p! = '\0' )
          {
          printf ("%c", *p);
          p++;
          }
    }
```

OUTPUT Night is black but day is white

String I/O ফাংশন

keyboard থেকে Input নেয়া এবং পর্দায় (screen) তার output দেখানোর জন্য C-তে বেশ কয়েকটি string সম্পর্কিত I/O ফাংশন রয়েছে। যেমন ঃ

OUTPUT ফাংশন ঃ printf ( ), puts ( ), putchar ( )

INPUT ফাংশন ঃ scanf ( ) , gets ( ), getchar ( ), getch ( ), getche ( )

printf ( )/ scanf ( ) ফাংশন ঃ

printf ( ) এবং scanf ( ) ফাংশনে string ব্যবহারের জন্য %s (format specification) ব্যবহৃত হয়।

Program prscanf.c

```
# include <stdio. h>
# include<conio.h>
main ( ) {
        char text1 [39], tex2 [39];
        printf ("Enter First and Second name :");
        scanf ("%s %s", text1, text2);
        printf ("\n your name : %s %s", text1, text2);
        getch ( ) ;
        }
```

OUTPUT

```
Enter First and Second name : Tareq Habib
your name : Tareq Habib
```

prscanf.c প্রোগ্রামে scanf ( ) এর মধ্যে tex1 এর পূর্বে & অপারেটর ব্যবহার করা হয়নি। কারণ, index ছাড়া arry হল array এর প্রথম element এর address.

NOTE

scanf ( ) ফাংশন space-এর পরবর্তি ক্যারেকটার উপেক্ষা করে। অর্থাৎ ঃ Tareq Habib এই নামটি scanf ( ) এর মাধ্যমে পড়তে হলে দুটি ভেরিয়েবল নিতে হবে। কারণ Tareq এর পর ফাকা স্থান থাকায় ফাকাস্থানের পরের ক্যারেকটারগুলো scanf ( ) ফাংশন সংরক্ষণ করবে না।

নিম্নের প্রোগ্রামে input না নিয়ে সরাসরি output দেখানো হয়েছে ঃ

Program onlyprin.c

```
# include <stdio. h>
main ( ) {
        char name [ ] = "Bangladesh";
        printf ("%s", name);
        }
```

OUTPUT Bangladesh

puts ( ) /gets ( ) ফাংশন ঃ

puts ( ) এবং gets ( ) ফাংশনের prtotype পাওয়া যায় stdio.h হেডার ফাইলে।

গঠন ঃ char *gets (char *str);

gets ( ) ফাংশনটি keyboard থেকে একটি লাইনের বাক্য ইনপুট নিয়ে str-এর মধ্যে সংরক্ষণ করে।

puts ( ) ফাংশনের prototype নিম্নরূপ ঃ

int puts (const char * str);

puts ( ) ফাংশনটি str-এর মধ্যে রক্ষিত string পর্দায় প্রদর্শন করে।

Program **getsputs.c**

```
# include < stdio. h>
# include<conio.h>
main ( ) {
        char buffer [80];
        puts ("Type your name and press Enter.");
        gets (buffer);
        puts (buffer);
        getch ( ) ;
        }
```

OUTPUT Type your name and press Enter : Michael Anjelo Michael Anjelo

getchar ( ) /putchar ( ) ফাংশন ঃ

getchar ( ) putchar ( ) ফাংশনের prototype পাওয়া যায় stdio.h হেডার ফাইলে। getchar ( ) ফাংশনের prototype নিম্নরূপ ঃ

```
int getchar (Void);
```

getchar ( ) ফাংশনটি keyboard থেকে unsigned character ইনপুট দেয়। putchar ( ) ফাংশনের prototype নিম্নরূপ ঃ

```
int putchar (int ch);
```

এই ফাংশনের মাধ্যমে ch-এর মধ্যে রক্ষিত ক্যারেক্টারটি screen-এ প্রদর্শিত হয়।

Program getchar.c

```
# include <stdio.h>
main ( ) {
   char ch, text [80];
   int i = 0;
   printf ("Type a sentence Enter to end) :\n");
   do
    {
    ch = getchar ( );
    text [i] = ch;
    ++i;
    } while (ch ! = '\n');
   text [i_1] = '\0';
   puts (text);
   }
```

OUTPUT

```
Type a sentence (Enter to and) :
Hello star
Hello star
```

getchar. c প্রোগ্রাম বিশ্লেষণ

- ch = getchar ( ) ;

getchar ( ) ফাংশনটি keyboard থেকে একটি করে ক্যারেকটার ইনপুট নেয় এবং ch-এর মধ্যে সংরক্ষণ করে।

- text [i] = ch;
  ++i;

প্রথমে i এর মান শূন্য। text [0] তে ch এর মান সংরক্ষিত হবে। এরপর i এর মান 1 বৃদ্ধি পাবে ফলে text [1] এ পুনরায় অন্য আর একটি ক্যারেক্টার সংরক্ষিত হবে। এভাবে প্রক্রিয়াটি চলতে থাকবে।

- while (ch! = '\n');

Enter প্রেস না করা পর্যন্ত লুপটি চলতে থাকবে। '\n' হল newline ক্যারেকটার।

নিম্নের প্রোগ্রাম putchar ( ) এর মাধ্যমে ASCII ক্যারেক্টার পিন্ট করা হয়েছে ঃ

Program **putchar.c**

```
# include < stdio.h>
main ( ) {
    int i;
    for (i=0; i <128; ++i)
    putchar (i);
    getch ( ) ;
    }
```

NOTE

getch ( ) ফাংশনের জন্য conio.h হেডার ফাইল যুক্ত করতে হবে।

String ব্যবহারের বিভিন্ন ফাংশন

string.h হেডার ফাইলে বেশ কয়েকটি ফাংশন আছে যাদের মাধ্যমে string নিয়ে বিভিন্ন কাজ করা যায়। এখানে বেশ কয়েকটি ফাংশন নিয়ে আলোচনা করা হল ঃ

strcpy ( ) ফাংশন

strcpy ( ) ফাংশনের prototype নিম্নরূপ ঃ

```
char *strcpy (char *str1, const char *str2) ;
```

strcpy ( ) ফাংশনটি str2-এর মধ্যে রক্ষিত string কপি করে str1-এর মধ্যে সংরক্ষণ করে।

Program strcpy.c

```
# include <string. h>
main ( ) {
    char str1 [20];
    char str2 [20] = "string2";
    strcpy (str1, str 2);
    puts (str1);
    puts (str2);
    }
```

OUTPUT

```
string2
string2
```

strcat ( ) ফাংশন

strcat ( ) ফাংশনের prototype নিম্নরূপ ঃ

```
char *strcat (char *str1, const char *str2);
```

strcat ( ) ফাংশনটি str2-এ রক্ষিত string-কে str1-এর শেষে সংযুক্ত করে। দুটি string যোগ করতে এই ফাংশন ব্যবহৃত হয়।

Program strcat.c

```
# include <string. h>
main ( ) {
        char b [20], s [20];
        gets (b);
        gets (s);
        strcat (b,s);
        puts (b);
        }
```

OUTPUT

```
blue
sky
bluesky
```

strlen ( ) ফাংশন

strlen ( ) ফাংশনের prototype নিম্নরূপ ঃ

```
size_t strlen (const char * str);
```

strlen ( ) ফাংশনটি str-এর মধ্যে রক্ষিত string-এ কতটি ক্যারেক্টার আছে তা নির্ণয় করে। এখানে size_t হল unsigned যা string.h-এ define করা আছে।

Program strlen.c

```
# include <string. h>
main ( ) {
        size_t number;
        char string [80] = "Love bug";
        number = strlen (string);
        printf ("Length of string is : % u", number).
        }
```

OUTPUT Length of string is : 8

strcmp ( ) ফাংশন

strcmp ( ) ফাংশনের prototype নিম্নরূপ ঃ

```
int strcmp (const char * str1, const char * stir2);
```

strcmp ( ) ফাংশনটি str1-এ রক্ষিত string-কে str2-এর string-এর সহিত তুলনা করে এবং int টাইপ মান return করে।

তুলনা	return int টাইপ মান
str1 ও str2 সমান	শূন্য (০)
str2 থেকে str1 ছোট	ঋণাত্মক মান (negative value)
str2 থেকে str1 বড়	শূন্য থেকে বড় (+) ve মান

Program **etrcmp.c**

```
# include <string.h>
main ( ) {
        int i;
        char str1 [80], str2 [80];
        printf ("Enter First string:");
        gets (str1);
        printf ("\n Enter second string:");
        gets (str2);
        printf ("\n");
        i = strcmp (str1, str2);
        if (i==0) printf ("Two string are equal");
        else if (i>0) printf ("First string is greatest");
        else printf ("second string is greater");
        }
```

OUTPUT

```
Enter First string : Tareq
Enter second string : Sardar
Second string is greater
Enter First string : blood is red
Enter second string : blood is red
Two string are equal
```

(re-rum)

ক্যারেক্টার string-এর গাণিতিক ব্যবহার ঃ

নাম্বারকে যেভাবে C প্রোগ্রামে ব্যবহার করা যায়, ঠিক একইভাবে ক্যারেক্টারকেও ব্যবহার করা যায়। string যেহেতু ক্যারেক্টারের সমষ্টি তাই ক্যারেকক্টারের মাধ্যমে string-এর গাণিতিক ব্যবহার সম্ভব।

C প্রোগ্রামে ক্যারেক্টার নিম্নোক্তভাবে ব্যবহৃত হতে পারে ঃ

```
ch> = 'a' && ch < ='z'
```

যখন কোন expression-এ ক্যারেক্টার ব্যবহৃত হয়, কম্পাইলার তখন তাকে automatcally ইনটিজার মানে রূপান্তর করে। এই মানটি হবে ঐ ক্যারেক্টারে ASCII মান।

```
ch = 'a' + 1;
printf ("%d", ch)
```

যেহেতু 'a' এর ASCII মান 97 তাই এখানে ch-এর মান 98 প্রিন্ট হবে।

আবার, ক্যারেক্টার '0' এবং মান 0 এক নয়। যেমন ঃ

