C-Language Notes

Part1

Basics of programming: Program, Code, Computer Language, Problem, Programming, C compiler, Software, Hardware, Keyboard, Monitor, RAM, Hard disk, CPU, Peripherals, Console, Standard I/O, Flow of data, Algorithm, Flowchart.

Part2

Introduction of C language, Characteristics of C language, Sample program of C language, Typing, Compiling and running C program, Structure of C program, C-Tokens: C character set, Constants,Variable, Keywords, Operators, Data Types, Variable Declaration.

Part3

Console Input output statement: Formatted & unformatted I/O, *scanf():printf(), getch, getche, getchar, putch, putchar.* Operators & Expressions: Arithmetic Operators, Relational Operators, Logical operators, Assignment operators, Increment(++) & Decrement(--) operator, Shorthand operators, Conditional (Ternary) operator, Bitwise operators. Rules for operation, Type Conversion: Implicit type conversion, Type casting. Operator precedence / priority & Associativity.

Part4

Control Statement: Conditional Control Statement- if, if-else, else if ladder, nesting of if-else, switch-case-default. Looping Control Statement- while, do-while, for, while v/s do-while, Nesting loop. Jumping Control Statements: break, continue, goto, avoid goto.

Part5

Array: One, two, Multi dimensional, String: accessing string using %s and gets() and puts(), String manipulation function. Structure, array of Structure, Structure within structure, Union, Enumeration.

Part6

Function: need of function, Types of function, Library function, User defines function, Working with user defined function: Function Definition, A return statement, Function Declaration, Function calling, Function terminology: Calling function v/s Called function, actual argument v/s Formal arguments. Function calling technique: Call by value or pass by value, Call by reference or pass by reference. Recursion, Command line argument, Local scope v/s Global scope, Lifetime of a variable, Static variable, Storage class, Identifier, Modifier, Qualifier, sizeof, typedef.

Part7

Pointer: Pointer operator, Advantages of pointer, DMA: malloc(), calloc(), realloc(), free(). Pointer to pointer, Pointer Arithmetic, Pointer of array, Array of Pointer, Pointer of string, Passing pointer of data, Passing array to function, Returning pointer from function, Pointer of function, Pointer of structure, Self Referencial Structure, Passing structure to function, returning structure from function, copy all members of structure into another.

Part8

File Handling: Properties of file, Operation on file, Types of file, File Pointer and mode, C libray function for file handling: fopen() and fclose(), fputc() & fgetc(), putw() & getw(), fputs() & fgets(), fprintf() & fscanf(), fwrite() & fread(). Sequential v/s Random access of file: feof(), ftell(), rewind(), fseek().

Part9

Preprocessor: Preprocessor statements, File include, Macro Directives, passing argument to macro, Conditional compilation: #ifdef, #ifndef, #if, #elif, #else, #endif.

PART-1

Basics of programming:
Program
Code
Computer Language
Problem, Programming
C compiler
Software
Hardware
Keyboard
Monitor
RAM
Hard disk
CPU
Peripherals
Console
Standard I/O
Flow of data
Algorithm
Flowchart

Basics of programming

Program: A set of instructions / commands / statements called program.

निर्देशों/कमांड/स्टेटमेंट के समूह को प्रोग्राम कहते हैं।

Code: Instructions which are written in particular language like C called code.

ऐसे निर्देश जिन्हे किसी विशेष भाषा जैसे सी मे लिखा जाता है उन्हे कोड कहा जाता है।

Computer Language: A language in which we write code and perform that task called computer language. Example: Binary, assembly, C, C++, Java, C#, php, Visual Basic etc.

एक ऐसी भाषा जिसमें हम कोड को लिखकर उस कार्य को कम्प्युटर द्वारा संपादित करवाते हैं उसे कम्प्युटर भाषा कहते हैं। जैसे बाइनरि, एसम्ब्ली, सी,सी++, जावा,सी#, पीएचपी, विजुअल बेसिक आदि।

Problem: Given task that has to be performed by computer called problem.

दिया गया कार्य जिसे कम्प्युटर द्वारा संपादित करवाना हो उसे प्रोब्लेम कहते हैं।

Programming: The art of writing codes in a computer language to solve problem and solution is ready in the form of program then it is called programming.

प्रोब्लेम को हल करने के लिए जब किसी कम्प्युटर भाषा मे कोड को लिखा जाता है और उसका हल प्रोग्राम के रूप मे तैयार हो जाए तो इसे प्रोग्रामिंग कहते हैं।

C-compiler: A programs that checks code of C program and convert whole C code into machine code. Example: Turbo C, gcc, ANSI C etc.

एक ऐसा प्रोग्राम जो सी प्रोग्राम के कोड को चेक कर उसे मशीन कोड मे परिवर्तित कर दे उसे सी कंपाइलर कहते हैं। जैसे टर्बो सी, gcc, ANSI सी आदि।

Software: Program to solve problem called software / application.

किसी प्रोब्लेम को हल करने वाले प्रोग्राम को ही सॉफ्टवेर / एप्लिकेशन कहते हैं।

Hardware: All physical components of computer machine called hardware. For example-keyboard, monitor, RAM, hard disk, CPU.

कम्प्युटर मशीन के सभी भौतिक उपकरणों को हार्डवेर कहते हैं। जैसे कीबोर्ड, मॉनिटर, रेम, हार्ड डिस्क, सीपीयू आदि।

Keyboard: Input device used to type program and input data.

यह एक इनपुट उपकरण होता है जिसका उपयोग प्रोग्राम को टाइप करने और डाटा को इनपुट करने के लिए करते हैं।

Monitor: Output device used to show program and output results.

यह एक आउटपुट उपकरण होता है जिसका उपयोग प्रोग्राम और परिणाम को देखने के लिए करते हैं।

RAM (Random access memory) – Primary Memory device used to load program and store data temporary.

यह एक मेमोरी डिवाइस होता है जिसका उपयोग प्रोग्राम को लोड करने और डाटा को अस्थाई रूप से स्टोर करने के लिए करते हैं।

Hard disk: Secondary storage device used to store progam and result in the form of file parmanently.

यह एक सेकंडरी स्टोरेज उपकरण है जिसका उपयोग प्रोग्राम और परिणाम को फ़ाइल के रूप मे स्थायी रूप से स्टोर करने के लिए करते हैं।

CPU (Central Processing Unit): Processing device used to control program and executes all operators of program using ALU (Arithmetic Logic Unit).

यह एक प्रोसेसिंग उपकरण है जिसका उपयोग प्रोग्राम को नियंत्रित करने और सभी ऑपरेटर को **ALU**(अरिथमेटिक लॉजिक यूनिट) द्वारा क्रियान्वित करने के लिए करते हैं।

Peripherals: All Input / Output devices commonly called peripherals.

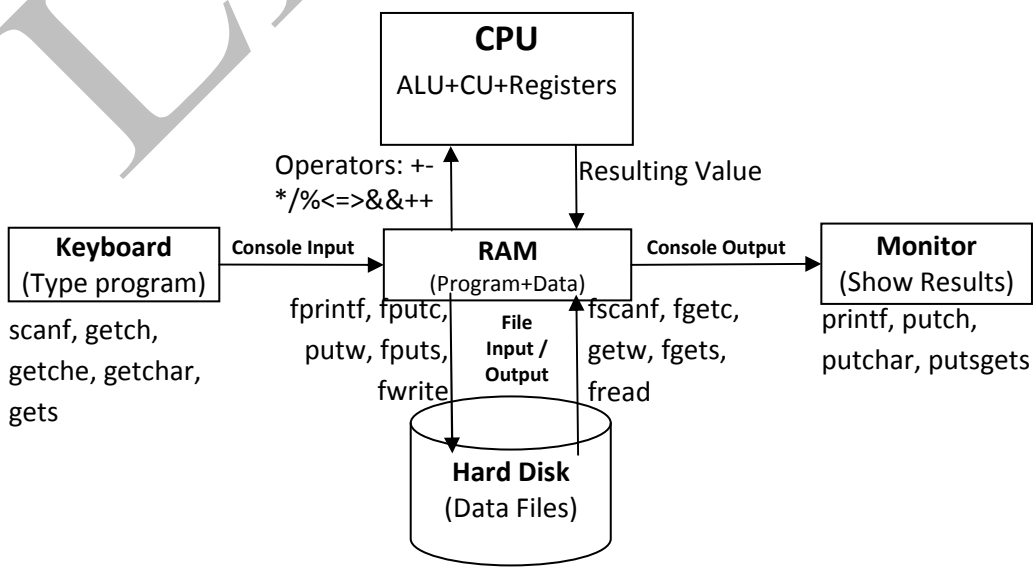
सभी इनपुट और आउटपुट उपकरणों को संयुक्त रूप से पेरिफेरल्स कहा जाता है।

Console: Combination of keyboard and monitor called console (terminal).

कीबोर्ड और मॉनिटर के कॉम्बिनेशन को कन्सोल(टरमीनल) कहते हैं।

Standard I/O: Keyboard, monitor and hardisk.

Flow of data in computer using I/O library functions of C:



Algorithm

When any problem is breaking into number of steps so that it can be solved then collection of steps are called algorithm.

जब किसी समस्या को कई चरणों मे लिखकर हल किया जाता है तब इन चरणों के समूहों को एल्गॉरिथ्म कहा जाता है।

Characteristic:

1) Always written in English.
2) Problem is break into number of steps.
3) Meaning of each step must be clear.
4) Next step is depending on completion of previous step.
5) Steps must be finite.
6) Up to last step problem has to be solved.

विशेषताएँ:

१. एल्गॉरिथ्म को हमेशा इंग्लिश मे ही लिखा जाना चाहिए।
२. समस्या को छोटे छोटे कई चरणों मे लिख लेते है।
३. प्रत्येक चरण कर अर्थ स्पस्ट होना चाहिए।
४. एक चरण के पूर्ण होने के बाद ही अगला चरण शुरू होता है।
५. चरण की संख्या निश्चित होना चाहिए।
६. अंतिम चरण होने तक समस्या का हल प्राप्त हों जाना चाहिए।

Example:

```
Algorithm: Given number is prime or not.
Step1: Input number  n.
Step2: set I=2
Step3: Check I<n. If false goto step6.
Step4: Check n MOD 2 is 0. If true goto step7.
Step5: i=i+1 then goto step3.
Step6: print-Prime number. goto step8.
Step7: print-Not Prime number.
Step8: Exit
```

Advantage:

- Good approach before coding in any computer language.
- Any person can also understand problem.

Limitation:

- It is not user friendly.
- Flow of steps are not clear.

लाभ:

- किसी भी कम्प्युटर भाषा मे कोड लिखने से पहले यह एक अच्छा विकल्प है।

- आलोगोरिथ मे लिखी गई समस्या के हल को कोई भी सरलता से समझ सकता है।

कमियाँ:

यह यूजर फ्रेंडली नहीं है।

चरणों का प्रवाह समझना मुश्किल होता है।

Flowchart

Pictorial representation of any problem is called flowchart.

किसी भी समस्या का चित्रात्मक वर्णन को फ्लोचार्ट कहा जाता है।

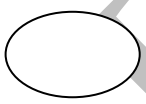
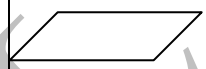
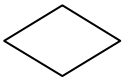
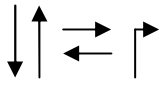
Characteristics:

1. Each task is defined in symbol.
2. Specific symbols are used for different task.
3. Symbols are linked via flow line.

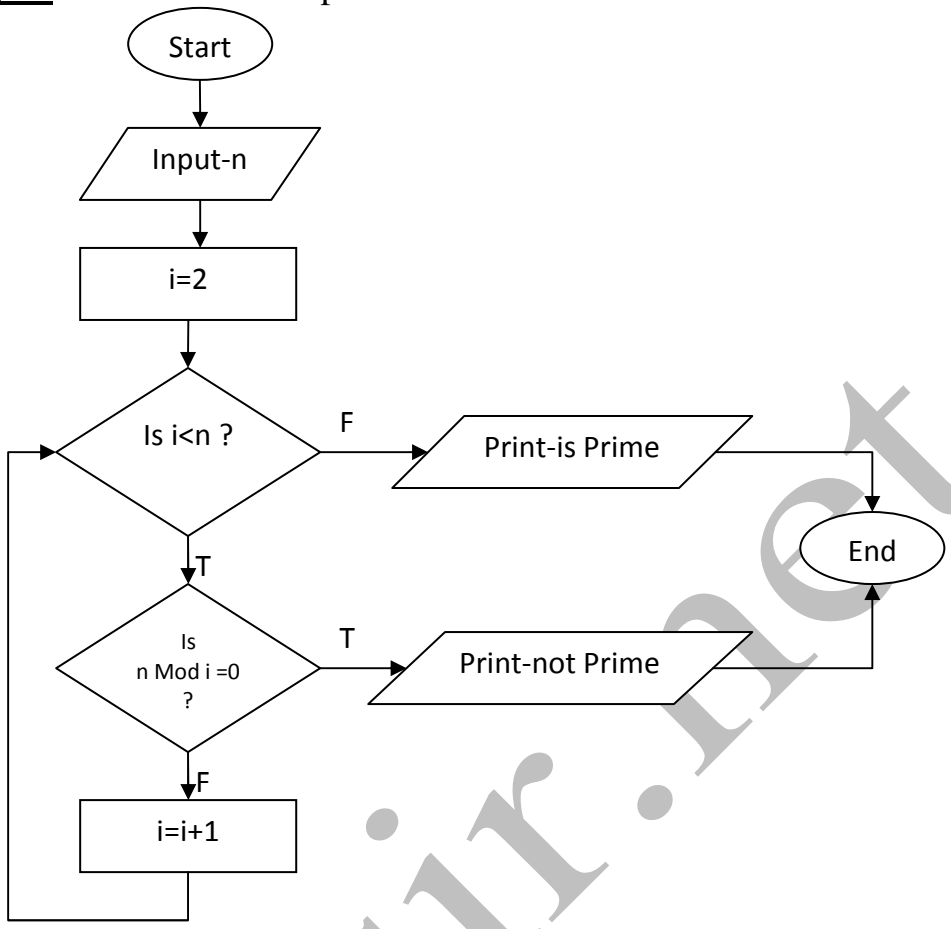
विशेषताएँ:

1. प्रत्येक कार्य चिन्हो मे परिभाषित होते है।
2. प्रत्येक कार्य का एक निश्चित चिन्ह होता है।
3. सभी चिन्ह फ्लो लाइन के द्वारा जुड़े होते है।

Symbol used in flowchart:

SNo	Symbol	Name	Uses
1		Oval Shape	Start / End of flowchart
2		Parallelogram	Input /Output operation
3		Rectangle	Internal Processing
4		Diamond	Decision making
5		Arrow	Connect Symbols
6		Circle	Connector one part of flowchart on next page.

Example: Given number is prime or not.



Advantages:

- Better clarity as compared to algorithms.
- Program can be easily coded by following flowchart.
- Flow of program can easily express by flow line.

Limitation:

- Needs large space.
- Not suited for a large problem.

लाभ:

एल्गॉरिथ्म की तुलना मे यह अधिक स्पस्ट है।

फ्लोचार्ट का अनुसरण करते हुये प्रोग्राम बनाना सरल हो जाता है।

फ्लो लाइन के द्वारा प्रोग्राम के फ्लो को सरलता से व्यक्त कर सकते है।

कमियाँ:

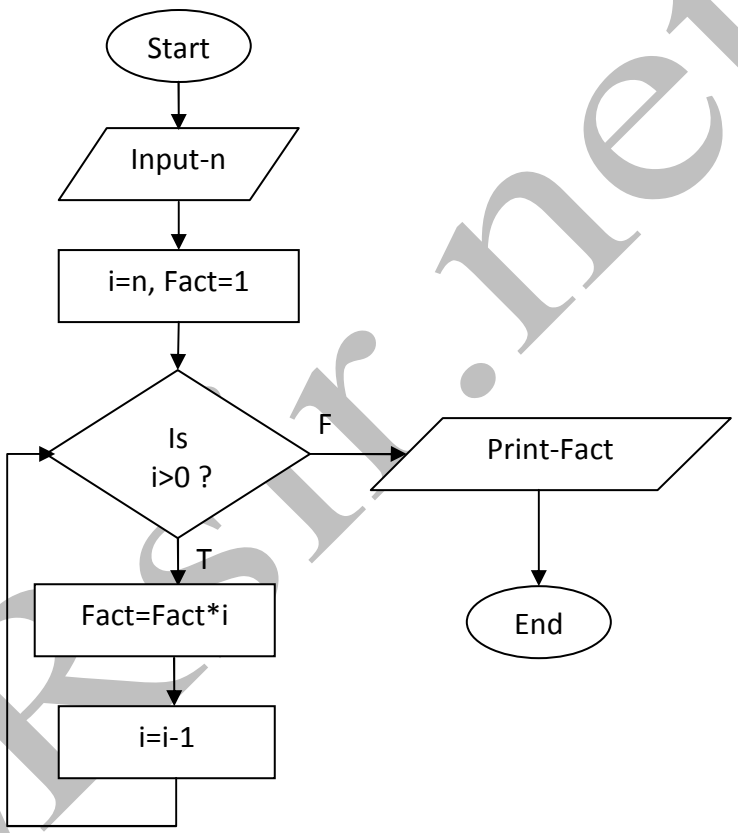
फ्लोचार्ट बनाने के लिए अधिक जगह की आवश्यकता होती है।

बड़ी प्रोब्लेम के लिए उपयुक्त नहीं है।

Algorithm and flowchart for factorial of given number

Algorithm: Factorial of given number.
Step1: Input number: n
Step2: Initialize: i = n, fact=1.
Step3: Check while i>0 If false go to step:
Step4: fact=fact*i.
Step5: **i**=i-1 and go to step3.
Step6: Print- fact
Step7: Exit

Flowchart for factorial



PART 2

Introduction of C language
Characteristics of C language
Sample program of C language
Typing, Compiling and running C program
Structure of C program
C-Tokens:
- C character set
- Constants
- Variable
- Keywords
- Operators
- Data Types

Variable Declaration

Introduction of C language

Before 1960 programming languages were developed on the basis of working areas, like COBOL language for business field, FORTRAN language for science and engineering field and so on. Thus programmer had to learn all programming languages for different fields. To use one language for all fields, International committee was developed a new language called "ALGOL60". It were a lot of limitations to programming for all fields therefore a number of new languages were developed one after one in 10 years like CPL, BCPL, B language etc. All these languages were unsuccessful to program for all fields.