```
ch = '0';
i = 0;
```

printf ("ch = %d", ch); ——→ OUTUPT হবে 48

printf ("ch = % d", ch); ——→ OUTPUT হবে 0

ch = '0' লেখার ফলে '0'-এর ASCII মান ch-এ সংরক্ষিত হয়েছে। i = 0, এখানে i-এর মান শূন্য। এখানে 0-এর উভয় পার্শ্বে (' ') ব্যবহার করা হয়নি। তাই কম্পাইলার এই শূন্যকে ক্যারেক্টার হিসেবে ব্যবহার করেনি।

'0'-এর ASCII মান 48। '0' থেকে '9' ক্যারেক্টার গুলোকে integer সংখ্যায় রূপান্তরিত করতে হলে প্রতিটি ক্যারেক্টার থেকে '0' ক্যারেক্টার বিয়োগ দিতে হবে। যেমন ঃ '0'-এর ASCII মান 48 এবং '3'-এর ASCII মান 51। ক্যারেক্টার '3'-কে integer সংখ্যায় রূপান্তরিত করতে হলে নিম্নোক্ত statement লিখতে হবে ঃ

i = '3'— '0'—→ অর্থাৎ i = 51–48 = 3

নিম্নের প্রোগ্রামে string-কে কি পদ্ধতিতে integer- এ পরিণত করা হয় তা দেখানো হল ঃ

Program **strtoin.c**

```
# include <stdio. h>
int integer (char str [ ]);
main ( ) {
      printf ("%d \n", integer ("100") + integer ("200"));
      printf ("%d \n", integer ("10A7"));
      }
int integer (char str [ ] {
    int i, int_val, total = 0;
    for (i = 0; str [i]> = '0' && str [i] < = '9'; ++i)
       {
        int_val = str[i]_'0';
        total = total * 10+int_val;
       }
    return (total);
    }
```

OUTPUT

```
300
10
```

C ল্যাংগুয়েজ (কমপ্লি

stdlib.h হেডার ফা
নিম্নে এদের তালিকা

ফাংশন	p
atoi ( )	in
atof ( )	d
atol ( )	lo
strtol	lo cha

নিম্নের প্রোগ্রামে উপ

Program

```
# include <std
main ( ) {
      printf
      printf
      getch
      }
```

OUTPUT

NOTE

সংখ্যাকে string-
যায়। এটা stalib

stdlib.h হেডার ফাইলে বেশ কয়েকটি ফাংশন আছে যাদের মাধ্যমে string-কে সংখ্যায় রূপান্তর করা যায়। নিম্নে এদের তালিকা দেয়া হল ঃ

ফাংশন	prototype	কাজ
atoi ( )	int atoi (const char * str)	string-কে int ডাটা টাইপ
atof ( )	doubl atof (const chan * str)	string-কে float ডাটা টাইপ
atol ( )	long atol (const ahar * str)	string-কে long int ডাটা টাইপ
strtol	long strtol (const char * start, char **end, int radix)	string-কে long ডাটা টাইপ

নিম্নের প্রোগ্রামে উপরোক্ত ফাংশনগুলো ব্যবহার করে string-কে সংখ্যায় রূপান্তরিত করা হল ঃ

Program **strtonum.c**

```
# include <stdlib.h>
main ( ) {
        printf ("Total is : %d", atoi ("100") + atoi ("100"));
        printf ("\n Total is : %f", atof ("100. 50") + atof ("100.50"));
        getch ( );
        }
```

OUTPUT

```
Total is : 200
Total is : 201.000000
```

NOTE

সংখ্যাকে string-এ রূপান্তর ঃ itoa ( ) ফাংশনের মাধ্যমে integer সংখ্যাকে string-এ রূপান্তর করা যায়। এটা stalib.h হেডার ফাইলে ডিকলেয়ার করা আছে।

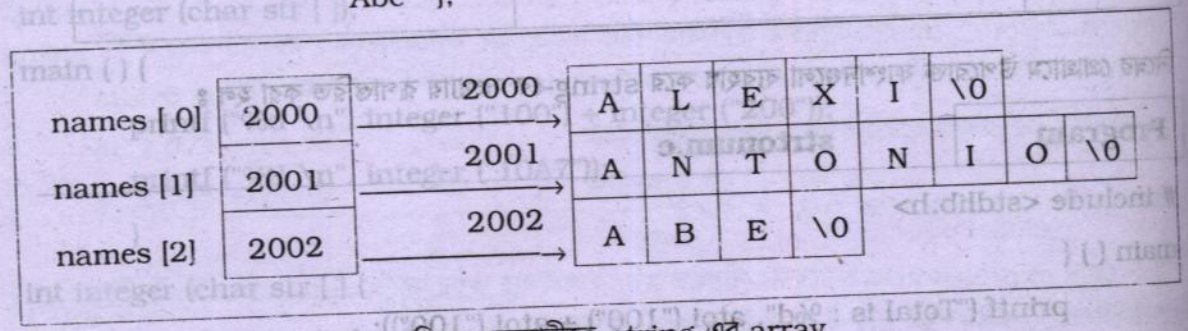
string-এর array (array of strings)

string যেহেতু array-এর এক রূপ, সেহেতু string-এর array হল মূলত array-এর array. যেমন ঃ

```
char names [3] [10] = {   "Padma",
                          "Meghna",
                          "jamuna" };
```

Pointer-এর array-এর মাধ্যমেও string-এর array গঠন করা যায়। যেমন ঃ

```
char * names [ ] = {   "Alexi",
                       "Antonio",
                       "Abe"  };
```



চিত্র ঃ মেমোরীতে string-এর array

Example

নিম্নের প্রোগ্রামে puts ( ) ফাংশনের মধ্যে gets ( ) ফাংশন ব্যবহার করা হয়েছে। প্রোগ্রামটি প্রথমে একটি লাইন input নিবে এবং পরে তা প্রদর্শন করবে।

Program arggets.c

```
# include <stdio. h>
main ( ) {
        char line [80];
        puts (gets (line));
        getch ( ) ;
        }
```

Example

strlwr ( ) ফাংশনের মাধ্যমে এখানে uppercas string-কে lower case-এ এবং strupr ( ) ফাংশনের মাধ্যমে lower case string-কে upper case-এ পরিবর্তন করা হয়েছে।

Program **uplower.c**

```
# include <string.h>
main ( ) {
        char up [3] = "ABC";
        char low [3] = "def";
        puts (strlwr (up));
        puts (strupr (low));
        }
```

Example

strncat ( ) ফাংশনের prototype নিম্নরূপ ঃ

```
char * strncat (char * str1, const char * str2, size_t count);
```

এখানে, str2-এ রক্ষিত string থেকে count সংখ্যক ক্যারেক্টার str1-এ রক্ষিত string-এর শেষে সংযুক্ত করার জন্য strncat ( ) ফাংশন ব্যবহৃত হয়।

Program **strncat.c**

```
# include <string.h>
main ( ) {
    char * prg2 [ ] = "PROGRAM";
    char * prg1 [7];
    int i;
    for (i=1; i <=strlen (prg2); ++i)
```

continue

```
                                                    continue
        {
        strcpy (prg1, " ");
        strncat (prg1, prg2, i);
        puts (prg1);
        }
    }
```

OUTPUT

```
p
PR
PRO
PROG
PROGR
PROGRA
PROGRAM
```

Example

নিম্নের প্রোগ্রামে strrev ( ) ফাংশন ব্যবহার করে string-কে উল্টা করে লেখা হয়েছে ঃ

Program **strrev.c**

```
# include <string. h>
main ( ) {
    char string [ ] = "abcdefghi";
    printf ("NOW string is : %s", string);
    printf ("\n Reverse string is : %s", strrev (string));
     }
```

OUTPUT

```
Now string is : a b c d e f g h i
Reverse string is : i h g f e d c b a
```

Example

কোন string-এ কোন বিশেষ ক্যারেক্টার যোগ করার জন্য strset( ) ও strnset ( ) ফাংশন ব্যবহৃত হয়।

Program **strnsct.c**

```
# include <string.h>
main ( ) {
        char str [ ] = "Grass is green";
        printf ("Fill * : % s", strnset (str, '*', 3));
        printf ("Fill * : %s", strset (str, '*'));
        }
```

OUTPUT

```
Fill * : ***ss is green
Fill * : ***** ** *****
```

Q & A :

1. Q. string এবং array of character-এর মধ্যে পার্থক্য কি?

উঃ string হল কতগুলো ক্যারেক্টারের সমষ্টি যার শেষে NULL ক্যারেক্টার থাকে।

2. Q. "Tristan De chanha" লেখাটি পেতে হলে printf ( ) কে কিভাবে ব্যবহার করতে হবে?

উঃ printf ( ) এর মাধ্যমে (" ") ক্যারেক্টার output পেতে হলে backslash (\) ব্যবহার করতে হবে। যেমন ঃ

```
printf ("\" Tristan Dc Chanha\"");
```

3. Q. strcpy ( ) ফাংশনটি কিভাবে লেখা হয়েছে string.h ফাইলে?

উঃ
```
char * strcpy (char * s1, register const char * s2)
        {
          register char * p = s1;
          while ( *p++ = * s2 ++)
          ;
          return (s1)
        }
```

4. Q. itoa ( ) ফাংশনের কাজ কি?

উঃ itoa ( ) ফাংশনটি integer নাম্বারকে string-এ পরিণত করে।

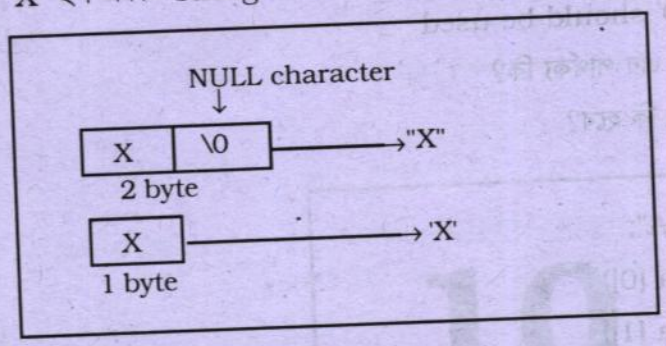
5. Q. strcat ( ) ফাংশনটি নিজে তৈরি করুন।

উঃ
```
char * strcat (char * s1, register const char * s2)
        {
          register char *p = s1;
          while (*p)
          ++p; // go to the end of string
          while (*p++ = * s2++)
          ;
          *p = '\0';
          return (s1);
        }
```

6.9. 'X' এবং "X"-এর পার্থক্য কি?

উঃ 'X' হল একটি ক্যারেক্টার

"X" হল একটি string যার শেষে NULL ক্যারেকটার যোগ হবে।



চিত্র ঃ মেমোরীতে 'X' এবং "X"

Exercise

1. strlen ( ) ফাংশনটি নিজে লিখুন।
2. array of pointer-এর সীমাবদ্ধতা কি?

(pointer অধ্যায়ের শেষে দেখুন)

3. getch ( ) ও getche ( ) ফাংশনের পার্থক্য কি?
4. printf ( ) ফাংশনের (" ")-এর মধ্যে \7 ব্যবহার করলে কি হয়?
5. সংখ্যাকে string- এ রূপান্তর করার একটি ফাংশন লিখুন।
6. কোন string থেকে একটি ক্যারেক্টার delete করার ফাংশন লিখুন।
7. scanf ( ) ফাংশনের সীমাবদ্ধতা কি?
8. putc ( ) ফাংশনের কাজ কি?
৯. stderr কি?
10. Formatted input এবং output ফাংশন এর অর্থ কি?
11. নিম্নের প্রোগ্রামে ভুল কি?