Finally in 1972, C-language was introduced by Mr. Dennis Ritchie at invented at AT & T's Bell Laboratory, USA.

सन् 1960 के पहले कार्य क्षेत्र के आधार पर कम्प्युटर भाषाओं का निर्माण किया जाता था। जैसे व्यापार से संबन्धित कार्य के लिए कोबोल भाषा, विज्ञान और अभियांत्रिकी से संबन्धित कार्य के लिए फ़ोरट्रान भाषा इत्यादि। इसलिए प्रोग्रामर को सभी प्रकार की फील्ड के लिए एक से अधिक प्रोग्रामिंग भाषाओं का ज्ञान होना जरूरी होता था। सभी फील्ड के लिए केवल एक ही भाषा का उपयोग किया जा सके इसके लिए 1960 मे इंटरनेशनल कमेटी द्वारा "ALGOL60" नाम की भाषा का निर्माण किया गया। इसमे कुछ कमियाँ थी जिनको दूर करने के प्रयास मे एक के बाद एक 10 सालों मे CPL, BCPL, B भाषाओं का निर्माण किया गया, किन्तु सभी पूर्ण रूप से अलग अलग फील्ड की प्रोग्रामिंग के लिए सफल नहीं हो सकी।

अंतत: सबसे अंत मे 1972 मे C-भाषा का निर्माण मिस्टर डेनिस रिची के द्वारा AT&T's बैल लैबोरेटरी USA मे किया गया।

Characteristics of C language

1. **One for all:** using C language, we can write program for all field like business or science field. We do not need to learn different language for every field.
2. **Small & simple language:** C language is smaller than other languages. It has only 32 keywords and due to English like language, C is easy to learn and simple.
3. **Easy to debug:** Code errors can be easily traced and removed using debug tool of c language.
4. **Powerful:** Using C language, we can developed any type of software like operating system (Unix), hardware's driver etc.
5. **Middle level:** Writing code in C language is easy as compared to low language(Machine and assembly) and execution speed of C program is faster as compared to high level language. Therefore C language is also called middle level language.

6. **Portable:** Program that write using C language can be running on any computer machine and operating system.
7. **Mother language:** When we have to learn first C language then learning all advance languages becomes easy like C++, Java, C# etc.
8. **POP(Procedure Oriented Programming) language:** All codes of program can be divided in to a number of blocks(procedures) and these can be used on same program or different programs.
9. **Compile nature:** First, all lines (statements) of program are checked for any errors by C compiler program (gcc / tc), after then error free program is converted into binary / machine code. Later that code run by operating system.
10. **Reliable and robustness:** Program developed using C language can work on any system therefore we can believe on C language for the programming of any fields.
11. **Free form language:** Every C instruction ends with semi column(;) and a set of instructions (compound statement) are enclosed in curly braces {}. In this way we can write more than two instructions in a single line.

सी-भाषा की विशेषताएँ:

१. **सभी फील्ड की एक भाषा:** सी भाषा के द्वारा किसी भी फील्ड की प्रोग्रामिंग की जा सकती है। अर्थात प्रत्येक फील्ड के लिए अलग अलग भाषाओं को नहीं सीखना पड़ता है।

२. **छोटी और सरल भाषा:** सभी भाषाओं की तुलना मे सबसे छोटी भाषा है, केवल ३२ किवर्ड्स है और इंग्लिश लाइक होने के कारण सीखना बहुत सरल है।

३. **आसान debugging:** सी भाषा मे बने प्रोग्राम की त्रुटियों(error) को आसानी से दूर किया जा सकता है।

४. **पावरफुल:** सी भाषा के द्वारा ऑपरेटिंग सिस्टम(यूनिक्स) और हार्डवेर के ड्राइवर जैसे सिस्टम सॉफ्टवेर का निर्माण किया जा सकता है।

५. **मध्य स्तरीय:** निम्न स्तरीय भाषा की तुलना मे प्रोग्राम का कोड सी भाषा में लिखना सरल है तथा अन्य उच्च स्तरीय भाषा की तुलना मे सी भाषा मे लिखा हुआ प्रोग्राम अधिक तीव्र गति से कार्य कर सकता है। इसलिए सी भाषा को मध्य स्तरीय भाषा भी कहा जाता है।

६. **पोर्टेबल:** सी भाषा मे बने हुये प्रोग्राम को किसी भी कम्प्युटर मशीन और ऑपरेटिंग मशीन पर रन किया जा सकता है।

७. **मातृभाषा:** पहले सी भाषा सीखने से एडवांस भाषाएँ जैसे सी++, जावा, सी# आदि को सीखना सरल हो जाता है।

८. **POP(प्रोसीजर ओरिएंटेड़ प्रोग्राम) भाषा:** प्रोग्राम के कोड को छोटे छोटे ब्लॉक मे विभक्त कर सकते है जिन्हे आवश्यकतानुसार उसी प्रोग्राम या अन्य प्रोग्राम मे उपयोग कर सकते है।

९. **कंपाइल प्रक्रति:** प्रोग्राम की सभी लाइन को सबसे पहले सी-कंपाइलर प्रोग्राम(gcc, tc) के द्वारा जांचा जाता है, त्रुटिरहित होने पर ही कंपाइलर सी प्रोग्राम को बाइनरी कोड (मशीन कोड) मे परिवर्तित कर देता है, जिसे ऑपरेटिंग सिस्टम द्वारा रन कर देता है।

१०. **भरोसेमंद और अडिग:** सी मे बना प्रोग्राम किसी भी सिस्टम पर कार्य करने की क्षमता रखता है इसलिए किसी भी फील्ड की प्रोग्रामिंग के लिए सी-पर भरोसा किया जा सकता है।

११. **फ्री फॉर्म भाषा:** प्रत्येक निर्देश का अंत अल्प विराम(;) से होता है और निर्देशों के समूह (कंपाउंड निर्देश) को मझला कोष्ठक {} मे लिखते है। इस प्रकार से एक से अधिक निर्देशों को एक ही लाइन मे भी लिखा जा सकता है।

Sample program of C language

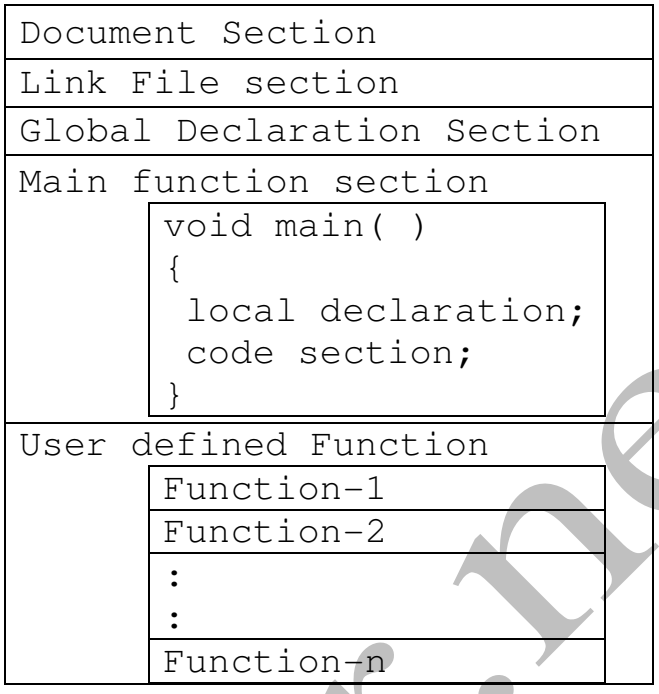
```
#include<stdio.h>
#include<conio.h>
void main()
{
      int a,b,c;
      clrscr();
      printf("\nInput two numbers:");
      scanf("%d%d", &a,&b);
      c=a+b;
      printf("\nSum=%d", c);
      getch();
}
 Output:
 Input two numbers: 10   20 <Enter>
 Sum=30
```

Typing, Compiling and running C program

1. Install Turbo C software in DOS supported operating system.
   टर्बो सी सॉफ्टवेर को **DOS** आधारित सिस्टम मे इन्स्टाल करते हैं।
2. Open TC.exe file from location c:\TC\BIN\.
   TC.exe फ़ाइल को **c:\ TC\BIN** लोकेशन से ओपेन करते हैं।
3. Type C program and save(F2).(ex: prog1.c)
   सी प्रोग्राम को टाइप कर सेव करते है।
4. Compile using Alt+F9.
   Alt+F9 दबाकर प्रोग्राम को कंपाइल करते है।
5. If program is error free then run using Ctrl+F9.
   यदि प्रोग्राम त्रुटि रहित है तब **ctrl+F9** दबाकर प्रोग्राम रन कर देते है।
6. Check output screen using Alt+F5.
   आउटपुट स्क्रीन को देखने के लिए **alt+F5** दबाते हैं।

Structure of C program

A complete structure of C program is divided into following 5 sections.



1) Document Section: It is used to write information about program like title, developer, date etc. To ignore such information at compile time, we write in C comment(/*------*/).

इस सेक्शन का उपयोग प्रोग्राम से संबन्धित जानकारियों को लिखने के लिए करते है जैसे प्रोग्राम का टाइटल, डेवलपर, डेट आदि। ऐसी जानकारियों को कंपाइल टाइम पर इग्नोर करने के लिए सी कमेंट (/*------*/) का उपयोग करते हैं।

Ex:
```
/*    Title: Addition of two number
      Author: www.LRsir.net -9425034034
      Date: 14/11/1978
*/
```

2) Link File Section: It is used to include those header files that contain declaration of library functions which are using in current program. For example, printf() and scanf() are library functions which are declared in stdio.h header file.

इसका उपयोग ऐसी हैडर फ़ाइल को प्रोग्राम मे शामिल करने के लिए करते हैं जिसमे उपयोग मे लाये जा रहे लाइब्रेरी फंक्शन का डिक्लेरेशन होता है। जैसे, **printf()** और **scanf()** लाइब्रेरी फंक्शन **stdio.h** हैडर फ़ाइल मे डिक्लैर हैं।

Ex:`#include<stdio.h>`

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

3) Global Declaration: Anything that can be shared by all functions of program is declared in this section like declarations of variable and user define function.

सी प्रोग्राम मे वे सभी कुछ जिन्हे सभी फंक्शन के द्वारा शेयर किया जा सके उन सभी को इस सेक्शन मे डिक्लैर किया जाता है। जैसे वेरियबल और फंक्शन का डिक्लैरेशन।

Ex:
```
int x;
int add(int,int);
```

4) Main function Section: This is compulsory section for every C program. In this section first we declare local variables then write code. Main function is also called driver function because program run from its first line and ends to its last line.

प्रत्येक सी प्रोग्राम मे यह सेक्शन अनिवार्य है। इसमे सबसे पहले वेरियबल को डिक्लैर करते है उसके बाद कोड को। इस फनशन को ड्राईवर फंक्शन भी कहा जाता है क्यूंकी इसी की पहली लाइन से प्रोग्राम रन होना शुरू होता है और अंतिम लाइन पर खत्म होता है।

Ex:
```
void main()
  {
    int a=5,b=4,c;
    c=add(a,b);
    printf("%d",c);
  }
```

5) User defined Function Section: In this section we write code separately that needs more than one times by other function.

इस सेक्शन में हम ऐसे कोड को सेपरेट फनशन मे लिखते है जिनकी जरूरत अन्य फंक्शन द्वारा बार बार होती है।

Ex:
```
int add(int r1,int r2)
{
 x=r1+r2;
 return(x);
}
```

Open TC.exe→Type program→save to file **prog1.c** in to "c:\tc\bin\"

```
/*Title: Addition of two number
 Author: www.LRsir.net -9425034034
```

```
    Date: 14/11/1978
*/
#include<stdio.h>
#include<conio.h>
int x;
int add(int,int);
void main()
{
  int a=5,b=4,c;
  c=add(a,b);
  printf("%d",c);
}
int add(int r1,int r2)
{
  x=r1+r2;
  return(x);
 }
```

Perform following steps:

1. **Compile (Alt+F9):** To check error & convert into machine code
2. **Run (Ctrl+F9):** To execute program
3. **Output(Alt+F5):** To see output screen.

Ex: 9

C-Tokens

Smallest unit of C program through which we define data and write programming code, called C-tokens. It consists following things.

1) C character Set
2) Constants
3) Variables
4) Keywords
5) Operators

C character set:

It includes all ASCII characters of 1 Byte such as all lower and upper alphabets (a to z, A to Z), digits (0-9), special symbols (+ - = . # etc) and back slash characters (\n, \t, \a, \\, \', \", \0 etc).

इसमें उन सभी **ASCII** कैरक्टर को शामिल किया गया है जिनकी साइज़ 1 बाइट होती है। जैसे अंग्रेजी के सभी छोटे और बड़े अक्षर (**a-z, A-Z),** अंक(0-9), विशेष चिन्ह (+ - = . **#** आदि) एवं बैक स्लेश कैरक्टर(**\n, \t, \\, \', \", \0** आदि)।

Constants:

A quantity that remains unchanged during the program execution. It includes integer constant (ex. 122, 345), float constant (ex. 23.34, 2e7, .2E3) and character constant ('a', '0', '+', '\0').

एक ऐसी राशि जो प्रोग्राम के क्रियान्वयन के समय अपरिवर्तित रहती है, उसे कोंस्टेंट कहते हैं। इसमे शामिल है- इंटीजर कोंस्टेंट (122, 345), फ्लोट कोंस्टेंट(23.34, 2**e7**, .2**E3**) एवं कैरक्टर कोंस्टेंट('a', '0', '+', '\0').

Variable:

A quantity that may be changed during the program execution. It is an identifier used to access data. (ex. p, r, t and si are variable to hold data for simple interest).

एक ऐसी राशि जो प्रोग्राम के क्रियान्वयन के समय परिवर्तित होती रहती है उसे वेरिएबल कहते हैं। यह एक आइडेंटिफ़ायर होता है जिसके द्वारा डाटा को एक्सैस कर सकते है। (उदा. **p, r, t** और **si** का उपयोग साधारण ब्याज के डाटा को होल्ड करने के लिए कर सकते है)

Keywords:

Reserved word of C language that can be used to define an instruction. Only 32 keywords are available in C language. All keywords are written only in small letter.

सी भाषा के सभी आरक्षित शब्द जिनके द्वारा निर्देशों को परिभाषित करते है उन्हे कीवर्ड कहते हैं। सी भाषा में मात्र 32 कीवर्ड है। सभी कीवर्ड को अंग्रेजी की छोटी वर्ण मे लिखा जाता है।

(ex. `void, int, float, char, signed, unsigned, short, long, double, return, if, else, switch, case, default, do, while, for, break, continue, goto, struct, union, enum, auto, register, static, extern, typedef, const, near and far`).

Operators:

Special symbols used to operate operands called operators.

विशेष चिन्ह जिनके द्वारा डाटा को ओपरेट करते है उन्हे ऑपरेटर कहते हैं।

ex: Arithmetic (+, -, *, /, %), Relational (<,<=, >, >=, ==,!=), Logical (&&, ||, !), Assignment (=, +=, -=, *=, /=, %=), Increment (++), decrement (--), Conditional (? :), Bitwise (&, |, ~, ^, >>, <<)

Data Types

A quantity that can be used define data at the time of variable declaration and function creation called data types. All data types are classified into following categories.

एक ऐसी राशि जिसका उपयोग वेरियबल को डिक्लैर करते समय तथा फंक्शन को बनाते समय किया जाता है उसे डाटा टाइप कहते हैं। सभी डाटा टाइप को निम्न भागों मे वर्गीक्रत कर सकते हैं।

1. **Primitive Data type** (Primary /Basic / fundamental / predefine / inbuilt data type)
   1.1 ***Numeric data type:*** It is used to define integer and float data types.
   इसका उपयोग इंटीजर और फ्लोट डाटा टाइप्स को परिभाषित करने के लिए करते हैं।
   C keywords for integer data type:
   int (Space 2 Byte)
   long(Space 4 Byte)
   C keywords for float data type:
   float (Space 4 Byte)
   double(Space 8 Byte)
   1.2 ***Non numeric data type:*** It is used to define character data type.
   इसका उपयोग कैरक्टर डाटा टाइप के लिए करते हैं।
   C keywords for character data type:
   char (Space 1 Byte)
2. **Non primitive data types** (Secondary data type)
   2.1 ***Derived Data type***: It uses existing one data type for multiple data.
   इसका उपयोग उपलब्ध डाटा टाइप के द्वारा कई सारे डाटा के लिए करते हैं।
   Example: Array, Pointer
   2.2 ***User define data type:*** It also uses existing different data types for multiple data.
   इसका उपयोग भी उपलब्ध डाटा टाइप के द्वारा कई सारे अलग अलग प्रकार के डाटा टाइप के लिए करते हैं।
   Example: Structure, Union.