```
char str[3];
str[0] = 'A';
str[1] = 'B';
```

```
str[2] = 'C';
for (i=0; i<3; ++i)
    printf ("%c", str[i]);
      //str[3] = '\0' should be used
```

12. gets ( ) এবং scanf ( ) এর পার্থক্য কি?
13. নিম্নের প্রোগ্রামের output কি হবে?

```
main ( ) {
        char a [2] = "A";
        printf ("%c", a [0]);
        printf ("%d", a [1]);
        printf ("%s", a);
        }
```

14. * p++ ও ++(*p) এর মধ্যে কোন পার্থক্য আছে কি?
15. strcmp ( ) এবং strncmp ( ) ফাংশনের মধ্যে পার্থক্য কি?
16. strchr ( ) ফাংশনের কাজ কি?
17. strstr( ) ফাংশনটি ব্যবহার করে একটি প্রোগ্রাম করুন।

10

Structure এবং Union

- structure
- structure-এর array
- structure-এর pointer
- ফাংশনের argument হিসেবে structure
- ফাংশন থেকে structure return করা
- nested structure
- union
- examples

প্রোগ্রামে বিভিন্ন ধরনের ডাটা টাইপের ভেরিয়েবলকে ব্যবহার করা যায়। যেমন ঃ int, float ইত্যাদি। এগুলো হল built-in ডাটা টাইপ। C প্রোগ্রামে নিজস্ব ইচ্ছানুযায়ী ডাটা টাইপ তৈরি করা যায়। এদেরকে custom data type বলা হয়। custom data type পাঁচ রকমভাবে তৈরি করা যায়। আলোচ্য অধ্যায়ে নিম্নোক্ত দু'ধরনের custom data type নিয়ে আলোচনা করা হয়েছে।

1. structure (conglomerate data type)
2. union

Structure

এক বা একাধিক ভেরিয়েবল এর সনন্বয়ে structure গঠিত এবং প্রতিটি structure-এর নির্দিষ্ট নাম থাকে। sturcture-এর অভ্যন্তরের ভেরিয়েবলগুলো একই বা বিভিন্ন টাইপের হতে পারে।

array-এর সহিত structure-এর মূল পার্থক্য হলে যে array-এর সব element একই ডাটা টাইপের হয় কিন্তু, structure-এর মধ্যে বিভিন্ন ধরনের ডাটা টাইপের ভেরিয়েবল ব্যবহার করা যায়। এবং একারণেই array-এর পরিবর্তে structure ব্যবহার করায় অনেক সুবিধা আছে।

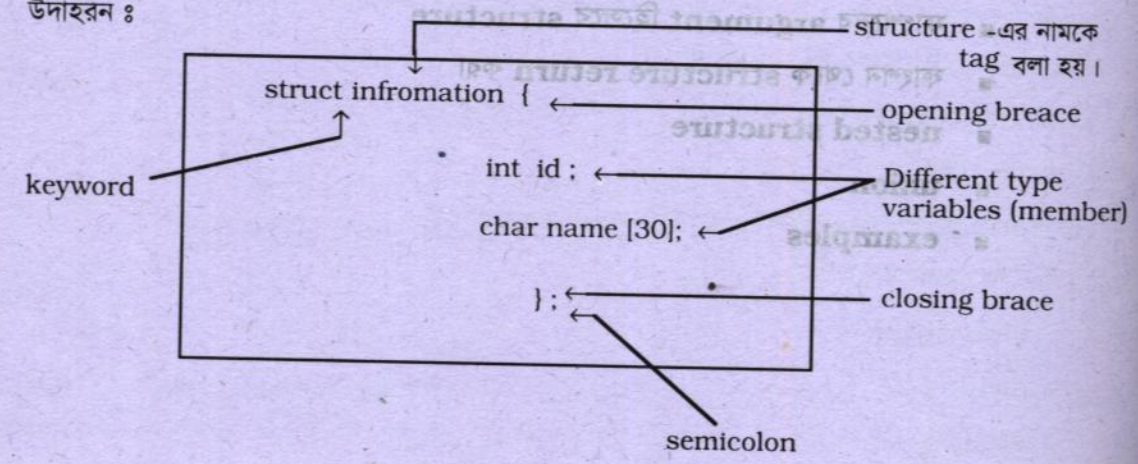
Structure ডিকলেয়ারেশন

নিজস্ব ডাটা টাইপ তৈরির জন্য stucture ডিকলেয়ার করা হয়। একারণেই মনে রাখতে হবে যে stucture ডিকলেয়ার করা এবং কোন structure-এর ভেরিয়েবল ডিকলেয়ার করা দুটি ভিন্ন অবস্থা।

structure ডিকলেয়ার করার নিয়ম নিম্নরূপ ঃ

```
struct structure_tag_name {
                    data_type variables;
                    ......
                    ........
                    };
```

উদাহরন ঃ



প্রতিটি structure-এর নিম্নোক্ত ৫টি অংশ থাকে।

(ক) **struct** : এটি একটি keyword, struct ব্যবহার করে compiler-কে নতুন ডাটা টাইপ সম্পর্কে জ্ঞাত করা হয়।

(খ) **Structure-tag-name** : প্রতিটি structure-এর একটি নাম দিতে হয়। পরে এই নাম উল্লেখ করেই এই structure টাইপ ভেরিয়েবল ডিক্লেয়ার করা হয়।

structure-এর নাম না দিলেও ক্ষতি নেই, তবে সেক্ষেত্রে struture ডিকলেয়ার করার সময় ঐ structure-এর ভেরিয়েবলও ডিকলেয়ার করতে হবে।

(গ) **member elements** : structure-এর মধ্যে ডিকলেয়ার করা প্রতিটি ভেরিয়েবল বা sturcture হল ঐ structure-এর member রা সদস্য। structure-এর প্রতিটি momber-এর মান ভিন্ন হতে হবে। এবং এরা সেমিকোলোন (;) দ্বারা পৃথক থাকবে। যেমন ঃ

```
struct student {

        char name [10];
        char name [10];

};
```

একই নাম (ভুল)

এখানে, দুটি member-এর নাম একই। তাই compiler এখানে error দেখাবে।

(ঘ) **semicolon** : structure ডিকলেয়ারেশন এর শেষে অবশ্যই semicolon ব্যবহার করতে হবে।

(ঙ) **Braces** : structure-এর member বা opening এবং closing braces দ্বারা আবদ্ধ থাকে।

NOTE

Structure ডিকলেয়ার করার সময় member ভেরিয়েবল intialize (মান নির্ধারণ) করা যায় না। নিম্নের ডিকলেয়ারেশনটি ভুল ঃ

```
struct info {

    int i = 0; // Error

};
```

structure টাইপের ভেরিয়েবল ডিকলেয়াররেশন ঃ

এতক্ষণstructure-এর মাধ্যমে নিজস্ব ডাটা টাইপ তৈরি করা হয়েছে। এখন structure-এর মাধ্যমে তৈরিকৃত নিজস্ব ডাটা টাইপের ভেরিয়েবল ডিকলেয়ার করার নিয়ম আলোচনা করা হবে।

বিভিন্নভাবে structure টাইপ এর ভেরিযেবল ঘোষণা করা যায়। যেমন ঃ

উদাহরণ 1 : struct student {

int roll;

char name [30];

};

struct student stud; ←——structure-এর ভেরিয়েবল হল stud

data type variable

এখানে stud-কে student টাইপ structure-এর ভেরিয়েবল ডিকলেয়ার করা হয়েছে।

উদাহরণ 2 : এখানে structure ডিকলেয়ার করার সাথে সাথে structure-এর ভেরেয়েবল ও ডিকলেয়ার করা হয়েছে ঃ

```
struct student {
int roll;
char name [30];
} stud;
```

উদাহরণ 3 : কমা (,) ব্যবহার করে structure ডাটা টাইপের একই সাথে একাধিক ভেরিয়েবল ডিকলেয়ার করা যায়। যেমন ঃ

```
struct student {
int roll;
char name [30];
} stud, stud 1, stud 2;
```

উদাহরণ 4 : tag ছাড়া structure ডিকলেয়ার করা যায়। এক্ষেত্রে structure ডিকলেয়ার করার সময় এই structure টাইপ এর ভেরিয়েবল ডিকলেয়ার করতে হবে। যেমন ঃ

```
struct {
int age;
int height;
} army;
```

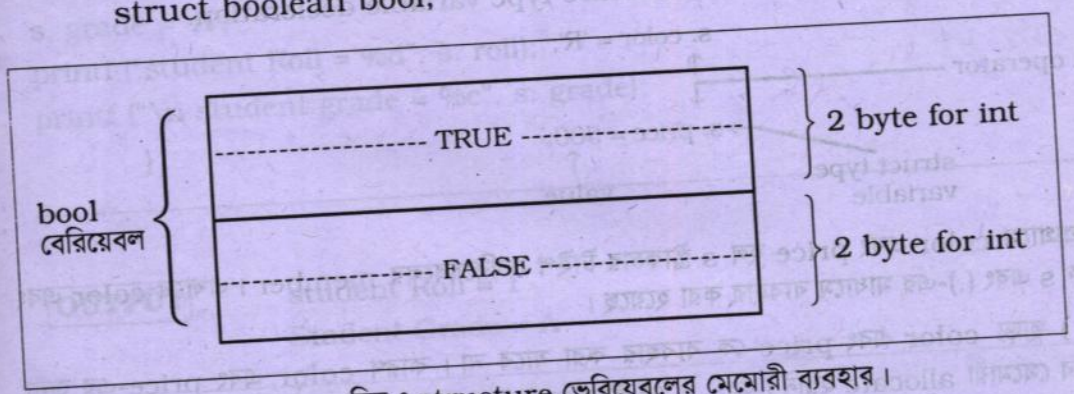
structure এবং মেমোরী

structure হল একধরনের ডাটা টাইপ। তাই structure ডিকলেয়ার করলে কম্পাইলার তার জন্য মেমোরীতে কোন স্থান allocate (নির্দিষ্ট) করে না। যেমন ঃ

```
struct boolean {
        int TRUE;
        Int FALSE;
        };
```

এখানে boolean structure টি মেমোরীতে কোন স্থান দখল করবে না। কিন্তু, boolean টাইপের কোন ভেরিয়েবল ডিকলেয়ার করলে structure-এর প্রতিটি member ভেরিয়েবল এর জন্য কম্পাইলার মেমোরীতে স্থান দখল করবে–

```
struct boolean bool;
```



চিত্র ঃ structure ভেরিয়েবলের মেমোরী ব্যবহার।

NOTE

structure ডিকলেয়ার করার মাধ্যমে নতুন ডাটা টাইপের গঠন তৈরি করা হয়।

structure-এর member ব্যবহার

structure-এর অভ্যন্তরে যে ভেরিয়েবল থাকে তাকে member ভেরিয়েবল বলা হয়। যেমন ঃ name এবং price হল book structure-এর member ভেরিয়েবল।

```
struct book {
        char name [5];
        float price;
        };
```

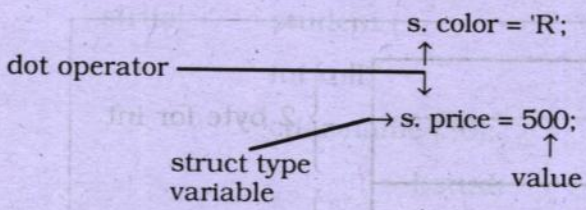
আমরা জানি, array-এর কোন element-কে ব্যবহার করতে হলে subscript ব্যবহার করা হয়। যেমন ঃ arr [0], arr [1] ইত্যাদি।

structure-এর member-কে ব্যবহার করতে হলে dot operator (.) ব্যবহার করা হয়। একে অনেক সময় membership operator'ও বলা হয়। member ভেরিয়েবল ব্যবহার করার গঠন নিম্নরূপ ঃ

struct_type_variable. member variable

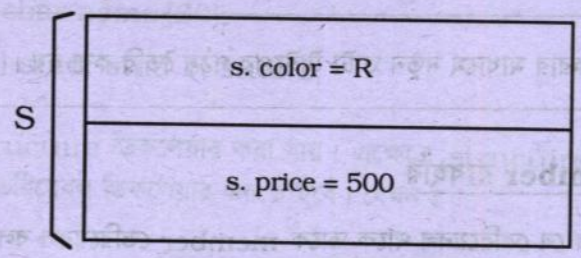
উদাহরণ ঃ

```
struct shirt {
        char color;
        int price;
        };
struct shirt s; ←——structure type variable decleration.
```



উপরের প্রোগ্রামে color এবং price হল s স্ট্রাকচার টাইপ ভেরিয়েবলের member। এখানে color এবং price-কে s এবং (.)-এর মাধ্যমে ব্যবহার করা হয়েছে।

s এবং (.) ছাড়া color এবং price-কে ব্যবহার করা যাবে না। কারণ color এবং price-এর জন্য আলাদাভাবে মেমোরী allocate হয়নি। বরং, s-কে স্ট্রাকচার টাইপ ভেরিয়েবল ডিকলেয়ার করার জন্য color এবং price-এর জন্য memory-তে স্থান allocate (নির্দিষ্ট) হয়েছে।



চিত্র ঃ S. member ব্যবহার

NOTE

DOT OPERATOR : structure-এর member-কে access (ব্যবহার) করার জন্য (.) অপারেটর ব্যবহার হয়। একে membership operator-ও বলা হয়।

নিম্নের প্রোগ্রামের মাধ্যমে structure সম্বন্ধে সাধারণ ধারণা দেয়ার চেষ্টা করা হল ঃ

Program **struct.c**

```
# include <stdio.h>
main ( ) {
        struct student
                {
                int roll;
                char grade;
                };
struct student s;
s. roll = 1;
s. grade = 'A';
printf ("student Roll = %d", s. roll);
printf ("\n student grade = %c", s. grade);
        }
```

OUTPUT

```
student Roll = 1
Stadent Grade = A
```

Keyboard থেকে structure member-এ মান নেয়া

Keyboard থেকে ডাটা ইনপুট নিয়ে structure-এ রাখা যায় এবং সেখান থেকে কম্পিউটারের screen (পর্দা)-এ প্রদর্শন করা যায়। নিম্নের প্রোগ্রামে scanf ( )-এর মাধ্যমে structure member-এ ডাটা ইনপুট নিয়ে printf ( )-এর মাধ্যমে কম্পিউটারে দেখানো হয়েছে।

Program **struct1.c**

```
# include <stdio.h>
struct emp {
            char name [30];
            int salary;
            } person;
```

continue

Program **struct1.c** continue

```
main ( ) {
    printf ("Enter Name :");
    scanf ("%s", person. name); /* array-এর জন্য & ব্যবহার করা ভুল */
    printf ("\n Enter salary:");
    scanf ("%d", & person. salary); // & ব্যবহার করা হয়েছে।
    printf ("\n Name = %s", person. name);
    printf ("\n salary = %d", person. salary);
     }
```

INPUT/OUTPUT

```
Enter : Tareq
Enter salary : 20000
NAME = Tareq
Salary = 20000
```

Structure intialization (ডিকলেয়ারের সময় structure-এর মান নির্ধারণ)

structure টাইপ ভেরিয়েবল ডিকলেয়ার করার সময় এর মান নির্ধারণ করা যায়। যেমন ঃ

```
struct person {
        char name [20];
        int id;
        };
```

```
struct person p = { "Tareq", 1}; // initialization at declaration.
```

এখানে structure টাইপ ভেরিয়েবল ডিকলেয়ার করার সময় এর মান নির্ধারণ করা হয়েছে। এখানে name-এর মধ্যে Tareq এবং id-এর মধ্যে 1 সংরক্ষিত হবে। (p.name = "Tareq"; p.id=1;)

নিম্নোক্তভাবেও structure ভেরিয়েবল-এর মান নির্ধারণ করা যায় ঃ

```
struct person {
        char name {20};
        int id;
        } p = { "Honululu", 2};
```

ফাংশনের মধ্যে structure-এর মান initialize করতে হলে struct-এর পূর্বে static লিখতে হবে। যেমন ঃ

```
struct emp {
        int id;
        char name [30];
        };
main ( )
        {
        static struct emp e = {1, "Julius"};
        ..............
        ..............
        }
```

NOTE

ফাংশনের মধ্যে structure ডিকলেয়ারের সময়-এর মান নির্ধারণের জন্য static kayword ব্যবহার করতে হবে এমন কথা ANSI standard-এ উল্লেখ নেই। তবে কিছু compiler-এ static লেখার প্রয়োজন হয়।

structure-এর মাধ্যমে struature-এর মান নির্ধারণ

দুটি structure ভেরিয়েবল যদি একই structure টাইপের হয় তবে একটি structure ভেরিয়েবল এর মাধ্যমে অপর structure ভেরিয়েবলের মান নির্ধারণ করা যাবে। যেমন ঃ নিম্নের প্রোগ্রামে student1 এবং student2 হল stud টাইপ structure-এর ভেরিয়েবল।

```
student2 = student1;
```

এখানে student1-এর মান student2-এর মধ্যে সংরক্ষিত হবে। পরে student2-এর মান print করা হবে।

Program **stustu.c**

```
# include <stdio.h>
struct stud {
        char name [9];
        };
struct stud student1 = {"Romario"};
struct stud student2;
main ( ) {
        student2 = student1;
        printf ("Name : %s", student2. name);
        }
```

OUTPUT Name : Romario

দুটি structure ভেরিয়েবল-এর তুলনা (comparison)

দুটি ভেরিয়েবলকে তুলনা করার জন্য ralational operator (==, <=, != ইত্যাদি) ব্যবহার করা হয়। যেমন ঃ

```
int a=10, b=10;
if (a==b) printf ("Equal");
```

দুটি structure ভেরিয়েবল যদি একই structure টাইপের হয়, তবে ঐ দুটি structure-এর তুলনা করা যায়, যেমন ঃ

```
struct stud student1, studunt2;
    if (student1== student2)
```

বা, if (student2! = student1)..... ইত্যাদি।

structure ভেরিয়েবলের array