Variable Declaration

Place holder of data is called variable. It is declared using following syntax.
डाटा को जिसमे होल्ड किया जाता है उसे वेरियबल कहते हैं। इसे निम्न सिंटेक्स के द्वारा डिक्लैर करते हैं।

Syntax: `data-type var1, var2, …., varn;`

Example:
```
int a, b,c;
float p,q,r;
char ch;
```

Initialization of variable: Default value of variable is garbage. Assign any initial value of variable called initialization.
वेरियबल की प्रारम्भिक वैल्यू को इनिशिलाइजशन कहते है। वेरियबल की डिफ़ाल्ट वैल्यू **garbage** होती है।

Ex:
```
int a=10;
float pi=3.14;
```

Variables can be declared inside function (local) or outside the function (global).
वेरियबल को फंक्शन के अंदर(लोकल) या बाहर(ग्लोबल) डिक्लैर कर सकते हैं।

PART3

Console Input output statement:
- Formatted & unformatted I/O
- scanf()
- printf()
- getch()
- getche()
- getchar()
- putch()
- putchar()

Operators & Expressions:
- Arithmetic Operators
- Relational Operators
- Logical operators
- Assignment operators
- Increment(++) & Decrement(--) operator
- Shorthand operators
- Conditional (Ternary) operator
- Bitwise operators

Rules for operation

Type Conversion:
- Implicit type conversion
- Type casting

Operator precedence / priority & Associativity.

Console Input output statement

A statement that can be used to read data from keyboard called input statement and that used to show data on monitor called output statement.
All I/O statements are classified into following two fields.
एक ऐसा स्टेटमेंट जो कीबोर्ड से इनपुट किए गए डाटा को रीड करे उसे इनपुट स्टेटमेंट और जो डाटा को मॉनिटर पर दिखा सके उसे आउटपुट स्टेटमेंट कहते हैं। सभी **I/O** स्टेटमेंट को निम्न दो भागों मे वर्गीकृत किया गया है।

1. formatted I/O statement 2. Unformatted I/O statement

1) **Formatted I/O:** They are used to access one or more data.
इनका उपयोग एक या अधिक डाटा को एक्सैस करने के लिए करते हैं।

a) ***scanf():*** It is a library function of *stdio.h* header file, that can read one or more data from keyboard and assign to the specified variables.
यह **stdio.h** हैडर फ़ाइल का एक फंक्शन है जो कीबोर्ड से एक या अधिक डाटा को रीड कर दिये गए वेरियाब्ल्स मे असाइन कर देता है।

Syntax:

```
scanf("formatted codes", &var1, &var2, …..,&varn);
```

Formatted codes are:
%d for integer
%f for float
%c for character

At runtime, scanf will stop execution. We input sufficient data using keyboard. When we press <Enter> key after very data then data assigning to specified variable.

रन टाइम पर, **scanf** क्रियान्वयन को रोक देता है। जब हम कीबोर्ड से पर्याप्त डाटा को इनपुट करते हैं। जब प्रत्येक डाटा को इनपुट कर **<enter>** की प्रैस करते हैं तब दिये गए वेरियबल मे डाटा असाइन हो जाता है।

b) ***printf():*** It is also library function of *stdio.h* header file, that can show data of one or more variables, constant and message on monitor.
यह भी **stdio.h** हैडर फ़ाइल का एक लाइब्रेरी फंक्शन है जो एक या अधिक वेरियाब्ल्स, कोंस्टंट और मैसेज को मॉनिटर दिखा सकता है।

Syntax:

```
printf("formattedCodes/Message/backSlashCharacter",
        var1, var2, ….., varn);
```

Formatted code:

% d	for integer
%f	for float
%c	for character

Backslash character:

\n	move curser to new line
\t	move curser to 8 space
\\	to use \
\"	to use "
\'	to use '

Remark: Formatted code will be skip when we do not shows values of variable.

फोरमेटेड कोड को छोड़ देते हैं जब हमे वेरियाब्ल्स की वैल्यू को नहीं दिखाना हो।

```
Example of printf() and scanf():
#include<stdio.h>
void main()
{
 char a;
 int b;
 float c;
 printf("\nInput character, integer then float value:");
 scanf("%c%d%f", &a, &b, &c);
 printf("\ncharacter=%c\tInteger=%d\tfloat=%f", a,b,c);
}
output:
Input character, integer then float value:
+ <enter>
23<enter>
2.5<enter>
Character=+     Integer=23      float=2.500000
```

2) **Unformatted I/O statement:** ***(getch, getche, getchar, putch, putchar)***: They are used access only one character data in different ways.

इनका उपयोग केवल एक हे कैरक्टर को अलग अलग तरीके से एक्सैस करने के लिए करते है।

Unformatted input statement:

a) **getch(): ** Read one character before <enter> but not shows and alter. It is declare in conio.h header file.

यह एक कैरक्टर को **<enter>** की के प्रैस होने से पहले ही रीड कर लेता है किन्तु दिखाता नहीं है और ना ही उसे बदल सकते हैं। यह **conio.h** हैडर फ़ाइल मे डिक्लैर है।

Syntax:

```
char-var=getch();
```

b) **getche():** Read one character before <enter> and shows but never alter. It is also declare in conio.h header file.

यह एक कैरक्टर को **<enter>** की के प्रैस होने से पहले ही रीड कर लेता है और दिखाता भी है किन्तु उसे बदल नहीं सकते हैं। यह **conio.h** हैडर फ़ाइल मे डिक्लैर है।

Syntax:

```
char-var=getche();
```

c) **getchar():** Read one character after <enter> and shows as well as can be alter. It is declare in stdio.h header file.

यह एक कैरक्टर को **<enter>** की के प्रैस होने के बाद ही रीड करता है और दिखाता भी है, साथ ही उसे बदल भी सकते हैं। यह **stdio.h** हैडर फ़ाइल मे डिक्लैर है।

Syntax:

```
char-var=getchar();
```

Unformatted output statement:

a) **putch():** Shows one character that in any form like constant, variable or ASCII value. It is declare in conio.h header file.

यह एक कैरक्टर जैसे कोंस्टंट, वेरियबल या **ASCII** वैल्यू के रूप मे हो, उसे दिखा सकता है। यह **conio.h** हैडर फ़ाइल मे डिक्लैर होता है।

Syntax:

```
putch(character);
```

b) **putchar**(): Shows also one character that in any form like constant, variable or ASCII value. It is declare in stdio.h header file.

यह एक कैरक्टर जैसे कोंस्टंट, वेरियबल या **ASCII** वैल्यू के रूप मे हो, उसे दिखा सकता है। यह **stdio.h** हैडर फ़ाइल मे डिक्लैर होता है।

Syntax:

```
putchar(character);
```

Example of unformatted I/O statement:

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

```
#include<conio.h>
void main()
{
 char ch;
 ch=getch();
 putch(ch);
 ch=getche();
 ch=getchar();
 putchar(ch);
}
```

Operators & Expressions

To operate data for different operation, c provides a list of symbols called operator and a statement that made up of data (operand) and operators called expressions.

विभिन्न प्रकार के ऑपरेशन के लिए जब डाटा को ऑपरेट करना हो तब सी भाषा मे विभिन्न साइन की एक लिस्ट होती है जिनहे ऑपरेटर कहते हैं और एक ऐसा स्टेटमेंट जो डाटा(ओपेरेंड) तथा ऑपरेटर से तैयार होता है उसे एक्स्प्रेशन कहते हैं।

Example: 2+3 is arithmetic expression. + is operator used to add operand 2 and 3.

2+3 एक अरिथमेटिक एक्स्प्रेशन है। + का यूस ओपेरंड 2 और 3 को जोड़ने के लिए करते हैं।

C supports following operators.

सी भाषा मे निम्न ऑपरेटर सहायक होते हैं।

1. Arithmetic Operators
2. Relational Operators
3. Logical operators
4. Assignment operators
5. Increment(++) & Decrement(--) operator
6. Shorthand operators
7. Conditional (Ternary) operator
8. Bitwise operators

1) Arithmetic Operators: Usable for mathematical calculations.
ये गणितीय गणना करने के लिए उपयोगी है।

Operator	Name	uses
+	Add	sum of two data
-	Subtract	subtract two data
*	Multiply	product of two data
/	Division	gives quotient of two data
%	Modulo	gives remainder of two data

```
Example:
#include<stdio.h>
void main()
{
 int a,b;
 printf("\nInput two integer data:");
 scanf("%d%d", &a, &b);
 printf("\nsum=%d", a+b);
 printf("\nsubtract=%d", a-b);
 printf("\nproduct=%d", a*b);
 printf("\ndivision=%d", a/b);
 printf("\nremainder=%d", a%b);
}
Output:
Input two integer data:
14
8
sum=22
subtract=6
product=112
division=1
remainder=6
```

Remark: % (modulo) operates on only integer data.

%(**modulo**) ऑपरेटर केवल इंटीजर डाटा को ही ऑपरेट करता है।

2) Relational Operators: Used to compare data numerical data.

इनका उपयोग संख्यात्मक डाटा की तुलना करने के लिए करते हैं।

Operator	**Name**
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal to
!=	Not Equal to

Example and uses

<	<=	>	>=	==	!=
4<5(true)	4<=5(true)	4>5(false)	4>=5(false)	4==5(false)	4!=5(true)
5<4(false)	5<=4(false)	5>4(true)	5>=4(true)	5==4(false)	5!=4(true)
5<5(false)	5<=5(true)	5>5(false)	5>=5(true)	5==5(true)	5!=5(false)

Relational operators are mostly applying on conditional and looping statement (if, while and for).

रिलेशनल ऑपरेटर का उपयोग कंडिशन और लूप स्टेटमेंट मे ही सबसे अधिक होता है।

3) Logical operator: Used to compare Boolean data (true / false).
इनका उपयोग बूलियन डाटा(**true/false**) की तुलना करने के लिए करते हैं।

Operator	Name as
&&	Logical AND
||	Logical OR
!	Unary Logical NOT

Example and uses

&& (both true mean true)	|| (one true mean true)	! (invert)
True && True (True) True && False (False) False && True (False) False && False (True)	True || True (True) True || False (True) False || True (True) False || False (False)	!True (False) !False (True)

True / False values are generated by combining relational operators.
True/false वैल्यूस को रिलेशनल ऑपरेटर के द्वारा उत्पन्न किया जाता है।
Ex: (4<5)&&(4==5) similar to True && False (False)

4) Assignment operators (=): Used to assign right side value to left variable. Only one = sign is used.
इसका उपयोग राइट साइड की वैल्यू को लेफ्ट वेरियबल मे असाइन करने के लिए करते हैं। = को एक ही बार लिखते हैं।

```
Example:
#include<stdio.h>
void main()
{
 int a, b;
 a=10;
 b=a;
 printf("\na=%d\tb=%d",a,b);
}
Output:
a=10          b=10
```

5) Increment(++) & Decrement(--) operator:

++ is used to increase variable's value by 1.
-- is used to decrease variable's value by 1.
++ का उपयोग वेरियबल की वैल्यू को 1 से बड़ाने के लिए करते है।
-- का उपयोग वेरियबल की वैल्यू को 1 से घटाने के लिए करते है।

These operators can be uses in following form –
इन ऑपरेटर को निम्न दो रूप से उपयोग मे लाते है।

Prefix(++a and --a): First variable's value increment / decrement then uses in current statement.

इस रूप मे सबसे पहले वेरियबल की वैल्यू बड़ती/घटती हैं उसके बाद करेंट स्टेटमेंट मे उपयोग होती है।

Postfix(a++ and a--): First variable's value uses in current statement then increment / decrement.

इस रूप मे सबसे पहले वेरियबल की वैल्यू करेंट स्टेटमेंट मे उपयोग होती है उसके बाद बड़ती/घटती हैं है।

```
Example:
#include<stdio.h>
void main()
{
 int a,b;
 a=10;
 b=10;
 printf("\n In Current Statement: a=%d\tb=%d", ++a, b++);
 printf("\n After Increment=%d\tb=%d", a,b);
}
Output:
In current statement:    a=11    b=10
After Increment:         a=11    b=11
```

6) **Shorthand operators:** Used to assign right side value to left variable before applying arithmetic operation.

   इनका उपयोग राइट साइड वैल्यू को लेफ्ट वेरियबल मे दिये गए अरिथमेटिक ऑपरेशन के साथ असाइन करने के लिए होता है।

Operator	example	similar to
+=	a+=2	a=a+2
-=	a-=2	a=a-2
*=	a*=2	a=a*2
/=	a/=2	a=a/2
%=	a%=2	a=a%2

7) **Conditional (Ternary) operator**: It has three parts.

   इसके तीन भाग होते हैं।

Syntax: `Expression1? Expression2: Expression3`

Expression1 gives true / false, Expression 2 evaluates for true value whereas expression3 evaluates for false value.

एक्स्प्रेशन1 **true/false** वैल्यू रिटर्न करता है। एक्स्प्रेशन2 **true** वैल्यू के लिए और एक्स्प्रेशन3 **false** वैल्यू के लिए सक्रिय होता है।

Example: find bigger value

```
#include<stdio.h>
void main()
{
 int a,b,c;
 a=10;
 b=15;
 c=(a>b)?a:b;
 printf("\bigger value=%d",c);
}
Output:
Bigger value=15
```

Since expression1 (a>b) is false therefore value of expression3 assign to variable c.

चूंकि एक्स्प्रेशन1(**a>b)** फाल्स है इसलिए एक्स्प्रेशन3 की वैल्यू वेरियबल **c** मे असाइन होगी।

8) Bitwise operator: Used to operate data at binary level.

इनका उपयोग डाटा को बाइनरि स्तर पर ऑपरेट करने के लिए करते हैं।

Operator	Name	Meaning
&	Bitwise AND	both1 means 1
|	Bitwise OR	any 1 means 1
~	Bitwise NOT	1 means 0
^	Bitwise XOR	both 1 or 0 means 0
<<	Left Shift(n<<2)	shift 2 bit of n towards left
>>	Right Shift(n>>2)	shift 2 bit of n towards right

Internally integer value convert into binary form then operator apply on bits one by one.

आंतरिक रूप से पहले इंटीजर वैल्यू उसके बाइनरि रूप मे परिवर्तित होगी फिर उनकी बिट्स को एक के बाद एक ऑपरेट होगी।

Ex:

Result of 5&6 is 4 due to following internal operation

1	0	1	binary value of 5
&	&	&	bitwise & operator
1	1	0	binary value of 6
1	0	0	binary value of 4

Type of Expression:

1) Arithmetic expression: 2+3 or a+2 or a/b
2) Relational expression: **2**<5 or a==2 or a>b
3) Mixed expression : a<b && a<3

Rules for operation:

1) Data must be similar type before operation.
   ऑपरेशन के पहले डाटा एक समान टाइप के होने चाहिए।
2) Resulting data type is similar to operand type.
   ओपेरेंड्स का जो टाइप होता है परिणाम भी उसी टाइप का प्राप्त होता है।

Example: data1 **operator** data2 gives resulting data

Data1	**Data2**	**Result data**
Integer	Float	Invalid
Integer	Integer	Integer
Float	Float	Float
Character	Character	Character

Type Conversion

Before applying operators, type of data should be similar. e.g. 2 + 3.4 is invalid expression because 2 is integer and 3.4 is float. Thus we have to convert 2 into float(2.0) or 3.4 into integer(3) so that both data becomes same type, then it is called type conversion.

ऑपरेटर को अप्लाई करने से पहले, डाटा का टाइप एक समान होना चाहिए। जैसे 2+3.4 एक तरह से अमान्य होता है क्यूंकी 2 इंटीजर और 3.4 फ्लोट है। अत: हमे 2 को फ्लोट टाइप (2.0) या 3.4 को इंटीजर टाइप (3) मे परिवर्तित करना होगा जिससे कि दोनों डाटा एक समान टाइप के बन जाएँ। इसे ही टाइप कन्वर्शन कहते हैं।

In C language, type conversion performs in following two ways.
सी भाषा मे टाइप कन्वर्शन निम्न दो प्रकार से हो सकता है।

1) **Implicit type conversion(Auto conversion):**
   When input data types are not same types then C compiler, automatically convert small size of data into bigger one, so that actual value persist. It is called implicit type conversion.
   जब इनपुट डाटा के टाइप्स एक समान नहीं हों तब सी कंपाइलर स्वत: ही ऊनमे से छोटे वाले डाटा को बड़े वाले डाटा के टाइप मे परिवर्तित कर देता है जिससे कि वास्तविक वैल्यू बनी रहती है। इसे ही एंप्लीसिट कन्वर्शन कहते हैं।
   2+3.4 becomes 2.0+3.4
2) **Explicit type conversion(Type casting):**
   If we want to force type conversion of any data into another called type casting. Actual value may be loss.
   यदि हम एक टाइप के डाटा को किसी अन्य टाइप मे बलपूर्वक परिवर्तित करते है तब इसे टाइप कास्टिंग कहते हैं। वास्तविक वैल्यू लॉस हो सकती है।

Syntax:

```
(cast-type) variable
```

Cast-type = int / float / char etc

```
Example: Division of integer number.
#include<stdio.h>
void main()
{
 int a=10,b=4;
 float c;
 c= (float) a / b;
 printf("division=%f", c);
}
Output:
division=2.500000
```

When value of **a** is casted to float so value of b automatically converts into float, thus result will be float, therefore we choose float for variable c.

जब वेरियबल **a** की वैल्यू को **float** मे कास्ट करते हैं तब **b** की वैल्यू स्वत: **float** मे परिवर्तित हो जाती है, अत: परिणाम भी **float** मे प्राप्त होगा इसलिए हम इसके लिए वेरियबल **c** का टाइप **float** का चयन करते हैं।

Operator precedence (priority) & Associativity

Operator precedence: An expression in which many operators then order of operator is called priority or precedence of that operator. It means first higher rank of operator will be performed.

एक ऐसा एक्स्प्रेश्न जिसमे एक से अधिक ऑपरेटर हो तब उनके हल होने के क्रम को ही ऑपरेटर की वरीयता या प्रीसिडेंस कहते हैं। इसका अर्थ यही हुआ कि सबसे पहले वही ऑपरेटर अपना कार्य करेगा जिसकी वरीयता अन्य सभी से अधिक हो।

Ex: 5+7*2 →5+14→19 (rank of * is higher than +)

Meaning of Associativity: An expression on which many operators have same rank then evaluations will perform from left to right or right to left called associativity.