```
struct phone no {
                long int phno;
                char name [30]);
                };
struct phoneno directory 1;
```

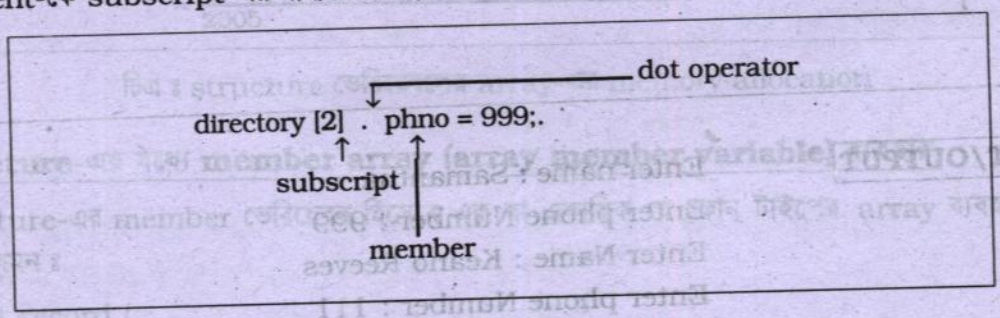
উপরের phoneno structure-এর member হল phno এবং name। directory1 হল phoneno টাইপ structure-এর ভেরিয়েবল। এখানে directory1-এ একজন ব্যক্তির ফোন নাম্বার এবং নাম রাখা যাবে। কিন্তু, আমরা যদি 100 জন ব্যক্তির ফোন নাম্বার রাখতে চাই তবে phoneno stucture-এর 100 টি ভেরিয়েবল ডিকলেয়ার করতে হবে। যেমন ঃ struct phoneno directory1, directory2, directary3;.....dirctory100; কিন্তু, এভাবে অসংখ্য ভেরিয়েবল ডিকলেয়ার করে প্রোগ্রাম করা খুবই দূরহ ব্যাপার। structure ভেরিয়েবল এর array ডিকলেয়ার করে এই সমস্যার সমাধান করা যায়। যেমন ঃ

```
struct phoneno directory [100];
```

এখানে phoneno stucture-এর একটি ভেরিয়েবল হল directory এবং এর মধ্যে 100 টি element রয়েছে। প্রতিটি element এক একটি স্বতন্ত্র phoneno টাইপ ভেরিয়েবল। যেমন ঃ

```
directory [0], directory [1].....directary [99]
```

structure ভেরিয়েবল এর array ডিকলেয়ার করলে, ঐ structure array-এর কোন নির্দিষ্ট element-কে subscript-এর মাধ্যমে ব্যবহার করা হয়। যেমন ঃ



structure-এর array element-এর ডাটা assignment operator (=)-এর মাধ্যমে অপর কোন element-এ রাখা যায়–

```
directory [0] = directory [1];
directory [10]. phno = directory [2]. phno;
```

নিম্নের প্রোগ্রামে structure ভেরিয়েবল এর array ব্যবহার করে input এবং output দেখানো হল ঃ

Program **strarr.c**

```
# include <stdio.h>
      struct record {
                  char name [30];
                  char phone [10];
                  };
struct record rec[2];
main ( ) {
       int i;
  for (i=0; i<2; ++i)
       {
         printf ("Enter Name : ");
         scanf ("%s", rec [i]. name);
         printf ("\n Enter phone Number :");
         scanf ("%s", rec [i]. phone);
       }
   for (i=0; i<2; ++i)
       {
         printf ("\n Name : %s", rec [i]. name);
         printf ("\n phone Number : %s", rec [i]. phone);
       }
  }
```

INPUT/OUTPUT

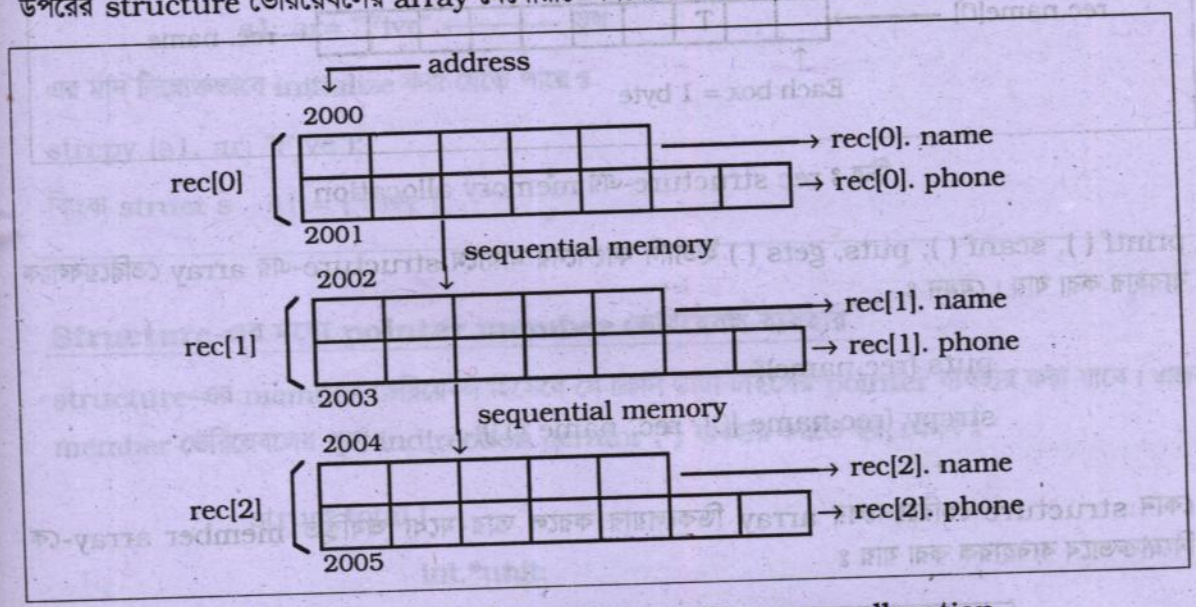
```
Enter name : Samantha
Enter phone Number : 999
Enter Name : Keano Reeves
Enter phone Number : 111
Name : Samantha
Phone Number : 999
Name : Keano Reeves
Phone Number : 111
```

Structure ভেরিয়েবলের array এবং memary' র ব্যবহার

সাধারণ array ভেরিয়েবলের মত structure ভেরিয়েবলের array ও memory-তে একের পর এক বিন্যস্ত থাকে। যেমন ঃ

```
struct record {
        char name [5];
        char phone [7];
        } rec [3];
```

উপরের structure ভেরিয়েবলের array মেমোরীতে নিম্নোক্তভাবে সন্নিবেশিত থাকবে ঃ



চিত্র ঃ structure ভেরিয়েবলের array-এর memory allocation

structure-এর মধ্যে member array (array member variable) ব্যবহার

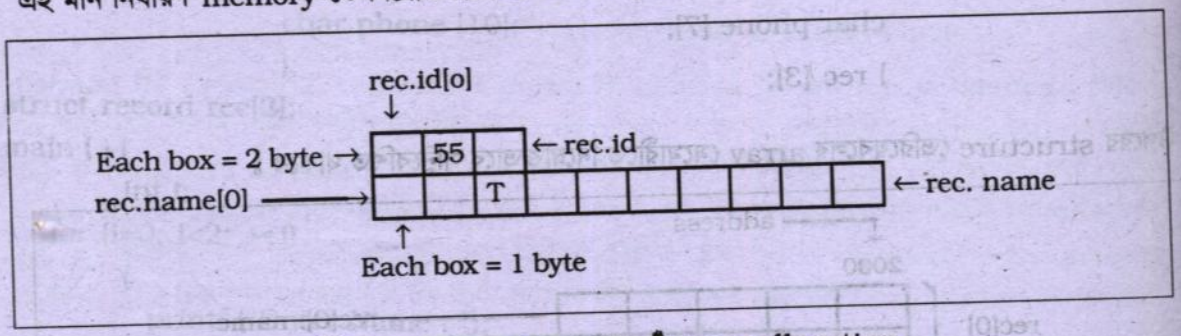
structure-এর member ভেরিয়েবল হিসেবে এক বা একাধিক যে কোন টাইপের array ব্যবহার করা যায়। যেমন ঃ

```
struct record {
        int id[3];  <-- member variable of type array
        char name [10];
        };
struct record rec;
```

structure ভেরিয়েবলের member যদি array হয় তবে তার মান নিম্নোক্তভাবে নির্ধারণ করা যায়। যেমন ঃ

```
rec.id [1] = 55;
rec.name [2] ='T';
```

এই মান নির্ধারণ memory-তে নিম্নোক্তভাবে প্রতিফলত হবে ঃ

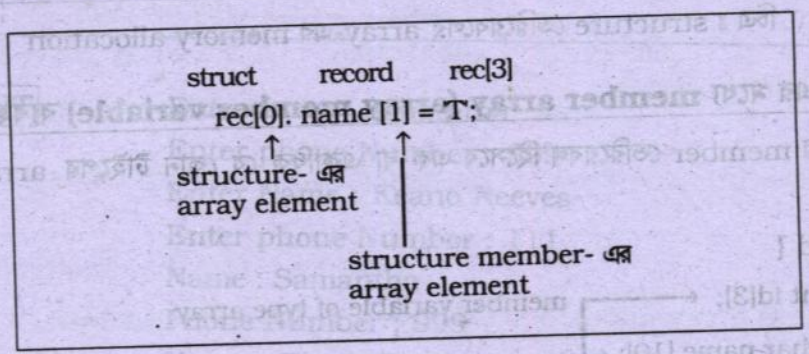


চিত্র ঃ rec structure-এর memory allocation

printf ( ), scanf ( ), puts, gets ( ) ইত্যাদি ফাংশনের মাধ্যমে structure-এর array ভেরিয়েবলকে ব্যবহার করা যায়। যেমন ঃ

```
puts (rec.name);
strcpy (rec.name [0], rec. name [1]);
```

কোন structure ভেরিয়েবলের array ডিকলেয়ার করলে তার মধ্যে অবস্থিত member array-কে নিম্নোক্তভাবে ব্যবহারক করা যায় ঃ



NOTE

কোন structure-এর member ভেরিয়েবল char টাইপ array হলে তার মান নিম্নোক্তভাবে নির্ধারণ করা যায় না–

```
struct s {
        char ar [5];
        } s1;
s1. ar= "Five".←——— ভুল
```

এর মান নিম্নোক্তভাবে initialize করা যেতে পারে ঃ

```
strcpy (s1. ar, "Five");
```

কিংবা `struct s  s1 = {"Five"};`

Structure-এর মধ্যে pointer member ভেরিয়েবল ব্যবহার

structure-এর member ভেরিয়েবল হিসেবে যে কোন ডাটা টাইপের pointer ব্যবহার করা যাবে। এজন্য member ভেরিয়েবলের পূর্বে indirection gerator (*) ব্যবহার করতে হয়, যেমন ঃ

```
struct total {
        int *unit;
        int *price;
        } value;
```

এখানে, unit এবং price হল integer টাইপ pointer ভেরিয়েবল এবং এরা অপর কোন int ভেরিয়েবলের address সংরক্ষণ করতে পারবে। যেমন ঃ

```
int number = 2, cost = 100;
value. unit = & number;
value. price = & cost;
```

নিম্নের প্রোগ্রামের structure-এ char টাইপ pointer ভেরিয়েবল ব্যবহার করা হয়েছে ঃ

Program **strptr.c**

```
#include <stdio.h>
struct test {
        char str [5];
        char * ptr;
        };
struct test t = { "Dhaka", "Khulna" };
main () {
    printf ("str = %s", t. str);
    printf ("\n ptr = %s", t. ptr);
    }
```

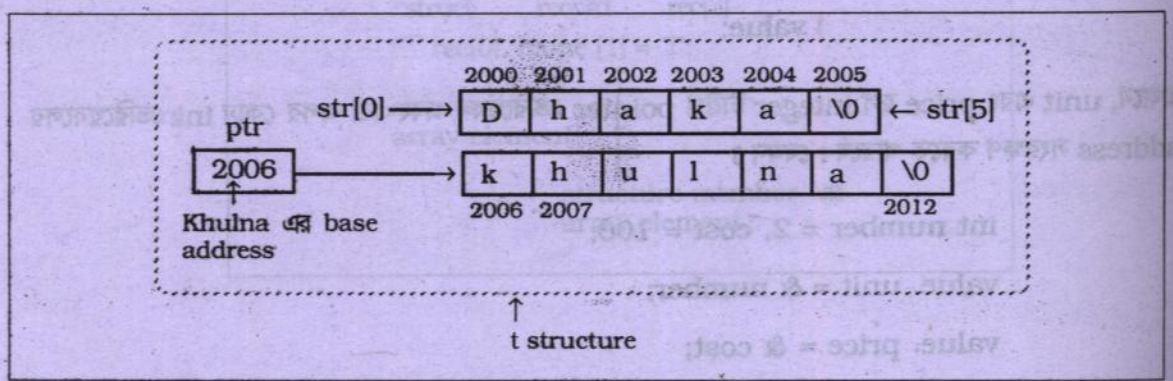
OUTPUT

```
str = Dhaka
ptr = Khulna
```

Strptr.c প্রোগ্রাম বিশ্লেষণ

❑ char * ptr;

ptr-কে char টাইপ pointer ডিকলেয়ার করার ফলে এটি "Khulna"-এর starting address সংরক্ষণ করবে—



চিত্র ঃ structure-এর member pointer ভেরিয়েবল।

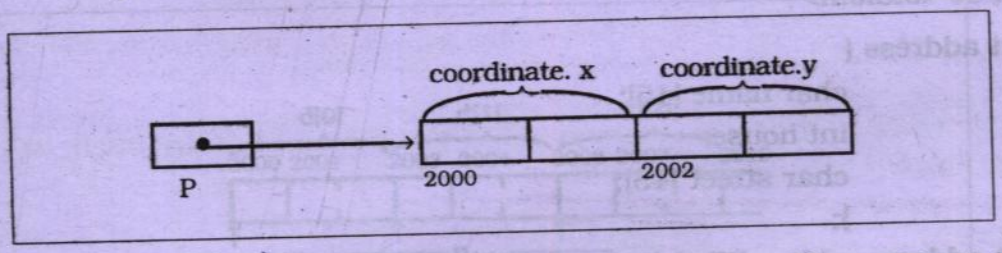
structure-এর pointer তৈরি

Inderection operator (*) ব্যবহার করে structure ভেরিয়েবলের pointer তৈরি করা যায়। structure ভেরিয়েবলের pointer অপর কোন একই টাইপ structure ভেরিয়েবলের address সংরক্ষণ করে। structer-এর pointer সাধারণত link lisk তৈরিতে বা ফাংশনের argument হিসেবে সম্পূর্ণ structure পাঠানোর (pass) জন্য ব্যবহার হয়।

```
struct point {
        int x;
        int y;
        };
struct point *p;
```

এখানে p হল point টাইপ structure-এর pointer। এখন point টাইপ অপর কোন structure-এর address রাখা যাবে p-এর মধ্যে। যেমন ঃ

```
struct point coordinate;
p = & coordinate;
```



চিত্র ঃ structure-এর pointer অপর কোন structure-এর starting address-তে point করে।

আমরা জানি, pointer ভেরিয়েবল যে ভেরিয়েবলের address সংরক্ষণ সেই ভেরিয়েবলের মান pointer ভেরিয়েবল এর মাধ্যমে ব্যবহার করতে হলে pointer ভেরিয়েবলের পূর্বে '*' অপারেট ব্যবহার করতে হয়। যেমন ঃ

```
int x = 10, *p;
p = & x;
printf ("%d", *p) → output হবে 10 যা x-এর মান
printf ("%u", p) → output হবে x-এর address
```

ঠিক একইভাবে, structure-এর pointer-এর মাধ্যমে structure-এর member ভেরিয়েবলকে মান নির্ধারণ করতে হলে pointer structure-এর পূর্বে '*' ব্যবহার করতে হবে। যেমন ঃ

```
(*p). x = 10;
(*p). y = 20;
```

এখানে *p-কে ( )-এর মধ্যে রাখতে হবে কারণ (.)-এর precedence (*)-এর চেয়ে বেশি।

Indirect membership operatar : ' → '-এর সাহায্যে structure-এর pointer-এর মাধ্যমে structure-এর member-কে access করা যায়। যেমন—

```
p→ x = 10;
p→ y = 20;
```

hyphen (-) এবং greater than (>) ব্যবহার করে → তৈরি করা হয়। একে **indirect** membership operator বলা হয়।

Program ptrtostu.c

```
#include <stdio.h>
struct address {
                char name [15];
                int house;
                char street [15];
                };
struct address addr = {"Rafid", 1, "One"};

struct address *p;
main ( ) {
          p = & addr;
          printf ("Name : %s", p→ name);
          printf ("\n House No : %d", p→ house);
          printf ("\nstreet No : %s", p → street);
          }
```

Structure-এর array এবং pointer-এর ব্যবহার

C প্রোগ্রামে structure-এর pointer এবং structure-এর array মিলিয়ে অতি উন্নতমানের প্রোগ্রাম তৈরি করা যায়।

```
struct data {
        char name [10];
        int id;
        };
struct data   d [10];
```

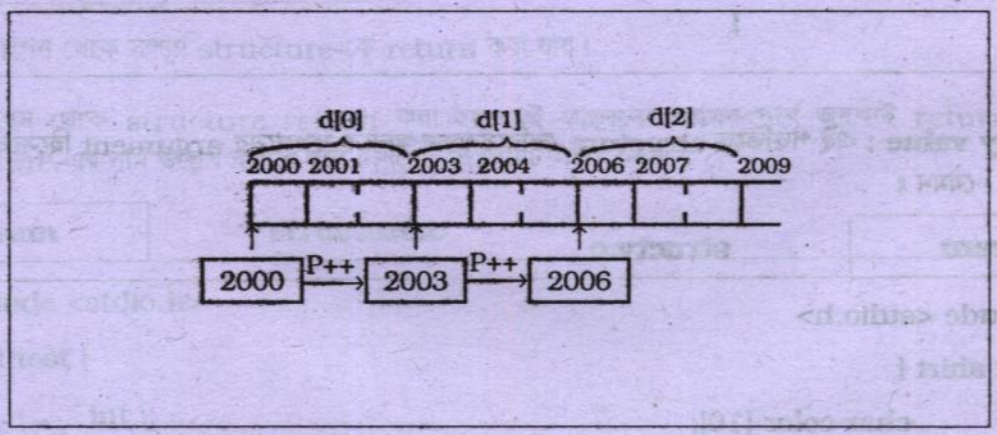
এখানে d [10] হল data টাইপ structure-এর array এবং এর মধ্যে 10টি element রয়েছে।

```
struct data *p;
p = & d [0]; // বা p = d লেখা একই ব্যাপার
```

এখানে p-এর মধ্যে d [10]-এর প্রথম element d [0]-এর address সংরক্ষিত হল।

```
p++;
```

এখানে p পয়েন্টারটি d[1]-কে point করবে।



চিত্র ঃ পয়েন্টারকে বৃদ্ধি করলে এটি পরবর্তি array element-কে point করে।

ফাংশনের argument হিসেবে structure ব্যবহার

অন্যান্য ভেরিয়েবলের মত stoucture-কে ফাংশনের argument হিসেবে ব্যবহার করা যায়। call by valu এবং call by reference উভয় পদ্ধতিতে structure-কে ফাংশনের argument হিসেবে pass করা যায় ঃ

call by reference : এই পদ্ধতিতে structure ভেরিয়েববলের address-কে pass করা হয়। যেমন ঃ

Program **structcr.c**

```
# include <stdio.h>
struct    shirt {
          char    color [10];
          int     price;
          } s = {" Blue", 500};
show (struct shirt *s1);
main ( ) {
        show (& s1);
        }
show (struct shirt *s1) {
        printf ("color : %s \t Price : %d", s1→ color, s1→ price);
                }
```

call by value : এই পদ্ধতিতে structure ভেরিয়েবলের মানকে ফাংশনের argument হিসেবে pass করা হয়। যেমন ঃ

Program **structv.c**

```
# include <stdio.h>
struct shirt {
          char color [10];
          int price;
          } s = {"Blue", 500};
show (struct shirt s1);
main ( ) {
```

Continue

Program structv.c Continue

```
        show (s);
        }
show (struct shirt s1) {
        printf ("cloIor : %s \t price : %d", s1. color, s1. price);
                }
```

NOTE

structure ভেরিয়েবলের কোন member-কে ফাংশনের argument হিসেবে pass করা যায়। যেমন ঃ

```
main ( ) {
        show (s.price)
        }
```

ফাংশন থেকে structure return করা

কোন ফাংশন থেকে সম্পূর্ণ structure-কে return করা যায়।

যে ফাংশন থেকে structure return করা হয় সেই ফাংশনের নামের পূর্বে অবশ্যই return টাইপ structure-এর নাম উল্লেখ করতে হবে। নিম্নের প্রোগ্রামটি লক্ষ্য করুন ঃ

Program structure.c