एक ऐसा एक्स्प्रेश्न जिसमे एक से अधिक ऑपरेटर एक जैसी रैंक के हो तब वे लेफ्ट से राइट या राइघ्त से लेफ्ट की ओर हल होंगे उसे ऑपरेटर की एसोसिएटिविटी कहते हैं।

Ex: 18/3*2→6*2→12 (associativity of / &* is left to right)

Priority and Associativity Chart

C language supports number operators so each has different priority and associativity.

सी भाषा मे कई सारे ऑपरेटर होते हैं इसलिए प्रत्येक की अपनी एक निश्चित वरीयता और एसोसिएटिविटी होती है।

Operator Type	Rank (Priority)	Associtivity
() Postfix ++ --	1	Left to Right
Prefix ++ -- Unary + - ! (Logical NOT) ~ (Bitwise NOT)	2	Right to Left
* (Arithmetic Multiply) / (Arithmetic Division) % (Arithmetic Modulo)	3	Left to Right
+ (Arithmetic Add) - (Arithmetic subtract)	4	Left to Right
<< (Bitwise Left Shift) >> (Bitwise Right Shift)	5	Left to Right
< (Relational Less than) <= (Relational Less than or equal) > (Relational Greater than) >= (Relational Greater than or equal)	6	Left to Right
== (Relational Equal to) != (Relational Not Equal to)	7	Left to Right
& (BitwiseAND)	8	Left to Right
^ (Bitwise XOR)	9	Left to Right
| (Bitwise OR)	10	Left to Right
&&(Logical AND)	11	Left to Right
|| (Logical OR)	12	Left to Right
?: (Conditional)	13	Right to Left
= (Assignment) += -= *= /= %= (Shorthand)	14	Right to Left

PART4

Control Statement:

Conditional Control Statement-

- if
- if-else
- else- if ladder
- nesting of if-else
- switch-case-default

Looping Control Statement

- While
- do-while
- for
- while v/s do-while
- Nesting loop

Jumping Control Statements

- break
- continue
- goto
- avoid goto.

Control Statement

At runtime, normally a statement run step by step but depends on logic we want to skips or repeat the execution of a set of steps. To handle such situations C language has following two types of control statements.

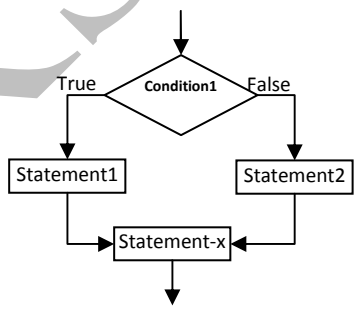
रन टाइम पर, सामान्यत: स्टेटमेंट एक के बाद एक रन होते जाते हैं किन्तु लॉजिक के आधार पर हम स्टेटमेंट के किसी समूह को रन होने से रोकना चाहते है या फिर से रन करना चाहते है। ऐसी परिस्थिति को हेंडल करने के लिए सी भाषा में निम्न सभी कंट्रोल स्टेटमेंट को निम्न दो भागों मे बांटा गया है।

1) Conditional (?:, if-else, switch-case)
2) Looping (while, do-while and for)
3) Jumping (break, continue and goto)

Conditional Control Statement:

When we have one or more sets of statement and we want to execute any one of them then such sets are written in following conditional statements.

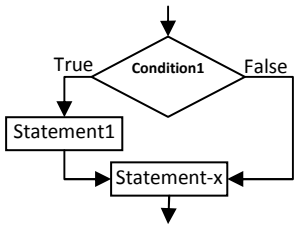
जब हमारे पास एक से अधिक स्टेमेंट के समूह है और उनमे से किसी एक को ही क्रियान्वित करवाना हो तब ऐसे स्टेटमेंट के समूहों को निम्न कंडीशनल स्टेटमेंट मे लिख सकते है।



- Using Conditional (Ternary) Operator (? :)
- Using only if keyword
- Using if-else keywords
- Using ladder form of if-else keywords
- Using nesting of if-else
- Using switch-case-default keywords

Using only if keyword:

Syntax:	Flowchart:

if(condition1)
{
Statement1
}
Statement-x
	

If **condition1 is true** then **Statement1** execute otherwise not.

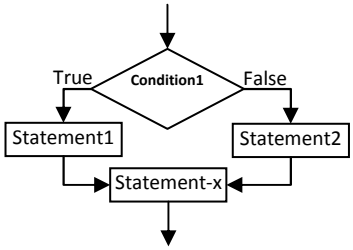
यदि **condition1** सत्य है तब **statement1** क्रियान्वित होगा अन्यथा नहीं।

Example: To convert negative to positive

```
#include<stdio.h>
void main()
{
 int a=-10;
 if(a<0)
 {
   a=-a;
 }
 printf("+ve vale is %d",a);
}
Output: +ve value is 10
```

Using if-else keyword:

Syntax:	Flowchart:

if(condition1)
{
Statement1
}
else
{
Statement2
}
Statement-x
	

If condition1 is true then Statement1 execute otherwise Statement2.

यदि **condition1** सत्य है तब **statement1** क्रियान्वित होगा अन्यथा **statement2** होगा।

Example: To find larger of two numbers:

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

```
void main()
{
 int a=10, b=15,c;
 if(a>b)
 {
   c=a;
 }
 else
 {
   c=b;
 }
 printf("Large is %d",c);
}
```

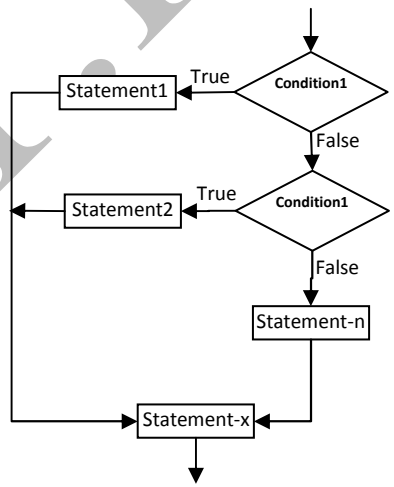
Output: Large is 15

Using ladder form of if-else keyword:

Syntax:

```
if(condition1)
{
      Statement1
}
else if(condition2)
{
       Statement2
}
::::::
      else
      {
        Statement-n
      }
Statement-x
```

Flowchart:



If **condition1 is true** then **Statement1** execute otherwise if **condition2 is true** then **Statement2** executes and this will continued. **Statement-n** executes when all above conditions are **false.**

यदि **condition1** सत्य है तब **statement1** क्रियान्वित होगा अन्यथा **condition2** के सत्य होने पर **statement2** क्रियान्वित होगा और यह क्रम निरंतर हो सकता है। जब सभी **condition** असत्य हो जाती है तब **statement-n** क्रियान्वित होगा।

Example: To print grade of percentage.

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

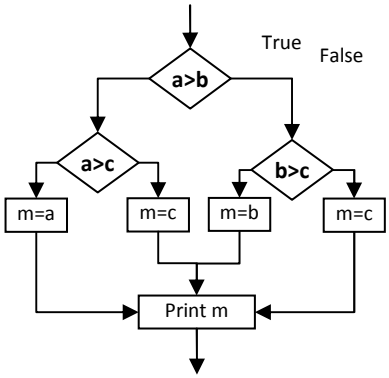
```
void main()
{
 int p=55;
 char grade;
 if(p>=60)
 {
   grade='A';
 }
 else if(p>=50)
 {
   grade='B';
 }
 else if(p>=40)
 {
  grade='C';
 }
 else
 {
  grade='D';
 }
 printf("Grade: %c", grade);
}
```

Output: Grade:B

Using nesting form of if-else

```
Syntax:
   if(condition1)
  {
    if(condition2)
    {
      Statement1
    }
    else
    {
      Statement2
    }
  }
  else
  {
    if(condition3)
      {
        Statement3
      }
    else
      {
        Statement4
      }
   }
 Statement-x
```

Flowchart:



If Condition1 is true then Condition2 checks otherwise Condition3 checks.

यदि **condition1** सत्य है तब **condition2** चेक होगी अन्यथा **condition3** चेक होगी।

Example: To find maximum of three numbers:

```
#include<stdio.h>
void main()
{
   int a=10, b=15,c=7,m;
   if(a>b)
   {
        if(a>c)
        {
             m=a;
        }
        else
        {
             m=b;
        }
   }
   else
   {
        if(b>c)
        {
             m=b;
        }
        else
        {
             m=c;
        }
   }
 printf("Large number is %d",m);
}
```

Output: Large number is 15

Using switch-case-default keywords:

Suppose we have a number of statement sets and everyone is labeled by unique integer or character value. We input any value from outsource so that it can matches to any labels. When any first one label is matched then their statements will be executes otherwise default statements executes. Switch-case-default structure is used mainly for menu driven / choice based programming.

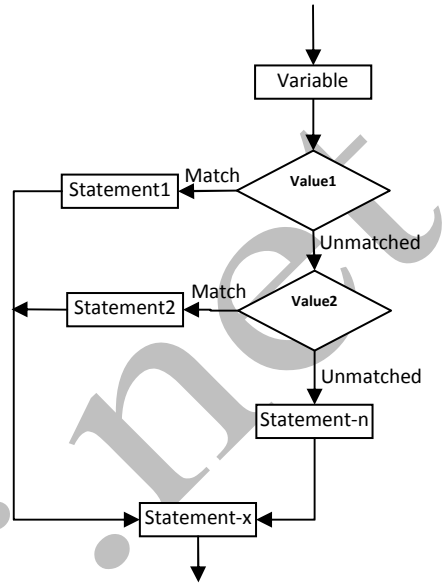
मानाकि हमारे लॉजिक मे कई सारे स्टेटमेंट के समूह हैं और प्रत्येक किसी अद्वितीय इंटीजर या कैरक्टर वैल्यू से लेबल्ड हैं। हम किसी वैल्यू को बाहर से इनपुट करते हैं

जिससे कि वह किसी लेबल से मैच हो सके। जब कोई पहला लेबल मैच हो जाता है उसके स्टेटमेंट क्रियान्वित हो जाएंगे अन्यथा डिफ़ाल्ट स्टेटमेंट क्रियान्वित होंगे। switch-case-default स्ट्रक्चर का उपयोग मुख्यत: मेनू ड्रिवेन / चॉइस बेस्ड प्रोग्रामिंग मे किया जाता है।

Syntax:

```
switch(variable/expression)
{
 case value1:
          statement1
          break;
 case value2:
          statement2
          break;
 …….
 …….
 case value-n:
          statement-n;
          break;
 default:
          default statement;
}
```

Flowchart:



Example: Arithmetic calculator.

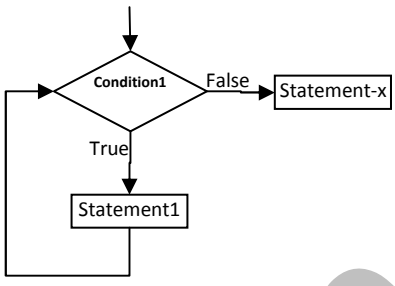
```
#include<stdio.h>
void main()
{
 int a=20,b=15;
 char ch='*';
 switch(ch)
 {
  case '+':
           printf("Sum=%d",a+b);
           break;
  case '-':
           printf("Subtract=%d",a-b);
           break;
  case '*':
           printf("Multiply=%d",a*b);
           break;
  case '/':
           printf("Division=%d",a/b);
           break;
  default :
          printf("\nInvalid operator");
 }
 }
```

Output: 5

Looping Control Statement:

When we have to execute one set of statements more than one times then following looping control statements can be used in C language.

जब हमे किसी एक समूह के स्टेटमेंट को एक से अधिक बार क्रियान्वित करवाना हो तब सी भाषा में निम्न लूपिंग कंट्रोल स्टेटमेंट का उपयोग करते हैं।



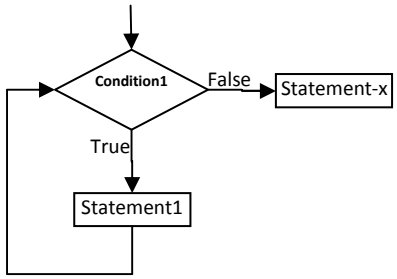
- Using **while** keyword
- Using **do-while** keyword
- Using **for** keyword

Using while keyword:

Syntax:

```
while(condition1)
{
    Statement1
}
Statement-x
```

Flowchart:



While is entry level and undefined loop structure. Initially condition1 is checked. If it true then Statement1 executes and again condition1 is checked. This process of execution will be continued until condition1 gives false value.

while एक एंट्री लेवेल और अंडिफ़ाइंड लूप स्ट्रक्चर है। इसमे सबसे पहले **condition1** चेक होती है। यदि यह सत्य है तब ही **statement1** क्रियान्वित होता है और फिर से **condition1** चेक की जाती है। यह प्रक्रिया निरंतर बनी रहती है जब तक की **condition1** असत्य नहीं हो जाती।

Example: Show your name until you want.

```
#include<stdio.h>
void main()
{
 char a='y';
 while(a=='y' || a=='Y')
 {
   printf("\nDownload Ebooks: www.LRsir.net");
   printf("\nPress y key to continue..");
   fflush(stdin);   //to clean input buffer
   a=getche();
 }
}
```

Output:

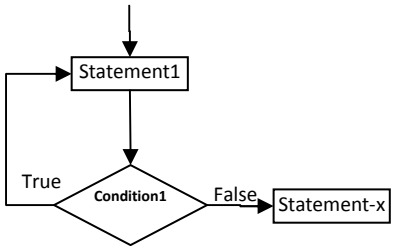
```
Download Ebooks: www.LRsir.net
Press y key to continue..y
Download Ebooks: www.LRsir.net
Press y key to continue..y
Download Ebooks: www.LRsir.net
Press y key to continue..n
<Stop loop>
```

Using do-while keyword:

Syntax:

```
do
{
    Statement1
}while(condition1)
Statement-x
```

Flowchart:



do-while is exit level and undefined loop structure. Initially loop statement1 executes then condition1 is checked. If it true then once again statement1 executes and again condition1 is checked. This process of execution will be continued until condition1 gives false value.

do-while एक एक्ज़िट लेवेल और अंडिफ़ाइंड लूप स्ट्रक्चर है। इसमे सबसे पहले **statement1** क्रियान्वित होता है उसके बाद **condition1** चेक होती है। यदि यह सत्य है तब ही **statement1** फिर से क्रियान्वित होता है और एक बार फिर से **condition1** चेक होती है। यह प्रक्रिया निरंतर बनी रहती है जब तक की **condition1** असत्य नहीं हो जाती।

Example: Show your name until you want.

```
#include<stdio.h>
void main()
{
 char a;
 do
 {
   printf("\nDownload Ebooks: www.LRsir.net");
   printf("\nPress y key to continue..");
   fflush(stdin);   //to clean input buffer
   a=getche();
 }while(a=='y' || a=='Y')
}
```

Output:

```
Download Ebooks: www.LRsir.net
Press y key to continue..y
Download Ebooks: www.LRsir.net
Press y key to continue..y
Download Ebooks: www.LRsir.net
Press y key to continue..n
<Stop loop>
```

while v/s do-while loop: When initially condition1 is false then in case of while loop, statement1 never execute whereas in case of do-while loop, statement1 will executes first time.

जब प्रारम्भ में ही **condition1** असत्य हो जाती है तब **while** लूप की स्थिति मे **statement1** एक बार भी क्रियान्वित नहीं होता है जबकि **do-while** लूप की स्थिति मे स्टेटमेंट एक बार अवश्य क्रियान्वित हो जाता है।

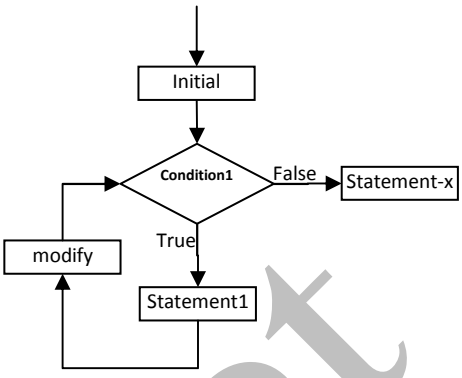
Example:while loop	Example:do-while loop

`#include<stdio.h>`
`void main()`
`{`
`char a='n';`
`while(a=='y'||a=='Y')`
`{`
`printf("I never run");`
`}`
`}`
`Output: Nil`
	`#include<stdio.h>`
	`void main()`
	`{`
	`char a='n';`
	`while(a=='y'||a=='Y')`
	`{`
	`printf("I run one time");`
	`}while(`
	`}`
	`Output: I run one time`

Using for keyword:

```
Syntax:
for(initial;condition1;modify)
{
   Statement1
}
Statement-x
```

Flowchart:



for is entry level and finite loop structure. for loop is consist using three parts. First part set initial value, second part check condition1 and third part modify value. If condition1 is true then Statement1 executes, after then modify part executes then again condition1 is checked. This process of execution will be continued until condition1 gives false value.

for एक एंट्री लेवल और फायनाइट लूप स्ट्रक्चर है। **for** लूप तीन भागों से मिलकर बना होता है। पहला भाग लूप की प्रारम्भिक वैल्यू को सेट करता है, दूसरा भाग **condition1** को चेक करता है और तीसरा भाग वैल्यू में बदलाव करता है। यदि **condition1** सत्य है तब ही **statement1** क्रियान्वित होता है, इसके बाद मोड़िफाइ भाग क्रियान्वित होगा, उसके बाद **condition1** फिर से चेक होगी। क्रियान्वयन की यह प्रक्रिया निरंतर बनी रहती है जब तक की **condition1** असत्य नहीं हो जाती।

Example: Show your name 5 times.

```
#include<stdio.h>
void main()
{
 int i;
 for(i=1;i<=10;i++)
 {
   printf("\nDownload Ebooks: www.LRsir.net");
 }
}
```

Output:

```
Download Ebooks: www.LRsir.net
Download Ebooks: www.LRsir.net
Download Ebooks: www.LRsir.net
Download Ebooks: www.LRsir.net
Download Ebooks: www.LRsir.net
<Stop loop>
```

Nesting loop:

When one loop structure is defined inside a loop structure then it is called nesting of loop. We can create nesting loop in multiple ways.