```
# include <stdio.h>
struct test {
        int i;
        };
main ( ) {
struct test increment (struct test t1);
        static struct test t = { 1};
```

Continue

Program **structure.c** Continue

```
    t = increment (t);
    printf ("Incremented value is : %d", t.i);
    }
struct test increment (struct test t1) {
        t1. i+ = 1;
        return (t1);
        }
```

structure-এর মধ্যে structure (Nested structure) :

structure-এর মধ্যে অপর কোন structure-এর ভেরিয়েবল ডিকলেয়ার করলে তাকে nested structure বলে। যেমন ঃ

```
struct person {
        char mane [20];
        struct  {
                int  salary;
                int  bonus;
        } allowance ;  // nested structare

} employee;
```

কিংবা নিচের মত করেও nested structure গঠন করা যায় ঃ

```
struct sal {
        int salary;
        int bonus;
        };
```

Continue

Continue

```
struct person {
        char name [20];
        //Nested structure
        struct sal allowance;
        } employee;
```

Nested structure-কে access করতে হলে দুবার (.) ব্যবহার করতে হবে। যেমন ঃ

```
employee. allowance. salary = 100;
```

Program **neststru.c**

```
# include <stdio.h>
main ( ) {
    struct address {
            char name [10];
            };
    struct person {
            char Id [5];
            struct address ad;
            };
static struct person p = {"Ao1", "Austin"};
printf ("\n Identification : %s", p.Id);
printf ("\n Name : %s", p. ad. name);
    }
```

Structure-এর মধ্যে structure-এর pointer ব্যবহার

structure-এর মধ্যে ঐ টাইপ বা ভিন্ন টাইপের structure-এর pointer ব্যবহার করা যায়। যেমন ঃ

```
struct    node
          {
          int data;
          node * next;
          } list;
```

এখানে node structure-এর মধ্যে node টাইপ structure-এর pointer ডিকলেয়ার করা হয়েছে যার নাম next। এখানে next হল node টাইপ structure-এর pointer যার মধ্যে অপর কোন node টাইপ structure-এর address সংরক্ষিত থাকবে।

Size of structure (structure-এর আকার byte-এ নির্ধারণ)

কোন structure-এর আয়তন জানতে হলে sizeof ( ) অপারেটর ব্যবহার করা যেতে পারে। যেমন ঃ

```
sizeof (t);
sizeof (emp);
```

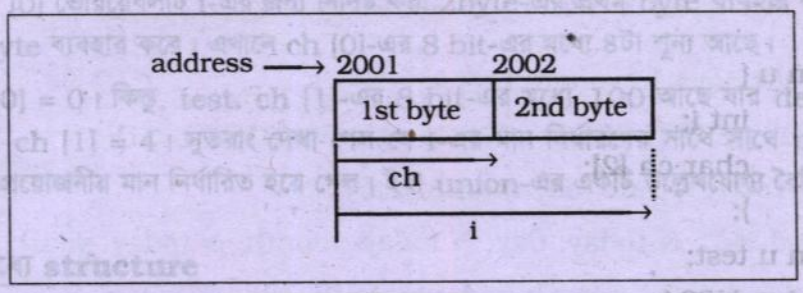
এখানে t কিংবা emp কত byte স্থান মেমোরীতে দখল করবে তা sizeof ( ) দ্বারা জানা যাবে।

UNIONS

Union-এর গঠন structure-এর মত তবে structure-এর সাথে union-এর মূল পার্থক্য হল যে union-এর member ভেরিয়েবল একই সাথে একটির বেশি ব্যবহার করা যায় না। কারণ union-এর member ভেরিয়েবলগুলো মেমোরীর একইস্থান share করে। যেমন ঃ

```
union test {
          int i;
          char ch;
          } mem;
```

এখানে mem union-এর member ভেরিয়েবল i-এর জন্য 2 byte স্থান মেমোরীতে allocate হবে এবং ch তার প্রয়োজনীয় 1 byte i-এর 2byte থেকে share করবে।



চিত্র ঃ union-এর member ভেরিয়েবলের memory share করা।

কোন Union-এর ভেরিয়েবল ডিকলেয়ার করা হলে ঐ union ভেরিয়েবলের এর যে member-এর জন্য সর্বোচ্চ memory byte-এর প্রয়োজন কম্পাইলার শুধু ততটুক memory allocate করে এবং অন্যান্য member ভেরিয়েবল ঐ একই memory এলাকা share করে ব্যবহার করে।

Union ডিকলেয়ার করা

Union ডিকলেয়ার করার মাধ্যমে নতুন ডাটা টাইপ গঠন করা হয়। union ডিকলেয়ার করার নিয়ম নিম্নরূপ ঃ

```
union tag_name
        {
    data type variable,
    data type variable;
      .....
    };
```

union একটি kay word, union declare করতে হলে অবশ্যই union keyword ব্যবহার করতে হবে। tag_name হল union-এর মান। tag_name-এর পর {.....}-এর মধ্যে ঘোষিত ভেরিয়েবলগুলো হল member variable। union ডিকলেয়ারেশনের শেষে অবশ্য semicolon (;) ব্যবহার করতে হবে।

Union-এর ভেরিয়েবল ঘোষণা

union-এর ভেরিয়েবল ডিকলেয়ার করতে হলে প্রথমেই union লিখে তার পর tag_name এবং সর্বশেষে ভেরিয়েবল এর নাম লিখতে হবে।

```
union tag_name instance;
```

Program union.c

```
# include <stdio.h>
main ( ) {
        union u {
                int i;
                char ch [2];
                };
        union u test;
        test. i = 1`024;
        printf ("test.i=%d", test. i);
        printf ("\n test. ch [0] =%d", test. ch [0]);
        printf ("\n test. ch [1] = %d", test. ch [1]);
        }
```

OUTPUT

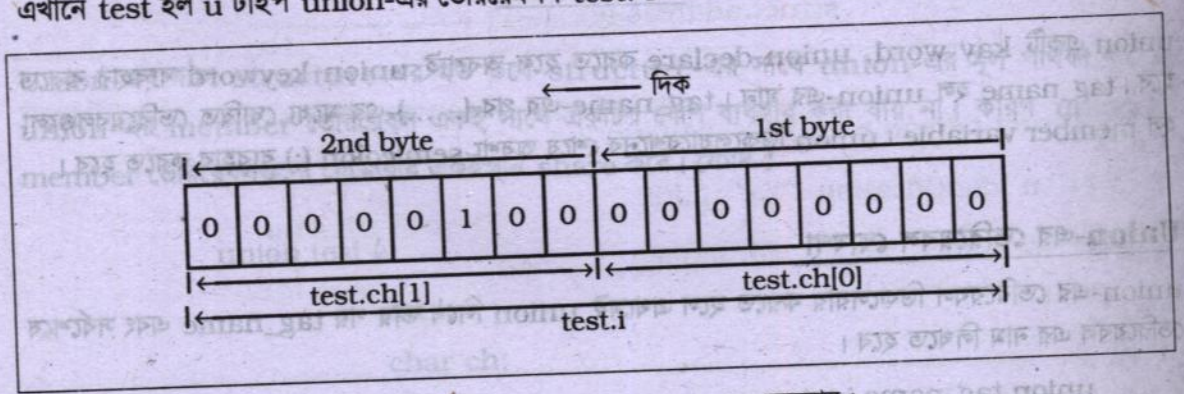
```
test. i = 1024
test. ch [0] = 0
test. ch [1] = 4
```

Union.c প্রোগ্রাম বিশ্লেষণ ঃ

- union u test;

  test. i = 1024;

এখানে test হল u টাইপ union-এর ভেরিয়েবল। test. i-এর মাধ্যমে i-এর মান নির্ধারণ করা হয়েছে।



চিত্র ঃ test union-এর memory ব্যবহার।

আমরা জানি, union-এর member গুলো memory share করে ব্যবহার করে। এখানে, test. i = 1024 লেখার ফলে 1824-এর binary মান (10000000000) i-এর memory-তে সংরক্ষিত হয়েছে। আবার test. ch [0] ভেরিয়েবলটি i-এর জন্য নির্দিষ্ট করা 2byte-এর প্রথম byte ব্যবহার করে এবং test. ch [1] দ্বিতীয় byte ব্যবহার করে। এখানে ch [0]-এর 8 bit-এর মধ্যে 8টা শূন্য আছে।

তাই test. ch [0] = 0। কিন্তু, test. ch [1]-এর 8 bit-এর মধ্যে 100 আছে যার decimel মান হল 4. সুতরাং test. ch [1] = 4। সুতরাং দেখা গেল যে i-এর মান নির্ধারণের সাথে সাথে ch [0] এবং ch [1]-এর জন্যও অপ্রয়োজনীয় মান নির্ধারিত হয়ে গেল। ইহা union-এর একটি উল্লেখযোগ্য বৈশিষ্ট্য।

Union-এর মধ্যে structure

union-এর মধ্যে structure লেখা যায়। যেমন ঃ

```
struct   p { int i; };
struct   q { int j; };
union    r {
         struct p  x
         struct q  y
         }z;
z. x. i = 10;
```

Example

নিম্নের প্রোগ্রামটি input হিসেবে আজকের তারিখ দিলে, কালকের তারিখ output হিসেবে দেখাবে ঃ

Program todtom.c

```
# include <stdio.h>
struct date {
        int month;
        int day;
        int year;
        };
```

continue

continue

```
struct date today, tomorrow;
main ( ) {
        static int days_in_month [12] =
            { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
printf ("Type Today's Date (day month year) : ");
scanf ("%d %d %d", & today. day, & today. month, & today. year);
if (today. day! = days_in_month [today. month–1])
        {
        tomorrow. day = today. day +1;
        tomorrow. month = today. month;
        tomorrow. year = today. year;
        }
else if (today. month = =12)
        {
        tomorrow. day = 1;
        tomorrow. month = 1;
        tomorrow. year = today. year +1;
        }
else
        {
        tomorrow. day = 1;
        tomorrow. month = today. month +1;
        tomorrow. year = taday. year;
        }
printf ("\n The next date is %d/%d/%d", tomorrow. day, tomorrow. month,
                                tomorrow. year);
        }
```

Example

. নিম্নের প্রোগ্রামে nested structure-এর ব্যবহার দেখানো হল ঃ

Program **nesstruc.c**