जब एक लूप स्ट्रक्चर को किसी अन्य लूप स्ट्रक्चर मे परिभाषित करते हैं तब इसे लूप की नेस्टिंग कहा जाता है। लूप की नेस्टिंग को कई प्रकार से निर्मित कर सकते हैं जैसे-

1) while loop inside while loop
2) do-while loop inside do-while loop
3) for loop inside for loop
4) while loop inside for or do-while loop
5) for loop inside while or do-while loop
6) do-while loop inside for or while loop

We can also increase level of nesting loop up to any desired level.

लूप कि नेस्टिंग का स्तर आवश्यकतानुसार बड़ाया भी जा सकता है।

```
Example: print multiplication table from 1 to 10 using
Nesting of for loop
#include<stdio.h>
void main()
{
 int i,j;
 for(i=1;i<=10;i++)
 {
  printf("\n");
  for(j=1;j<=10;j++)
  {
    printf("%d\t",i*j);
  }
 }
}
Output:
1   2   3   4   5 ...10
2   4   6   8   10...20
========
========
10  20  30  40  50...100
```

Loop from i=1 to 10 called outer loop where as loop from j=1 to 10 called inner loop. When condition of outer loop is true then condition of inner loop checked. When condition of inner loop is false then condition of outer loop

will again checked. This process will be continued until condition of outer loop becomes false.

i=1 से 10 तक आउटर लूप और **j=1** से 10 तक ईनर लूप कहलाता है। जब आउटर लूप की कंडिशन सत्य होती है तब ही ईनर लूप की कंडिशन चेक होती है। और जब ईनर लूप की कंडिशन असत्य होती है तब ही आउटर लूप की कंडिशन चेक होती है। यह प्रक्रिया निरंतर बनी रहती है जब तक कि आउटर लूप कि कंडिशन असत्य नहीं हो जाती।

Jumping Control Statements (break, continue and goto)

A statement that can be used to transfer program control to anywhere in program so that execution may start from that point called jumping control statement.

एक ऐसा स्टेटमेंट जो प्रोग्राम कंट्रोल को प्रोग्राम में किसी अन्यत्र सेट कर सके जिससे कि क्रियान्वयन उस बिन्दु से प्रारम्भ हो सके, ऐसे स्टेटमेंट को जंपिंग कंट्रोल स्टेटमेंट कहते है।

In C language jumping of program control can be take place using following three keywords.

सी भाषा में कंट्रोल की जंपिंग को निम्न तीन कीवर्ड की सहायता से सम्पन्न किया जाता है।

1) Using break keyword
2) Using continue keyword
3) Using goto keyword

Using break keyword:

It is applicable inside switch-case-default and loop structure (while, do-while and for). When break executes then program control comes out immediately from that structure and set at the bottom of structure. Statements after break never execute.

break का उपयोग **switch-case-default** एवं लूप स्ट्रक्चर(**while, do-while** एवं **for)** में ही किया जा सकता है। जब **break** क्रियान्वित होता है तब प्रोग्राम कंट्रोल उसी समय उस स्ट्रक्चर से बाहर आ जाता है और स्ट्रक्चर के नीचे सेट हो जाता है। **break** के क्रियान्वित होने के बाद शेष स्टेटमेंट क्रियान्वित नहीं होते है।

Syntax:

```
break;
```

break should be used conditionally.

ब्रेक का उपयोग कंडिशन के साथ किया जाना चाहिए।

```
Example: find input number is prime or not.
#include<stdio.h>
void main()
{
  int n,r,i;
  printf("\nInput any integer number:");
  scanf("%d",&n);
  for(i=2; i<n; i++)
  {
     r=n%i;
     if(r==0)
     {
          printf("\n%d is not prime number",n);
          break;
     }
  }
  if(i==n)
  {
     printf("\n%d is prime number",n);
  }
 }
Output:
Input integer number: 9<enter>
9 is not prime number
Input integer number: 5<enter>
5 is prime number
```

In this example, when break executes conditionally then program controller comes out from for loop before false value of loop condition.

इस प्रोग्राम मे, जब **break** दी गई कंडिशन के सत्य होने पर क्रियान्वित होता है और लूप की कंडिशन के असत्य होने से पहले ही प्रोग्राम कंट्रोलर को लूप से बाहर कर देता है।

Using continue keyword:

It is applicable inside loop structure (while, do-while and for). When continue executes then program control transfer to beginning of loop so that condition can be checked or modify value in for loop. Statements after continue never execute.

continue का उपयोग लूप स्ट्रक्चर (**while, do-while** एवं **for)** में ही किया जा सकता है। जब **continue** क्रियान्वित होता है तब प्रोग्राम कंट्रोल उसी समय लूप के प्रारम्भ में चला जाता है जिससे कि लूप की कंडिशन पुन: चेक हो या **for** लूप मे वैल्यू मोड़ीफ़ाई हो जाए। **continue** के क्रियान्वित होने के बाद शेष स्टेटमेंट क्रियान्वित नहीं होते है।

Syntax:

```
continue;
```

continue should be used conditionally.

continue का उपयोग कंडिशन के साथ किया जाना चाहिए।

```
Example: print odd number from 1 to 10.
#include<stdio.h>
void main()
{
  int i,r;
  for(i=1; i<=10; i++)
  {
     r=i%2;
     if(r==0)
     {
          continue;
     }
     printf("%d\t",i);
  }
 }
Output:1   3    5    7    9
```

In this example, when continue executes conditionally then program controller transfer to beginning of for loop before executing remaining statement.

इस प्रोग्राम मे, जब **continue** दी गई कंडिशन के सत्य होने पर क्रियान्वित होता है और प्रोग्राम कंट्रोलर लूप के शेष स्टेटमेंट को क्रियान्वित करने से पहले ही लूप के प्रारम्भ में चला जाता है।

Using goto keyword:

It is applicable inside function. When goto executes then program control immediately transfer to a point where label of goto is set and execution begins from that point.

goto का उपयोग फंक्शन में किया जाता है। जब **goto** क्रियान्वित होता है तब प्रोग्राम कंट्रोल उसी समय वहाँ चला जाता है जहाँ **goto** के साथ लिखा गया लेबल परिभाषित है। इसके बाद क्रियन्वयन उस बिन्दु से शुरू हो जाता है।

Syntax:

To skip execution of code:	**To repeat execution of code:**
goto label; statement1 label: statement-x	label: statement-1 goto label;

goto should be used conditionally.

goto का उपयोग कंडिशन के साथ किया जाना चाहिए।

Example: print number from 1 to 10 using goto.

```
#include<stdio.h>
void main()
{
  int i=1;
  X: printf("%d\t",i);
  if(i<=10)
  {
     goto X;
  }
 }
```

Output: 1 2 3 4 5 6 7 8 9 10

In this example, X is a label. When goto executes conditionally then program controller transfer to X:

इस प्रोग्राम मे, **X** एक लेबल है। जब **goto** कंडिशन के सत्य होने पर क्रियान्वित होता है तब प्रोग्राम का कंट्रोलर लेबल **X:** पर ट्रान्सफर हो जाता है।

Avoid use of goto:

We can write a number of goto statements with same label name or different label name but we must remember that one label must be defined only one times because it may causes ambiguity error. Since size of program grows then remembering flow of control using goto becomes very difficult for programmer. Therefore goto should be used in rear cases like to come out from innermost loop to outermost loop because break comes out from inner loop but not outer loop.

प्रोग्राम में हम आवश्यकतानुसार कई सारे **goto** स्टेटमेंट को एक ही लेबल नाम या भिन्न नाम के लेबल के साथ लिख सकते है। किन्तु हमे इस बात का अनिवार्यतः ध्यान रखना होता है कि, एक नाम का लेबल केवल एक ही बार परिभाषित हो अन्यथा एम्बिज्युटी एरर प्राप्त हो सकती है। चूंकि प्रोग्राम की साइज़ बड़ने पर प्रोग्रामर के लिए **goto** स्टेटमेंट के फ्लो को याद रखना बहुत कठिन हो जाता है। इसलिए **goto** का उपयोग प्रोग्राम मे केवल विशेष परिस्थितियों में ही किया जाना चाहिए जैसे प्रोग्राम कंट्रोलर को सबसे इनरमोस्ट लूप से सीधे आउटरमोस्ट लूप से बाहर करना हो, क्यूंकी **break** केवल इनर से बाहर कर सकता है।

PART5

Array:

One, two, Multi dimensional

String:

accessing string using %s

gets() and puts()

String manipulation function

Structure

Array of Structure

Structure within structure

Union

Enumeration.

Array: (One, two and Multi dimensional)

Collection of same data arranged in continuous order called array. Array is one example of derived data type of non primitive class. Array has following types with example.

एक समान डाटा के ऐसे समूह जो निरंतर क्रम मे व्यवस्थित हों उन्हे अरे कहा जाता है। अरे एक नॉन प्रीमिटिव डिराईव्ड डाटा टाइप है। अरे 3 प्रकार का होता है जो उधारण सहित निम्न है।

For example:

One Dimensional	Two dimensional	Multi dimensional
10 20 30 40 50	10 20 30 40 50 60 70 80 90	5 10 15 20 25 30 35 40 45 10 20 30 40 50 60 70 80 90

One dimensional array:

Collections of same data arrange in single row called one dimensional array. एक समान डाटा के ऐसे समूह जो केवल एक ही **row** मे व्यवस्थित हों उन्हे एक विमीय अरे कहा जाता है।

Syntax:

```
array-type array-name[size];
```

array-type = any data type like int / float / char

Example:

array of 5 integer: `int a[5];`

array of 5 float: `float b[5];`

Initialization of array:

```
int a[5]={10,20,30,40,50}; or
int a[]={10,20,30,40,50};
```

Size automatically set by counting number of values.

वैल्यू की संख्या के अनुसार साइज़ स्वत: सेट हो जाती है।

Accessing array data: (Read / Write data)

Syntax:

array-name[i]

where i = index of array (0 to size-1)

0 is lower bound of array.

(size-1) is upper bound of array.

Example: read and show 5 integer data with maximum value

```
#include<stdio.h>
void main()
{
 int a[5];
 int max,i;
 printf("\nInput 5 integer data:");
 for(i=0; i<5; i++)
 {
   scanf("%d", &a[i]);
 }
printf("\nArray data are:");
for(i=0; i<5; i++)
 {
   printf("%d\t", a[i]);
 }
max=0;
for(i=0; i<5; i++)
 {
   if(max<a[i])
   {
     max=a[i];
   }
  }
  printf("\nMaximum value=%d", max);
}
Output:
Input 5 integer data:    10    5    6    15    9<enter>
Array data are: 10  5    6    15    9
Maximum value=15
```

Memory Allocation:

a[0]	a[1]	a[2]	a[3]	a[4]
10	5	6	15	9

From this program it is clear that we can use for loop to access every data of one dimensional array.

इस प्रोग्राम से स्पस्ट है कि हम एक विमीय अरे के प्रत्येक डाटा को एक्सैस करने के लिए **for** लूप का उपयोग कर सकते है।

Two dimensional array:

Collections of same data arrange in tabular / matrix / grid form that contain multiple rows and each row contain equal columns called two dimensional arrays.

एक समान डाटा के ऐसे समूह जो टेबुलर / मेट्रीक्स / ग्रिड के रूप मे व्यवस्थित हों अर्थात जिसमें डाटा कई सारी **rows** हों और प्रत्येक **row** मे एक कॉलम की संख्या एक समान हो, ऐसे अरे को द्वि-विमीय अरे कहा जाता है।

Syntax:

```
array-type array-name[rows][columns];
```

array-type = any data type like int / float / char
rows = maximum rows
columns = maximum columns

Example:

array of 2x3 integer: `int m[2][3];`
array of 3x3 float: `float b[3][3];`

Initialization of array:

```
int m[2][3]={{10,20,30},{40,50,60}}; or
int m[][]={{10,20,30},{40,50,60}};
```

Size automatically set by counting number of values.
वैल्यू की संख्या के अनुसार साइज़ स्वत: सेट हो जाती है।

Accessing array data: (Read / Write data)

Syntax:

```
array-name[i][j]
```

where-

i = index of row (0 to rows-1)
j = index of column (0 to columns-1)

[0][0] is lower bound, [rows-1][columns-1] is upper bound o

Example: read 3x4 matrix and show its transpose.

```
#include<stdio.h>
void main()
{
 int m[3][4];
 int i,j;
 for(i=0; i<3; i++)
 {
```

```
      printf("\nInput 4 integer data row-wise:");
      for(j=0; j<4; j++)
      {
            scanf("%d", &m[i][j]);
      }
  }
printf("\nTranspose of matrix=");
for(i=0; i<4; i++)
 {
      printf("\n");
      for(j=0; j<3; j++)
      {
            printf("%d", m[i][j]);
      }
  }
Output:
Input 4 integer data row-wise:10    20  30   40<enter>
Input 4 integer data row-wise:50    60  70   80<enter>
Input 4 integer data row-wise:90    100 110  120<enter>
Transpose of matrix=
10    50    90
20    60    100
30    70    110
40    80    120
```

Memory Allocation:

m[0][0]	m[0][1]	m[0][2]	m[0][3]
m[1][0]	m[1][1]	m[1][2]	m[1][3]
m[2][1]	m[2][2]	m[2][2]	m[2][3]

From this program it is clear that we can use for loop to access every data of one dimensional array.

इस प्रोग्राम से स्पस्ट है कि हम एक विमीय अरे के प्रत्येक डाटा को एक्सैस करने के लिए **for** लूप का उपयोग कर सकते है।

Multidimensional array:

Collection of two dimensional array called three or multi dimensional array.

एक समान द्वि विमीय अरे के समूह को त्रि विमीय या बहू विमीय अरे कहते हैं।

Example:

```
int cube[3][3][3];
```

Initialization of array:

```
int cube[2][2][2]=
{{{10,20},{30,40}},{{10,20},{30,40}}};
```

String & string manipulation function:

Array of characters terminated by NULL(/0) is called string. String is written in "---". String is used to store information.

कैरक्टर्स का अरे जिसके अंत मे **NULL (/0)** हो उसे स्ट्रिंग कहते है। स्ट्रिंग को "---" मार्क मे लिखते हैं। स्ट्रिंग का उपयोग जानकारियों को स्टोर करने के लिए करते हैं।

Syntax:

```
char str-name[size];
```

Example:

```
char fname[15];
char lname[20];
```

Initialization of string:

```
char str[]={'L','R','s','i','r','\0'}; or
char str[]="LRsir";
```

Size automatically set by counting number of characters.

कैरक्टर्स की संख्या के अनुसार साइज़ स्वत: सेट हो जाती है।

Accessing string: (Read / Write)

1) **Using %s in printf & scanf:**

```
#include<stdio.h>
void main()
{
 char fname[15];
 printf("\nInput your name:");
 scanf("%s", fname);
 printf("\nYour name is %s", fname);
}
Output: Input your name: LRsir
Your name is LRsir
```

Remark: & mark never used and %s read or show only single word.

2) **Using gets() and puts():**

```
#include<stdio.h>
void main()
{
 char fname[15];
 puts("\nInput your name:");
 gets(fname);
 puts("\nYour name is");
 puts(fname);
```

```
}
Output: Input your name: LRsir
Your name is LRsir
```

Remark: gets() and puts() can read or show more words.

String manipulation function:

We can perform a number of operations on string using following library functions. Include string.h header file before using them.

निम्न फंक्शन्स के द्वारा स्ट्रिंग पर विभिन्न ऑपरेशन किए जा सकते हैं। उपयोग से पहले **string.h** हैडर फ़ाइल को शामिल करना अनिवार्य है।

1) **strlen(str):** returns number of characters
   यह स्ट्रिंग मे कुल कैरक्टर की संख्या को प्रदान करता है।
2) **strlwr(str):** converts all upper case into lower case
   यह स्ट्रिंग के बड़े अक्षरों को छोटे अक्षरों मे बदल देता है।
3) **strupr(str):** converts all lower case into upper case
   यह स्ट्रिंग के छोटे अक्षरों को बड़े अक्षरों मे बदल देता है।
4) **strcpy(str2,str1):** To copy str1 into str2.
   str1 को **str2** मे कॉपी करने के लिए।
5) **strcmp(str1,str2):** To compare str1 and str2. If equal then return 0 value.
   स्ट्रिंग **str1** और **str2** की तुलना करने के लिए। एक जैसी होने पर यह 0 वैल्यू प्रदान करता है।
6) **strcat(str1,str2):** To merge str2 at the end of str1.
   यह स्ट्रिंग **str2** को **str1** के अंत मे जोड़ सकती है।

```
Example: find length and copy string
#include<stdio.h>
#include<string.h>
void main()
{
 char str1[]="www.LRsir.net",str2;
 int size;
 size=strlen(str1);
 printf("\nSize of string=%d",size);
 strcpy(str2,str1);
 puts(str2);
}
Output: Size of string=9
LRsir.net
```

Structure:

Continuous collections of same or different data called structure. Structure is one example of user defined data type of non primitive class.
एक समान अथवा अलग-अलग प्रकार के क्रमश: डाटा के समूह को स्ट्रक्चर कहते हैं। स्ट्रक्चर एक नॉन प्रीमिटिव यूसर डिफ़ाइंड डाटा टाइप है।

Working with structure: Need three syntaxes.

Syntax1: Creating structure template

```
struct sname
{
      type member1;
      type member2;
      ============
      type member-n;
};
```

Syntax2: Declare structure variable

```
struct sname svar1, svar2,…,svar-n;
```

Syntax3: Accessing member: Using dot (.) period/membership operator

```
svar.member-name
```

svar means svar1, svar2, ..,svar-n

member-name means member1, member2,member-n.

Initialization of structure:

```
struct student
{
      int id;
      char name[15];
      float fee;
}std={101,"Rahul",1500.5};
Or
struct student std={101,"Rahul",1500.5};
```

Memory allocation and size of structure:

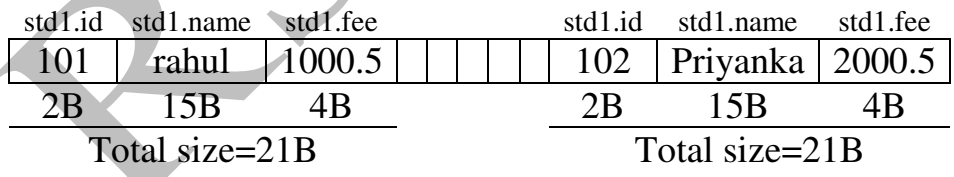
Separate memory spaces allocated to every data members and size of one structure variable is total sum of size of every data members.

स्ट्रक्चर के प्रत्येक डाटा मेम्बर्स के लिए प्रथक-प्रथक मेमोरी स्पेस आवंटित होता है और एक स्ट्रक्चर वेरियबल की साइज़ उसके प्रत्येक डाटा मेम्बर्स की कुल साइज़ के योग के बराबर होती है।

Example: read and show two students record.

```
#include<stdio.h>
void main()
{
      struct student
      {
            int id;
            char name[15];
            float fee;
      };
      struct student std1,std2;
      printf("\nInput id, name and fee:");
      printf("\nFor first record:");
      scanf("%d%s%f",&std1.id,std1.name,&std1.fee);
      printf("\nFor second record:");
      scanf("%d%s%f",&std2.id,std2.name,&std2.fee);
      printf("\nid\tname\tfee");
      printf("\n%d\t%s\t%f",std1.id,std1.name,std1.fee);
      printf("\n%d\t%s\t%f",std2.id,std2.name,std2.fee);
}
Output:
Input id, name and fee:
For first record:101     rahul     1000.5<enter>
For second record:102    priyanka  2000.5<enter>
id    name       fee
101   rahul      1000.5
102   priyanka   2000.5
```

Memory Allocation:



From this program it is clear that separate memory space allocated to every data members and size is sum of sizes of each data member.

इस प्रोग्राम से स्पस्ट है कि प्रत्येक डाटा मेम्बर्स के लिए अलग अलग स्पेस होता है और एक स्ट्रक्चर की साइज़ प्रत्येक डाटा मेम्बर्स की साइज़ के योग के बराबर होती है।

Array of Structure:

Continuous collections of same structure data type called array of structure. For example records of 10 students, list of players.

एक समान स्ट्रक्चर डाटा टाइप के निरंतर कलेक्शन को स्ट्रक्चर का अरे कहते हैं। जैसे 10 स्टूडेंट्स का रेकॉर्ड, 11 प्लेयर्स की लिस्ट आदि।

Working with array of structure: Need three syntaxes.

Syntax1: Creating structure template

```
struct sname
{
     type member1;
     type member2;
     ============
     type member-n;
};
```

Syntax2: Declare array of structure

```
struct sname sarr[size];
```

Syntax3: Accessing member: Using dot (.) period/membership operator

```
sarr[i].member-name
```

```
sarr[i] means sarr[0] to sarr[size-1]
```

```
member-name means member1, member2,member-n.
```

Memory allocation :

Continuous memory spaces allocated to all structures of array.

अरे के सभी स्ट्रक्चर को लगातार मेमोरी स्पेस आवंटित होता है।

Example: read and show two students record.

```
#include<stdio.h>
void main()
{
     struct student
     {
          int id;
          char name[15];
          float fee;
     };
     struct student std[5];
     int i;
     for(i=0; i<size;i++)
     {
          printf("\nInput id, name and fee:");
          scanf("%d%s%f",&std[i].id,std[i].name,&std[i].fee);

     }
     printf("\nStudent records:");
     printf("\nid\tname\tfee");
     for{i=0;i<size;i++)
     {
     printf("\n%d\t%s\t%f",std[i].id,std[i].name,std[i].fee);
     }
}
```

```
Output:
Input id, name and fee:101    rahul     1000.5<enter>
Input id, name and fee:102    nilesh    2000.5<enter>
Input id, name and fee:103    mukesh    3000.5<enter>
Input id, name and fee:104    rakesh    4000.5<enter>
Input id, name and fee:105    ritesh    5000.5<enter>
Student Records
id    name       fee
101   rahul      1000.5
102   nilesh     2000.5
103   mukesh     3000.5
104   rakesh     4000.5
105   ritesh     5000.5
```

Memory Allocation:

std[0].id	std[0].name	std[0].fee						std[4].id	std[4].name	std[4].fee
101	rahul	1000.5						105	ritesh	5000.5

From this program it is clear that we can create and access a list of records of same type using array of structure.

इस प्रोग्राम से स्पस्ट है कि स्ट्रक्चर का अरे बनाकर एक ही प्रकार के कई सारे रेकॉर्ड्स को एक्सैस किया जा सकता है।

Structure within structure :(nesting of structure)

When a structure variable is member of another structure then it is called structure within structure.

जब एक स्ट्रक्चर का वेरियबल अन्य स्ट्रक्चर का मेम्बर हों तब इसे स्ट्रक्चर का स्ट्रक्चर कहते हैं

Syntax1: Creating structure template

```
struct sname
{
      type member1;
      type member2;
      ============
      type member-n;
};
struct ssname
{
      struct sname svar;
      ============
      ============
};
```

Syntax2: Declare variable of nesting structure

```
struct ssname ssvar;
```

Syntax3: Accessing member: use dot (.) operator two times

```
ssvar.svar.member-name
```

Example: read and show two students record.

```
#include<stdio.h>
void main()
{
     struct address
     {
      int hno,
      char colony[10];
      char city[10];
     };
     struct student
     {
          int id;
          char name[15];
          float fee;
          struct address la,pa;
     };
     struct student std;
     printf("\nInput id, name and fee:");
     scanf("%d%s%f",&std.id,std.name,&std.fee);
     printf("\nlocal:Input hno, colony and city");
     scanf("%d%s%s",&std.la.hno,std.la.colony,std.la.city);
     printf("\nparmanent:Input hno, colony and city");
     scanf("%d%s%s",&std.pa.hno,std.pa.colony,std.pa.city);

     printf("\nid\tname\tfee");
     printf("\n%d\t%s\t%f",std.id,std.name,std.fee);
     printf("\nLocal Address:");
     printf("\n%d,%s,%s",std.la.hno,std.la.colony,std.la.city);
     printf("\nParmanent Address:");
     printf("\n%d,%s,%s",std.pa.hno,std.pa.colony,std.pa.city);
}
Output:
Input id, name and fee:
101  rahul     1000.5<enter>
local:Input hno, colony and city
11   rishinagar     Ujjain<enter>
parmanent:Input hno, colony and city
233  chanakypuri    dilli<enter>
id   name      fee
101  rahul     1000.5
Local Address:
11,rishinagar,Ujjain
Parmanent address:
233,chanakypuri,Ujjain
```

Union:

It also shows continuous collections of same or different data but a common memory space is available to all data of union. Union is one example of user defined data type of non primitive class.

Union भी एक समान अथवा अलग-अलग प्रकार के क्रमश: डाटा के समूह को व्यक्त तो करता है किन्तु यूनियन के सभी डाटा के लिए एक ही कॉमन मेमोरी स्पेस उपलब्ध होता है। यूनियन एक नॉन प्रीमिटिव यूसर डिफ़ाइंड डाटा टाइप है।

Working with union: Need three syntaxes.

Syntax1: Creating union template

```
union sname
{
      type member1;
      type member2;
      ============
      type member-n;
};
```

Syntax2: Declare union variable

```
union uname uvar1, uvar2,…,uvar-n;
```

Syntax3: Accessing member: Using dot (.) period/membership operator

```
uvar.member-name
uvar means uvar1, uvar2, ..,uvar-n
member-name means member1, member2,member-n.
```

Memory allocation and size of union:

A common memory space is allocated to every data members therefore last one data assigns into accurate form. Size of one union variable is maximum size of any data member.

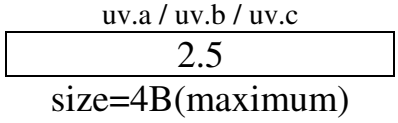
यूनियन के प्रत्येक डाटा मेम्बर्स के लिए एक ही मेमोरी स्पेस आवंटित होता है अत: केवल अंतिम असाइन वैल्यू ही शुद्ध रूप से स्टोर रहता है। एक यूनियन वेरियबल की साइज़ उसके सबसे बड़े डाटा मेम्बर की साइज़ के बराबर होती है।

Example: read and show three different values using union.

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

```
      union cif
      {
            char a;
            int b;
            float c;
      };
      union cif uv;
      printf("\nInput character, integer and float value:");
      scanf("%c%d%f",&uv.c,&uv.i,&uv.f);
      printf("\ncharacter\tinteger\tfloat");
      printf("\n%c\t%d\t%f",uv.a,uv.b,uv.c);
}
Output:
Input character, integer and float value: x 11 2.5<enter>
Character integer   float
G         G         2.5
G means garbach value.
```

Memory Allocation:



From this program it is clear that a common memory space allocated to every data members therefore only last assign value will be stores and size is equal to float members because its size is maximum(4B).

इस प्रोग्राम से स्पस्ट है कि प्रत्येक डाटा मेम्बर्स के लिए एक ही कॉमन मेमोरी स्पेस उपलब्ध रहता है इसलिए एक यूनियन वेरियबल में अंतिम बार असाइन किया गया डाटा ही बना रहता है। एक यूनियन की साइज़ सबसे बड़े डाटा मेम्बर जैसे **float** की साइज़(4**B)** के बराबर होती है।

Enumeration:

It is one example of user defined data type of non primitive class. It is used to specify word for integer constant. When we use such enumerated word in program then at run time it will replace by respected integer value.

यह भी एक नॉन प्रीमिटिव यूसर डिफ़ाइंड डाटा टाइप है। इसका उपयोग इंटीजर कोंस्टंट को किसी शब्द से परिभाषित करने के लिए करते हैं। जब ऐसे एनुमरेटेड शब्द का उपयोग प्रोग्राम कोड मे करते है तब रन टाइम पर वह संबधित इंटीजर वैल्यू से रिप्लेस हो जाती है।

Example: read integer value and show predefine day name.

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

```
void main()
{
      enum week
      {
            sun,mon,tue,wed,thus,fri,sat
      };
      enum week day;
      printf("\nInput integer value from 0 to 6:");
      scanf("%d",&day);
      switch(day)
      {
       case sun:
            printf("Sunday");
            break;
       case mon:
            printf("Monday");
            break;
       case tue:
            printf("Tuesday");
            break;
       case wed:
            printf("Wednesday");
            break;
       case thus:
            printf("Thursday");
            break;
       case fri:
            printf("Friday");
            break;
       case sat:
            printf("Saturday");
            break;
       default:
            printf("\nInvalid");
      }
 }
Output:
Input integer value from 0 to 6: 2 <enter>
Tuesday
```

In this program sun has 0, mon has 1 and so on values are assigned.

इस प्रोग्राम में **sun** की 0, **mon** की 1 एवं क्रमश: वैल्यू असाइन होती जाती है।

PART6

Function:
Need of function
Types of function
Library function
User defines function
Function Definition
A return statement
Function Declaration
Function calling
Function terminology:
    Calling function v/s Called function
    Actual argument v/s Formal arguments
Function calling technique:
    Call by value or pass by value
    Call by reference or pass by reference
Recursion
Command line argument
Local scope v/s Global scope
Lifetime of a variable
Static variable
Storage class
Identifier, Modifier, Qualifier, sizeof, typedef.

Function:

A block that contains some logic code called function.
एक ऐसा ब्लॉक जिसमें लॉजिक कोड होते हैं उसे फंक्शन कहा जाता है।
Ex: printf(), scanf(), clrscr(), getch() etc.

Need of function:

1) To avoid rewriting repetitive codes many times.
   दोहराए जाने वाले कोड को बार बार लिखने की आवश्यकता नहीं होती है।
2) To break a large program into a number of blocks.
   एक बहुत बड़े प्रोग्राम को छोटे छोटे ब्लॉक में विभक्त कर सकते है।

Types of function: C language supports two types of functions.

सी भाषा निम्न दो प्रकार के फंक्शन को सपोर्ट करती हैं।

1) Library function (predefined / inbuilt)
2) User define function

Library function: A number of functions those are defined in C compiler with different uses called library function. They are defined in object (machine) code and declared in header files therefore we need to include associated header file before using them.

सी कंपाइलर में कई सारे फंक्शन विभिन्न उपयोग के लिए पहले से ही परिभाषित रहते हैं जिन्हें लाइब्रेरी फंक्शन कहा जाता है। ऐसे फंक्शन ऑब्जेक्ट(मशीन कोड) मे परिभाषित रहते हैं और हैडर फ़ाइल में डिक्लैर होते हैं इसलिए इनका उपयोग करने से पहले संबन्धित हैडर फ़ाइल को शामिल करने की आवश्यकता होती है।

Example: printf(), scanf(), getchar(), putchar(), gets() and puts() are declared in stdio.h header fule. getch(), getche(), putch() and clrscr() are declared in conio.h header file.

```
Example:
#include<stdio.h>
#include<conio.h>
void main()
{
 int a;
 clrscr();
 printf("\nInput an integer number:");
 scanf("%d",&a);
 printf("\nYour input %d",a);
 getch();
}
Output: Input an integer number:22
You input 22
```

User defines function: A function that define and declared by user called user defined function.

एक ऐसा फंक्शन जिसे यूसर स्वयं परिभाषित और डिक्लैर करता है, उसे यूसर डिफ़ाइंड फंक्शन कहते हैं।

```
Example:
void show()
{
      printf("\nwww.LRsir.net/download ebooks");
}
```

Here show() is user defined function.
यहाँ **show()** एक यूसर परिभाषित फंक्शन है।

Working with user defined function:

We needs following three things when working with function.
जब फंक्शन के साथ कार्य करना हो तब हमें निम्न तीन चीजों की आवश्यकता होती है।

1) Function definition
2) Function declaration
3) Function calling

Function Definition: A block that contains executable code called function definition. Function definition has following characteristics.

एक ऐसा ब्लॉक जिसमें क्रियान्वित होने वाले कोड को लिखा जाता है उसे परिभाषित फंक्शन कहते हैं। परिभाषित फंक्शन में निम्न विशेषताएँ होती है।

a) **Function type / returning type:** specify which type of value return by a function.
फंक्शन द्वारा किस प्रकार की वैल्यू को लोटाया जाना है उसके प्रकार को परिभाषित करता है।

b) **Function name:** user define name. One program has a number of function with unique name.
यूसर द्वारा परिभाषित फंक्शन का नाम। एक प्रोग्राम में अद्वितीय नाम के कई सारे फंक्शन हो बनाए जा सकते हैं।

c) **List of arguments / parameters:** To receive nil or more values from outsources.
शून्य या कई सारी वैल्यू को बाहरी स्त्रोत से प्राप्त करने के लिए।

d) **Code block:** contains actual logic code.
इसमे वास्तविक लॉजिक कोड को लिखते हैं।

e) **A return statement:** used to return nil or one value.

यह अधिकतम एक वैल्यू को लोटता है।

Syntax:

```
ftype fname(type1 arg1, type2 arg2,..)
{
     Code block
    return value/expression;
}
```

Here

ftype=function type / return type (void / int / char / float etc)

fname= user define function name(show / factorial etc.)

type1, type2,…= data type of arguments

arg1, arg2,…= formal argument to hold outsource values

Code block= Logic code

return= Return program control with nil or one resulting value.

Example:

```
int sum(int x, int y)
{
 int z;
 z=x+y;
 return z;
}
```

Function Declaration: A single line of statement that tells about function definition to the compiler called function declaration or function prototype.

एक लाइन का स्टेटमेंट जो फंक्शन के बारें मे कंपाइलर को सूचित कर सके उसे फंक्शन डिक्लैरशन या फंक्शन प्रोटोटाइप कहते हैं।

Syntax:

```
ftype fname(type, type,..,type);
```

Example:

```
int sum(int, int);
```

It is declared before function calling.

इसे फंक्शन को कॉल करने से पहले डिक्लैर किया जाता है।

Function calling: A statement that uses function by passing values (arguments) and receives returning value called function calling or function call.

एक ऐसा स्टेटमेंट जिसके द्वारा फंक्शन में वैल्यूस (आर्गुमेंट्स) को पास करते हैं और रिटर्न होने वाली को प्राप्त करते हैं उसे फंक्शन कॉलिंग या फंक्शन कॉल कहते हैं।

Syntax: `var=fname(value1, value2,..,value-n);`

var= a variable that store returning value by function.

```
Example:
c=sum(10,20);
c=sum(5,2);
```

function calling can be made one or more times as per our requirements.
फंक्शन कॉलिंग को आवश्यकतानुसार एक या अधिक बार उपयोग में ला सकते हैं।

A complete program that uses library functions as well as user defines function:

```
#include<stdio.h>
#include<conio.h>
int sum(int,int);    //function declaration
void main()
{
 int a,b,c;
 clrscr();
 printf("\nInput two numbers:");
 scanf("%d%d", &a, &b);
 c=sum(a,b);          //function calling
 printf("\nResult=%d", c);
 getch();
}
//function definition
int sum(int x, int y)
{
 int z;
 z=x+y;
 return z;
}
Output: Input two numbers: 10      20 <enter>
Result=30
```

Function terminology:

1) **Calling function v/s Called function:** When function1 call another function2 then function1 known as calling function and function2 known as called function. In above example, main() is calling function whereas sum() is called function.
जब फंक्शन1 किसी अन्य फंक्शन2 को कॉल करता है तब फंक्शन1 को कॉलिंग फंक्शन कहते हैं और फंक्शन2 को कॉल्ड फंक्शन कहते हैं। उपरोक्त उदाहरण में **main()** कॉलिंग फंक्शन है जबकि **sum()** कॉल्ड फंक्शन है।
2) **Actual argument v/s Formal arguments:** Passing arguments of calling functions called actual arguments whereas receiving arguments of called function called formal arguments. In above example, <a,b> of main() are called actual arguments and <x,y> of sum() called formal arguments.