```
# include <stdio.h>
main ( ) {
      struct x {
              char ch [10];
              char * str;
              };
      struct  y {
              char * ch1;
              struct x xy;
              };
static struct y z = {"Biplob", "Tamanna", "Rafid"}; printf ("%s %s", z.ch1, z.xy.
str);
      }
```

Example

নিম্নের প্রোগ্রামে union ব্যবহার করে ROM-BIOS ফাংশনের মাধ্যমে কম্পিউটারের memory size বের করার পদ্ধতি দেখানো হল ঃ

Program **unionsz.c**

```
# include <stdio.h>
main ( ) {
     struct WORDREGS
      {
      unsigned int ax, bx, cx, dx, si, di, eflag, flags;
      };
```

continue

continue

```
struct BYTEREGS
    {
    unsigned char al, ah, bl, bh, cl, ch, dl, dh;
    };
union REGS
    {
    struct WORDREGS x;
    struct BYTEREGS h;
    };
union REGS inregs, outregs;
int86 (18, & inregs, & outregs);
printf ("Totat PC Memory = %d", outregs. x. ax);
}
```

OUTPUT Total PC Memory = 640

Q & A :

Q. 1. structure এবং union-এর মূল পার্থক্য কি?

উত্তর ঃ structure-এর member ভেরিয়েবলের জন্য আলাদা আলাদা মেমোরী allocate হয় কিন্তু, union-এর member ভেরিয়েবলগুলো একই memory area share করে।

Q. 2. structure এবং array-এর পার্থক্য কি?

উত্তর ঃ array-এর element-গুলো একই টাইপের হয় কিন্তু, structure-এর member ভেরিয়েবলগুলো বিভিন্ন ডাটা টাইপের হতে পারে।

Q.3.
```
struct show {
        char name [10];
        } s;
main ( ) {
        s. name = "Banana";
        }
```

উপরের প্রোগ্রাম ভুল কি?

উত্তর ঃ কোন structure-এর member ভেরিয়েবল char টাইপ array হলে তার মান নির্ধারণ করার এই পদ্ধতিটি ভুল। এখানে, s. name = "Banana" না লিখে strcpy (s. name, "Banana") লেখা যেতে পারে কিংবা structure-এর ভেরিয়েবল s ভিকলেয়ার করার সময় এর মান নির্ধারণ করা যেতে পারে।

Q.4.
```
struct a {
        int i ;
        };
struct b {
        struct a aa;
        };
struct b bb;
bb. aa. i = 10;
```

এখানে দুটি (.) অপারেটর কেন ব্যবহার করা হয়েছে?

উত্তর ঃ aa হল nested structure-এর variable। তাই i-কে access করতে হলে দুটি dot operator ব্যবহার করতে হবে।

Exercise

1. structure ডিকলেয়ার এবং structure-এর ভেরিয়েবল ডিকলেয়ার করার মধ্যে পার্থক্য কি?
2. structure ডিকলেয়ার করলে তার জন্য memory allocat হয় কি?
3. এমন একটি প্রোগ্রাম লিখুন যা বর্তমান সময় input নিয়ে এক সেকেণ্ড পরের সময় output দেখাবে।
4. Hardware সম্পর্কিত প্রোগ্রামিং-এর ক্ষেত্রে union ব্যবহার এর সুবিধা কি?
5.
```
Union {
        int i;
        float f;
        char ch;
        } u;
```
এই union ভেরিয়েবল u ডিকলেয়ার করার ফলে u-এর জন্য মেমোরীতে সর্বমোট কত byte allocate হবে?

6. structure tag এবং এর মধ্যে পার্থক্য কি?
7. নিম্নের প্রোগ্রামাংশে ভুল কোথায়?
```
struct {
        char ch;
        int i;
        } var = 'A', 100;
```
8. নিম্নের প্রোগ্রামাংশে ভুল কি?
```
union p {
        int i;
        char ch;
        } u = {100, 'A' };
```
সূত্র ঃ union-এর ক্ষেত্রে কেবলমাত্র একসাথে একটি ভেরিয়েবল নিয়ে কাজ করা যায়।

9. structure-এর pointer ব্যবহার করে একটি প্রোগ্রাম লিখুন।
10. union-এর array ভেরিয়েবল ডিকলেয়ার করা যায় কি?
11. int86 (-) ফাংশনের কাজ কি?
12. sizeof ( ) অপারেটরের মধ্যে structure ভেরিয়েবল ব্যবহার করার উপায় কি?
13. dot (.) এবং indirect memberslip (→) অপারেটরের পার্থক্য কি?

11

User defined data type

- **enumeration**
- **typedef**
- **bit-field**

C-তে বিভিন্ন ধরনের ডাটা টাইপ রয়েছে। যেমন ঃ int, char, float ইত্যাদি। এগুলো হল built-in data type। প্রোগ্রামের সুবিধার জন্য নিজস্ব ইচ্ছানুযায়ী ডাটা টাইপ তৈরি করা যায়। এদেরকে custom data type বলা হয়। নিম্নোক্ত পাঁচভাবে custom data type তৈরি করা যায় ঃ

1. structure
2. union
3. enumeration
4. typedef
5. bit-field

structure এবং union নিয়ে অধ্যায় ১০-এ আলোচনা করা হয়েছে। এই অধ্যায়ে enumeration, typedef এবং bit-field নিয়ে আলোচনা করা হবে।

enumeration

enumeration-এর মাধ্যমে নিজস্ব ডাটা টাইপ তৈরি করা যায়। enumeration হল কতগুলি integer constant-এর সমষ্টি এবং এদেরকে enumeration-এর member বলা হয়। enumeration তৈরির সাথে সাথে এর member গুলোর মান compiler নির্ধারণ করে দেয়।

enumeration-এর ডাটা টাইপ ডিকলেয়ার

enumeration-এর ডাটা টাইপ গঠনের নিয়ম নিম্নরূপ ঃ

enum identifier { enumeration member (s) };

এখানে enum হল keyward, enumeration ডাটা টাইপ তৈরি করতে হলে অবশ্যই enum ব্যবহার করতে হবে।

identifier হল enumeration এর tab_name। পরবর্তিতে enumeration এর ভেরিয়েবল তৈরিতে এই identifier ব্যবহৃত হয়। enumeration member লেখার সময় কোন ডাটা টাইপ উল্লেখ করতে হয় না। compiler এদেরকে automctically integer constant হিসেবে ধরে নেয়। enumeration member গুলো {.....} দ্বারা আবদ্ধ থাকে।

enumeration ডিকলেয়ারের শেষে অবশ্যই semicolon ব্যবহার করতে হবে।

উদাহরণ ঃ enum color { white, red, black };

enum booleam { TRUE, FALSE };

key word — indentifier — members

enumeration ডাটা টাইপের ভেরিয়েবল ডিকলেয়ার করা

enumeration ডাটা টাইপ ডিকলেয়ার করার পর এই ডাটা টাইপের ভেরিয়েবল ডিকলেয়ার করা যায়। enumeration ডাটা টাইপের ভেরিয়েবল ডিকলেয়ার করার গঠন নিম্নরূপ ঃ

enum identifier variable^1, variable^2.....variable^n;

এখানে identifier হল enumeration tag_name এবং variable হল variable-এর নাম।

উদাহরণ ঃ enum money {ONE, TWO, FIVE, TEN}; //enumeration declaration

enum money take, paisa; // variable declaration

key word — tag-name — variables

enumeration ডাটা টাইপ ডিকলেয়ার করার সময় এর ভেরিয়েবল ডিকলেয়ার করা যায়। যেমন ঃ

enum news { NORTH, EAST, WEST, SOUTH } TIMES, NEWSWEEK;

enumerated ভেরিয়েবলের member-এর মান

enumerated data টাইপের ভেরিয়েবল ডিকলেয়ার করার সাথে সাথে কম্পাইলার এর member-এর জন্য মান নির্ধারণ করে দেয়। এই member গুলোর মান 0 থেকে শুরু হয় এবং পরের member গুলোর মান এক করে বৃদ্ধি পেতে থাকে। যেমন ঃ

enum subject { BANGLA, ENGLISH, MATH}; → member

enum subject sub; → variable

এখানে BANGLA এর মান '0' ENGLISH এর মান 1 এবং MATH এর মান 2। কম্পাইলার এই মান নির্ধারণ করেছে।

এখানে মনে রাখতে হবে যে BANGLA, ENGLISH.....ইত্যাদি লেখার সময় কোন ডাটা টাইপ উল্লেখ করতে হয় না।

enumeration-এর member-এর মান আমরা নিজে থেকেও দিতে পারি। যেমন ঃ

```
enum subject { B=1, F,M };
```

এখানে, B এর মান 1 দেয়া হয়েছে তাই E=2 এবং M=3 হবে।

```
enum alphabet { A=-1, B,C, D = 100, E,F };
```

এখনে A = -1, সুতরাং B=0 এবং C=1 আবার B=100 দেয়া হয়েছে। তাই E = 101 এবং F = 102 হবে।

NOTE

enumeration member-এর প্রথমটির মান পরিবর্তন না করে পরবর্তি কোন member-এর মান নির্ধারণ করা যাবে না। যেমন ঃ

```
                    ↓
enum alphabet { A,B, C = 100, D,E }; ←————ভুল।
enum alphabet { A = 0, B, C =100,D,E } ←————সঠিক।
                ↑          ↑
```

Program **enum1.c**

```
# include <staio.h>
   main ( ) {
      enum num { ZERO, ONE, TWO, THREE };
      printf ("ZERO = %d ONE = %d", ZERO, ONE);
      printf ("\n TWO = %d THREE = %d", TWO, THREE);
        }
```

OUTPUT

```
ZERO = 0    ONE =1
TWO  = 2    THREE =3
```

enum1.c প্রোগ্রামে enumeration-এর কোন ভেরিয়েবল ডিকলেয়ার না করে member-গুলোকে ব্যবহার করা হয়েছে।