कॉलिंग फंक्शन के पासिंग आर्गुमेंट्स को एक्चुअल आर्गुमेंट्स कहते हैं जबकि कॉल्ड फंक्शन के रिसीविंग आर्गुमेंट्स को फॉर्मल आर्गुमेंट्स कहते हैं। उपरोक्त उदाहरण में **main()** फंक्शन के **<a,b>,** एक्चुअल आर्गुमेंट्स हैं और **sum()**फंक्शन के **<x,y>** फॉर्मल आर्गुमेंट्स है।

Function calling technique:

Arguments can be pass in following two techniques.
आर्गुमेंट्स को निम्न दो तरीके से पास किया जा सकता है।

1) Call by value or pass by value
2) Call by reference or pass by reference (reference / address / pointer)

Call by value: In this technique, calling function pass values of actual arguments to the formal arguments of called function. When formal arguments are changes then values of actual argument never changes.

इस तरीके में कॉलिंग फंक्शन एक्चुअल आर्गुमेंट्स की वैल्यूस को कॉल्ड फंक्शन के फॉर्मल आर्गुमेंट्स में पास करता है। जब फॉर्मल आर्गुमेंट्स में कोई बदलाव होता है तब एक्चुअल आर्गुमेंट्स की वैल्यूस कभी भी परिवर्तित नहीं होती है।

Example: Swapping of two numbers using call by value

```
#include<stdio.h>
void swap(int, int);
void main()
{
 int a=10,b=20;
 printf("\nBefore swapping a=%d\tb=%d",a,b);
 swap(a,b);
 printf("\nAfter swapping a=%d\tb=%d",a,b);
}
void swap(int x, int y)
{
 int t;
 t=x;
 x=y;
 y=t;
}
```

Output:

```
Before swapping a=10     b=20
After swapping  a=10     b=20
```

main():	a	b	Actual Arguments
Calling function	10	20	Values
	1000	2000	Addresses
	Pass values		
swap():	x	y	Formal argument
Called function	~~10~~ 20	~~20~~ 10	Values
	3000	4000	Addresses

We observe that values of a and b do not changed because formal arguments x and y are copy of actual arguments a and b.

उपरोक्त प्रोग्राम से यह निष्कर्ष निकलता है कि **a** एवं **b** की वैल्यूस मे कोई परिवर्तन नहीं होता है क्यूंकी फॉर्मल आर्गुमेंट्स **x** एवं **y** दोनों एक्चुअल आर्गुमेंट्स **a** और **b** की कॉपी है।

Call by reference: In this technique, calling function pass addresses of actual arguments to the formal arguments of called function. When formal arguments are changes then values of actual arguments always changes.

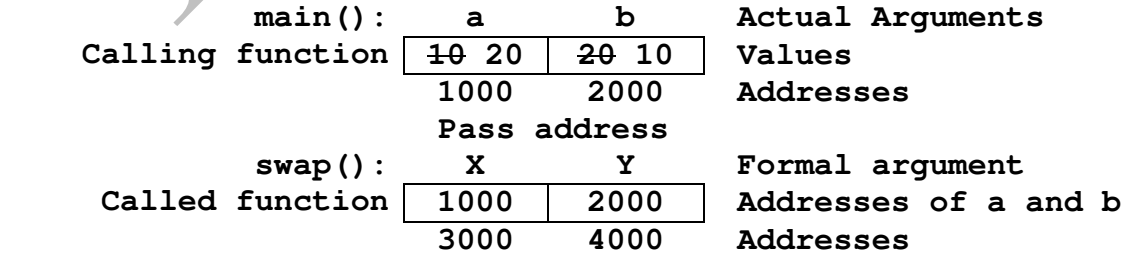
इस तरीके में कॉलिंग फंक्शन एक्चुअल आर्गुमेंट्स के एड्रैस को कॉल्ड फंक्शन के फॉर्मल आर्गुमेंट्स में पास करता है। जब फॉर्मल आर्गुमेंट्स में कोई बदलाव होता है तब एक्चुअल आर्गुमेंट्स की वैल्यूस ही हमेशा परिवर्तित होती है।

Example: Swapping of two numbers using call by value

```
#include<stdio.h>
void swap(int*, int*);
void main()
{
 int a=10,b=20;
 printf("\nBefore swapping a=%d\tb=%d",a,b);
 swap(&a,&b);
 printf("\nAfter swapping a=%d\tb=%d",a,b);
}
void swap(int *x, int *y)
{
 int t;
 t=*x;
 *x=*y;
 *y=t;
}
```

Output:

```
Before swapping a=10     b=20
After swapping  a=20     b=10
```



We observe that values of a and b becomes changed because formal arguments x and y are addresses of actual arguments a and b.

उपरोक्त प्रोग्राम से यह निष्कर्ष निकलता है कि **a** एवं **b** की वैल्यूस मे ही हमेशा परिवर्तन होता है क्यूंकी फॉर्मल आर्गुमेंट्स **x** एवं **y** दोनों एक्चुअल आर्गुमेंट्स **a** और **b** के एड्रैस को होल्ड किए हुये है।

Recursion

When a function call own self called recursion. In this process calling function and called function both are same.

जब कोई फंक्शन स्वयं को ही कॉल करे तब इसे रिकरसन कहते हैं। इस प्रक्रिया में कॉलिंग फंक्शन और कॉल्ड फंक्शन दोनों एक ही फंक्शन होते हैं।

Example: factorial of number using recursion n!=n*(n-1)!

```
#include<stdio.h>
long factorial(int);
void main()
{
 long fact;
 printf("\nInput number:");
 scanf("%d",&n);
 fact=factorial(n);
 printf("\nResult=%ld",fact);
}
long factorial(int n)
{
     int f;
     if(n==1)
     {
          return result;
     }
     else
     {
          f=n*factorial(n-1);
          return f;
     }
}
Output: Input number:4 <enter>
Result=24
```

factorial() is a recursive function because it has a statement that call own self. This process will be continues until n == 1.

factorial() एक रिकरसिव फंक्शन है क्यूंकी इसका एक स्टेटमेंट स्वयं को कॉल करता है। यह प्रोसैस **n==1** होने तक निरंतर बनी रहती है।

Demerits:

1) Slow down execution. क्रियान्वयन की गति को धीमी कर देता है।
2) Consume large memory space. मेमोरी स्पेस अधिक लगती है।

Command line argument:

Arguments of main() function called command line arguments.
main() फंक्शन के आर्गुमेंट्स को कमांड लाइन आर्गुमेंट्स कहा जाता है।
Syntax:

```
void main(int argc, char *argv[])
{
  Code block
}
```

argc and argv called command line arguments.
argc और **argv** को कमांड लाइन आर्गुमेंट्स कहते हैं।
argc=total arguments at command line.
कमांड लाइन पर कुल आर्गुमेंट्स की संख्या
argv=pointer to strings at command line
कमांड लाइन पर स्ट्रिंग्स का पाइंटर

Example:

```
#include<stdio.h>
void main(int argc, char *argv[])
{
 int i;
 printf("\nTotal arguments at command line=%d",argc);
 printf("\nArguments are:");
 for(i=0;i<argc;i++)
 {
  printf("%s\t",argv[i]);
 }
}
```

Running process:

```
Step1: save cmdline.c in c:\tc\bin\
Step2: Compile (alt+F9) then make exe file(F9).
Step3: Search cmdline.exe in c:\tc\ and copy into c:\ drive
Step4: open command prompt and move to c:\ drive
Step5: C:\cmdline.exe LRsir.net <enter>
Output:
     Total arguments at command line=2
     Arguments are: C:\cmdline.exe      LRsir.net
```

Local scope v/s Global scope:

Local scope: When variable / array / structure / pointer / function are declared inside function body then they are usable to only that function called local scope of that one.
जब वेरियबल / अरे / स्ट्रक्चर / पॉइंटर / फंक्शन को फंक्शन बॉडी के अंदर डिक्लैर किया जाता है तब इनका उपयोग केवल उसी फंक्शन मे किया जा सकता है इसे ही लोकल स्कोप कहा जाता है।

Global scope: When variable / array / structure / pointer/ function are declared outside all the function at top most places then they are usable to all functions of program called global scope of that one.

जब वेरियबल / अरे / स्ट्रक्चर / पॉइंटर / फंक्शन को फंक्शन बॉडी के बाहर, सबसे ऊपर डिक्लैर किया जाता है तब इनका उपयोग प्रोग्राम के फंक्शन कर सकते है, इसे ही ग्लोबल स्कोप कहा जाता है।

```
#include<stdio.h>
int x=10;          //global
void fun1();      //global
void main()
{
 int y=5;         //local
 printf("\nmain:Local %d",y);
 printf("\nmain:Global %d",x);
 fun1();
}
void fun1()
{
 int y=22;              //local
 printf("\nfun1:Local %d",y);
 printf("\nfun1:Global %d",x);
}
Output:
main: Local  5
main: Global 10
fun1: Local  22
fun1: Global 10
```

Lifetime of a variable: How long a variable alive throughout the program execution called lifetime. Lifetime of global variable is unlimited(begins to end) while lifetime of local variable is limited(until program control resides in that function except static variable)

प्रोग्राम मे कोई वेरियबल क्रियान्वयन के समय कब तक जीवित बना रहता है उसे ही लाइफटाइम कहते हैं। ग्लोबल वेरियबल का का लाइफटाइम असीमित होता है अर्थात प्रारम्भ से आन्टा तक जबकि स्टेटिक को छोड़कर सभी लोकल वेरियबल का लाइफटाइम सीमित होता है अर्थात जब तक प्रोग्राम कंट्रोलर उस फंक्शन मे मौजूद है।

In above program, lifetime of x has unlimited whereas y has limited.

इस प्रोग्राम मे, **x** का लाइफटाइम असीमित है जबकि **y** का सीमित।

Static variable:

A variable that declare using static keyword called static variable.
एक ऐसा वेरियबल जो **static** कीवर्ड का उपयोग कर डिक्लैर करते हैं उसे स्टेटिक वेरियबल कहते हैं।

Syntax: `static type var;`
Example: `static int x;`
Characteristics:

1) Default value is 0
   इसकी डिफ़ाल्ट वैल्यू 0 होती है।
2) Declare one time whether that function call many times.
   एक ही बार डिक्लैर होता है चाहे फंक्शन को कितनी बार भी कॉल करें।
3) It has local scope i.e. unknown by other function.
   इसका स्कोप लोकल होता है अर्थात अन्य फंक्शन के लिए अज्ञात रहता है।
4) Used to store a value when come back to same function.
   उसी फंक्शन पर फिर से लोटने पर पिछली वैल्यू को उपयोग मे ले सकते हैं।

Example: Count calling a function

```
#include<stdio.h>
void fun();
void main()
{
  fun();
  fun();
  fun();
}
void fun()
{
 static int x;
 int y=0;
 printf("\nx=%d\ty=%d", ++x, ++y);
}
```

Output:

```
x=1  y=1
x=2  y=1
x=3  y=1
```

Since x is a static therefore default value is 0 and always persist last assign value whereas y is non static therefore we assign 0 and never persist last assigned value because it create every time when a function calls.
चूंकि **x** एक स्टेटिक है इसलिए डिफ़ाल्ट वैल्यू 0 है और इसमे हमेशा अंतिम बार स्टोर की गई वैल्यू बनी रहती है जबकि **y** एक नॉन स्टेटिक है इसलिए अलग से 0 को असाइन करते हैं और यह अंतिम बार स्टोर की गई वैल्यू कभी भी बनी नहीं रहती क्यूंकी प्रत्येक फंक्शन कॉल पर यह निर्मित होता है।

Storage class:

Storage classes of variable are used to set following properties.
वेरियबल की स्टोरेज क्लासेस का उपयोग उसमे निम्न गुणों को सेट करने के लिए करते हैं।

1) ***Memory:*** RAM / CPU Register
2) ***Default value:*** 0 or garbage
3) ***Scope:*** Local / Global
4) ***Lifetime:*** Limited (until program control inside the function) / Unlimited (until program execution).

C supports four storage classes. There name, keywords, properties and examples are following.
सी भाषा चार प्रकार के स्टोरेज क्लासेस को सपोर्ट करती हैं। इनके नाम, केयवोर्ड, गुण और उदाहरण निम्न है।

SN	Storage class	Properties			
		Memory	Default value	scope	lifetime
1	auto(default) Ex: auto int x; or int x;	RAM	Garbage	Local	limited
2	register Ex: register int x;	CPU Register	Garbage	Local	limited
3	static Ex: static int x;	RAM	0	Local	Unlimited after creates
4	extern Ex: extern int x;	RAM	Garbage	global	unlimited

Identifier, Modifier, Qualifier, sizeof, typedef

Identifier: All user defined words called identifier. It includes name of variable, function, array, structure and label.
यूसर द्वारा परिभाषित सभी शब्दों को **identifier** कहा जाता है। इसमें वेरियबल, लेबल, फंक्शन, अरे, स्ट्रक्चर, यूनियन, एनुमारेशन आदि का नाम शामिल है।

Rules for identifier: (ex: variable)

1) Maximum character length is 8.
2) Allowed characters are a-z, A-Z, 0-9 and _ (underscore only).
3) Digit can't be used at first position.
4) Keywords are not allowed.
5) Blank space or comma not allowed.
Ex: Valid- principl, X_Y, a1, INT
Invalid- principle, X-Y, 1a, int, x y.

Modifier: Keywords which are used to change meaning of basic data type called modifier.

सी भाषा के ऐसे किवर्ड्स जिनका उपयोग आधारभूत डाटा टाइप में बदलाव करने के लिए कर सके उन्हे **modifier** कहा जाता है।

Ex: short, long, double, signed, unsigned

```
short  int or int: for 2 byte space
long  int or long: for 4 byte space
double float or double: for 8 byte space    double
signed:  to allow + or – data (default)
unsigned: to allow only + data
```

Qualifier: *const* keyword called qualifier and it is used to specify constant value to a word so that it never changed during program execution.

const कीवर्ड, एक **qualifier** है जिसका उपयोग किसी शब्द को **constant** बनाने के लिए करते हैं जिससे कि वह प्रोग्राम के क्रियान्वयन के समय बदल नहीं सके।

syntax: **const type cname=value;**

Example:

```
Example: area and circumference of circle.
#include<stdio.h>
void main()
{
      const float pi=3.14;
      float r=10,a,c;
      a=pi*r*r;
      c=2*pi*r;
      printf("\nArea=%f",a);
      printf("\nCircumference=%f",c);
}
```

sizeof: It is a keyword used to find memory size in butes for any data type / variable.

इस कीवर्ड का उपयोग किसी भी डाटा टाइप अथवा वरीयबल के साइज़ बाइट मे ज्ञात करने के लिए किया जाता है।

Syntax: **sizeof(type / var)**

Example:

```
#include<stdio.h>
void main()
{
      struct student
      {
       int id;
       char name[15];
```

```
      float fee;
     };
     int bytes;
     bytes=sizeof(struct student);
     printf("\nSize of student data type=%dB",bytes);
}
Output:
Size of student data type=21B
```

typedef: It is a keyword used to renaming any data type for current program..

इस कीवर्ड का उपयोग करेंट प्रोग्राम मे किसी भी डाटा टाइप को रिनेम करने के लिए करते है।

Syntax: `typedef type newname;`

Example:

```
#include<stdio.h>
typedef int integer
void main()
{
     integer n=10;  //valid
     printf("\nValue=%d",n);
}
Value=10
int renamed integer using typedef for this program.
```

इस प्रोग्राम के लिए **int** टाइप को **integer** मे **typedef** कीवर्ड द्वरा रिनेम कर दिया है।

PART7

Pointer
Pointer operator
Advantages of pointer
DMA:
- malloc()
- calloc()
- realloc()
- free()

Pointer to pointer
Pointer Arithmetic
Pointer of array
Array of Pointer
Pointer of string
Passing pointer of data
Passing array to function
Returning pointer from function
Pointer of function
Pointer of structure
Self Referencial Structure
Passing structure to function
Returning structure from function
Copy all members of structure into another.

Pointer:

Memory address of data is called pointer.
डाटा के मेमोरी एड्रैस को ही पोइंटर कहा जाता है।

Ex: `int a=10;`

a	Variable
10	Value
1000	Address

Pointer variable: Base address of data can be hold to a pointer variable.
डाटा के प्रारम्भिक एड्रैस को पोइंटर वेरियबल में होल्ड कर सकते है।

Syntax: `ptrtype *ptr;`

```
ptrtype= int / float / char etc.
ptr = pointer variables.
```

Example: `int *ptr;`

Pointer variable can hold only address of that data type. It means integer pointer variable can hold only address of integer data.
पोइंटर वेरियबल केवल उसी के प्रकार डाटा का एड्रैस होल्ड कर सकता है। इसका अर्थ यह है कि इंटीजर पोइंटर वेरियबल केवल इंटीजर डाटा का ही एड्रैस होल्ड करता है।

Pointer operator: There are following two operators.
पोइंटर ऑपरेटर निम्न दो प्रकार के होते है।

1) **& (Address of):** This unary operator returns memory address of variable.
यह यूनेरी ऑपरेटर वेरियबल के एड्रैस को रिटर्न करता है।
Syntax: `&var`

2) *** (Value at address):** This unary operator returns value at given address.
यह यूनेरी ऑपरेटर दिये गए एड्रैस पर स्टोर वैल्यू को रिटर्न करता है।
Syntax: `*ptr`

Example: find value using pointer variable.

```
#include<stdio.h>
void main()
{
  int a=10,b;
  int *p;
  p=&a;
  printf("\nAddress of a=%u", &a);
  printf("\nValue of a=%d", *p);
}
Output:
Address of a= 1000(Assumed)
Value of a=10
```

%u for unsigned address and %d for unsigned address.

Advantages of pointer:

1. We can modify value local variable using other function.
   हम लोकल वरीयबल की वैल्यू को अन्य फंक्शन में परिवर्तित कर सकते है।
2. We can return logically more than one value from any function.
   एक फंक्शन के द्वारा एक से अधिक वैल्यूस को लॉजिकली रिटर्न कर सकते है।
3. Pass whole array into function of any size.
   किसी भी साइज़ के सम्पूर्ण अरे को अन्य फंक्शन मे पास कर सकते है।
4. Pass all structure data to function.
   स्ट्रक्चर के सभी डाटा को फंक्शन मे पास कर सकते है।
5. Pass entire function to another.
   एक फंक्शन को ही अन्य फंक्शन मे पास कर सकते हैं।

DMA (Dynamic Memory Allocation):

At runtime we can allocate, extends and de allocate memory spaces for data called dynamic memory allocation. In C language, following library functions of alloc.h header file are used called DMA function.

रन टाइम पर हम डाटा के लिए मेमोरी स्पेसेस को आवंटित, विस्तारित एवं रद्द कर सकते हैं, इसे ही डाइनैमिक मेमोरी अलोकेशन कहते हैं। सी भाषा में alloc.h हैडर फ़ाइल के निम्न लाइब्रेरी फंक्शन का उपयोग करते हैं, जिन्हे DMA फंक्शन कहा जाता है।

1) malloc()
2) calloc()
3) realloc()
4) free()

malloc(): In this DMA function, we pass one argument for required number of bytes and it return pointer of allocates space. Default values of allocated spaces are garbage.

इस DMA फंक्शन में हम सिर्फ एक ही आर्गुमेंट को बाइट की संख्या के लिए पास करते है और यह स्पेस का आवंटन कर उसका पोइंटर रिटर्न कर देता है। आवंटित स्पेसेस की डिफ़ाल्ट वैल्यू गारबेज होती है।

Syntax: `type *ptr = (type*)malloc(bytes);`

Example: `int *pi= (int *)malloc(5*2); //for 5 integer data`

calloc(): In this DMA function, we pass two arguments; first for number of required data and second for size of each data. Default values of allocated spaces are zero.

इस DMA फंक्शन में हम दो आर्गुमेंट्स पास करते हैं; पहला आवश्यक डाटा की संख्या और दूसरा प्रत्येक डाटा की साइज़। आवंटित स्पेसेस की डिफ़ाल्ट वैल्यू ज़ीरो होती है।

Syntax: `type *ptr = (type*)calloc(n,size);`

```
Example: int *pi= (int *)calloc(5,2); //for 5 integer data
```

realloc(): This is used to extend number of bytes that was allocated by malloc().

इसका उपयोग **malloc()** द्वारा आवंटित मेमोरी स्पेस मे बाइट की संख्या का और अधिक विस्तार करने के लिए करते हैं।

Syntax: `type *ptr = (type*)realloc(ptr,bytes);`

```
Example: pi= (int *)realloc(pi,6); //for 3 integer data
```

free(): It is used to release memory space allocated by malloc() or calloc().

इसका उपयोग **malloc()** या **calloc()** द्वारा आवंटित मेमोरी स्पेस को मुक्त करने के लिए करते हैं।

Syntax: `free(ptr);`

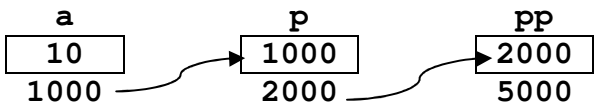
```
Example: free(pi); // release integer data
```

Pointer to pointer:

A pointer that holds address of lower level of pointer called pointer to pointer.
एक ऐसा पोइंटर जो उससे निम्न स्तर के पोइंटर को होल्ड करके रखता है उसे पोइंटर टु पोइंटर कहते हैं।

```
Example:
#include<stdio.h>
void main()
{
 int a,*p,**pp;
 a=10;
 p=&a;
 pp=&p;
```

```
 printf("\nValue of a= %d", **pp);
}
Output: Value of a=10
```



Value of pp = address of p
Value of p = address of a
Value of a = 10
Thus **pp→**2000(address of p)→*1000(address of a)=10.

***ppp called pointer to pointer to pointer. We can increase pointer level as per requirement.
***ppp को पोइंटर टु पोइंटर टु पोइंटर कहते हैं। इसप्रकार आवश्यकतानुसार पोइंटर के लेवेल को बड़ाया जा सकता है।

Pointer Arithmetic:

Arithmetic operators are used to move from current address to another. Following arithemetic operators can be appy on pointers.

अरिथमेटिक ऑपरेटर के द्वारा करेंट एड्रैस से किसी अन्य एड्रैस पर जया जा सकता है। निम्न अरिथमेटिक ऑपरेटर का ही उपयोग पोइंटर के साथ कर सकते हैं।

1) ptr + n : returns address of next n^{th} location
अगली n^{th} लोकेशन का एड्रैस प्रदान करता है
2) ptr – n : returns address of previous n^{th} location
पिछली n^{th} लोकेशन का एड्रैस प्रदान करता है
3) ptr1-ptr2 : number of locations between two addresses
दो एड्रैस के बीच कुल लोकेशन की संख्या प्रदान करता है।
4) ++ptr/ptr++ : update current address by next address
करेंट एड्रैस को अगले एड्रैस से अपडेट करता है।
5) --ptr/ptr-- : update current address by previous address
करेंट एड्रैस को पिछले एड्रैस से अपडेट करता है।
6) ptr+=n : update current address by next n^{th} address
करेंट एड्रैस को अगले n^{th} एड्रैस से अपडेट करता है।
7) ptr-=n : update current address by previous n^{th} address
करेंट एड्रैस को पिछले n^{th} एड्रैस से अपडेट करता है।

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

```
 int a=10,*ptr;
 ptr=&a;
 printf("\nCurrent address=%u", ptr);
 printf("\nNext address=%u", ptr+1);
 printf("\nPrevious address=%u", ptr-1);
}
Output:
Current address=2000 (assumed)
Next address=2002 (because int has 2 byte)
Previous address=1998
```

Pointer of array:

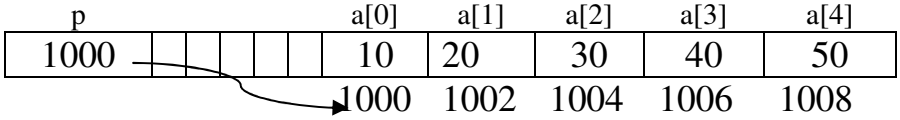
Base address of an array is called pointer of array. We can access all data of array using pointer of array.

किसी भी अरे के बेस एड्रैस को अरे का पोइंटर कहा जाता है। अरे के पोइंटर की सहायता से उसके सभी डाटा को एक्सैस कर सकते हैं।

Example:

```
#include<stdio.h>
void main()
{
 int a[5],i;
 int *p;
 p=&a;
 printf("\nInput 5 integers:");
 for(i=0;i<5;i++)
 {
  scanf("%d",&p[i]);
 }
 printf("\nArray data using pointer:");
 for(i=0; i<5;i++)
 {
  printf("%d\t",p[i]);
 }
}
Output: Input 5 integers: 10 20 30 40 50 <enter>
Array data using pointer: 10 20 30 40 50
```

Memory Allocation:



Remark:

1. a or &a[0] both are same that is base address of array

a या **&a[0]** दोनों का अर्थ एक मतलब एक जैसा ही है अर्थात अरे का बेस एड्रैस प्रदान करना ।

2. p[i] or *(p+i) both are same that access data of array
   p[i] या ***(p+i)** दोनों एक जैसे हैं जो अरे के डाटा को एक्सैस करते है।

Array of Pointer:

An array in which we can hold addresses of data called array of pointer. Using this feature we can access all data that are not store continuously in memory.

एक ऐसा अरे जिसमे कई सारे डाटा के एड्रेसेस को होल्ड कर सके उसे ही पोइंटर का अरे कहते हैं। इस गुण के द्वारा हम ऐसे सभी डाटा को एक ही क्रम मे एक्सैस कर सकते है जो अलग अलग मेमोरी एड्रैस पर स्थित है।

Example:

```
#include<stdio.h>
void main()
{
 int a,b,c;
 int p[3];
 int i;
 p[0]=&a;
 p[1]=&b;
 p[2]=&c;
 printf("\nInput 3 integers:");
 for(i=0;i<3;i++)
 {
  scanf("%d",p[i]);
 }
 printf("\ndata using array of pointer:");
 for(i=0; i<3;i++)
 {
  printf("%d\t",*p[i]);
 }
}
Output: Input 3 integers: 10 20 30 <enter>
data using array of pointer: 10 20 30
```

Memory Allocation:

p[0]	p[1]	p[2]		a		b			C
1000	2000	3000		10		20			30
				1000		2000			3000

Here p is array of pointer in which we first assign address of random places data then access continuously using loop.
यहाँ **p** एक पोइंटर का ही अरे है जिसमे हमने सबसे पहले अलग अलग स्थान के डाटा को असाइन किया फिर लूप के द्वारा क्रम से एक्सैस कर लिया।

Pointer of string: (string pointer)

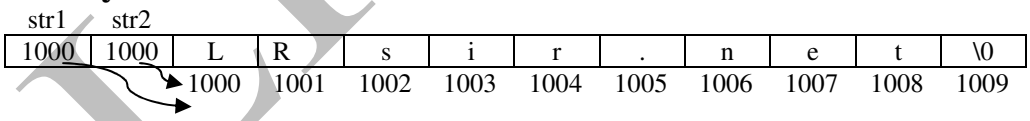
Base address of string is called pointer of string. using pointer of string, we can copy one string into another by = (assignment operator) and perform many operations.

किसी भी स्ट्रिंग के बेस एड्रैस को स्ट्रिंग का पोइंटर कहा जाता है। इसके उपयोग से एक स्ट्रिंग को अन्य स्ट्रिंग मे = (असाइनमेंट ऑपरेटर) के द्वारा कॉपी कर सकते हैं। इसके अलावा और भी स्ट्रिंग ऑपरेशन सरलता से किए जा सकते है।

Example:

```
#include<stdio.h>
void main()
{
 char *str1, *str2;
 printf("\nInput your name:");
 scanf("%s",str1);
 str2=str1;
 printf("\n%s\t%s",str1,str2);
}
Output: Input your name: LRsir.net <enter>
LRsir.net LRsir.net
```

Memory Allocation:



Thus while coping, whole string never copied, only base address.
अत: कॉपी करते समय पूरी स्ट्रिंग कॉपी नहीं होती, सिर्फ बसे एड्रैस कॉपी होता है।

Passing pointer of data: See call by reference

Passing array to function:

By passing base address and size of array, we can pass whole array to function.
अरे के बेस एड्रैस और साइज़ को किसी फंक्शन मे पास कर सम्पूर्ण अरे को पास कर सकते हैं।

```
Example:
#include<stdio.h>
Void passarray(int[], int);
void main()
{
 int a[5],i;
 printf("\nInput 5 integer numbers:");
 for(i=0;i<5;i++)
 {
     scanf("%d",&a[i]);
 }
 passarray(a,5);  //passing whole array
}

void passarray(int p[],int n)
{
 int i;
 printf("\nPassing Array data are:");
 for(i=0;i<n;i++)
 {
  printf("%d\t",p[i]);
 }
}
Output: Input 5 integer numbers:10 20   30   40   50<enter>
Passing Array data are:10    20   30   40   50
```

Here p[] is formal argument of passarray() function that hold base address of passing array and n is size of array used for ending of array. In function if we made any changes in array data then this changes made on actual array.
यहाँ **passarray()** एक फंक्शन है जिसका फॉर्मल आर्गुमेंट **p[],** पास किए गए अरे के बेस एड्रैस को होल्ड करता है और **n** एक इंटेगर वेरियबल है जिसके द्वरा अरे की साइज़ को स्टोर करते है। फंक्शन मे यदि अरे मे कोई बदलाव किया जाए तो वह वास्तव मे एक्चुअल अरे पर ही होगा।

Returning pointer from function:

When address of a variable returns from a function then such variable should be static because static variable persist value after removing program controller from that function.
जब फंक्शन से उसके किसी वेरियबल के एड्रैस को रिटर्न किया जाता है तो ऐसा वेरियबल स्टेटिक होना चाहिए क्यूंकी प्रोग्राम कंट्रोलर के फंक्शन से हटते ही केवल स्टेटिक वेरियबल मे वैल्यू बनी रहती है।

```
Example:
#include<stdio.h>
int* preturn();
```

```
void main()
{
 int *p;
 p=preturn();
 printf("\n:Value of other function=%d", *p);
}
int* preturn()
{
 static int a;
 printf("\ninput any integer value:");
 scanf("%d",&a);
 return &a;    //return address
}
Output: input any integer value:212 <enter>
Value of other function=212
```

Here preturn() function returns address of static variable a (&a) therefore function type is int*.

यहाँ **preturn()** फंक्शन स्टेटिक वेरियबल **a** का एड्रैस (**&a**) को रिटर्न करता है इसलिए फंक्शन का टाइप **int*** है।

Pointer of function:

When a function loads then it has also base address called pointer of function. We can hold address of any function then call them by pointer.

जब कोई फंक्शन लोड होता है तब उसका भी एक बेस एड्रैस होता है जिसे फंक्शन का पोइंटर कहा जाता है। हम किसी भी फंक्शन के एड्रैस को होल्ड कर उसके पोइंटर द्वरा फंक्शन को कॉल कर सकते हैं।

```
Example:
#include<stdio.h>
Void show();
void main()
{
 void *ptrfun();
 ptrfun=show;
 ptrfun();
 show();
}
void show()
{
 printf("\nLRsir.net");
}
Output:
LRsir.net
LRsir.net
```

Here ptrfun is a pointer of function. It can hold address of any function which has void type and without arguments like show. Only Function name gives its address.

यहाँ **ptrfun** एक फंक्शन का पोइंटर है। यह किसी भी ऐसे फंक्शन का एड्रैस होल्ड कर सकता है जिसका फंक्शन टाइप **void** हो और आर्गुमेंट नहीं हो जैसे **show**। केवल फंक्शन का नाम उसके एड्रैस को प्रदान करता है।

Pointer of structure:

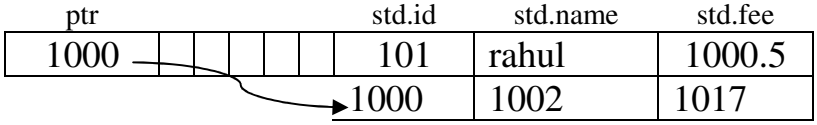
Base address of a structure variable is called pointer of structure. We can access all member of structure using pinter of structure.

किसी भी स्ट्रक्चर के बेस एड्रैस को स्ट्रक्चर का पोइंटर कहा जाता है। स्ट्रक्चर के पोइंटर की सहायता से उसके सभी डाटा मेम्बर्स को एक्सैस कर सकते हैं।

Example:

```
#include<stdio.h>
struct student
{
      int id;
      char name[15];
      float fee;
};
void main()
{
      struct student std;
      struct student *ptr;
      ptr=&std;
      printf("\nInput id, name and fee:");
      scanf("%d%s%f",&ptr->id,ptr->name,&ptr->fee);
      printf("\nStudent records:");
      printf("%d\t%s\t%f",ptr->id,ptr->name,ptr->fee);
}
Output:
Input id, name and fee:101     rahul      1000.5<enter>
Student records:101 rahul      1000.5
```

Memory Allocation:



Remark:

1. ptr is pointer of structure that hold only base address of student structure type. (invaild for other structure type)

ptr एक स्ट्रक्चर का पोइंटर है जो सिर्फ स्टूडेंट स्ट्रक्चर टाइप के बेस एड्रैस को होल्ड करेगा अर्थात किसी अन्य स्ट्रक्चर टाइप के लिए अमान्य होता है।

2. -> is called reference operator that is apply with pointer of structure to access members.
   -> एक रिफ्रेन्स ऑपरेटर है जिसे केवल स्ट्रक्चर के पोइंटर के साथ मेम्बर्स को एक्सैस करने के लिए करते है।

Self Referencial Structure:

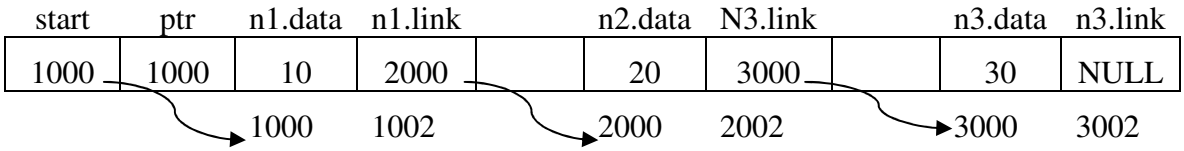
A member of structure that can hold address of same structure type then such structure called self referencial structure. That structure is used create a link list.

किसी स्ट्रक्चर का ऐसा मेम्बर जो उसी टाइप के किसी अन्य स्ट्रक्चर का एड्रैस होल्ड करे तब ऐसे स्ट्रक्चर को सेल्फ रिफ्रेन्सियल स्ट्रक्चर कहते है। इस स्ट्रक्चर का उपयोग लिंक लिस्ट बनाने के लिए करते हैं।

Example:

```
#include<stdio.h>
struct node
{
     int data;
     struct node *link;
};
void main()
{
     struct node n1,n2,n3;
     struct node *start,*ptr;
     printf("\nInput three integer data:");
     scanf("%d%d%d",&n1.data,&n2.data,&n3.data);
     start=&n1;
     n1.link=&n2;
     n2.link=&n3;
     n2.link=NULL;
     printf("\nAll data using link list:");
     ptr=start;
     while(ptr!=NULL)
     {
      printf("-->%d",ptr->data);
     }
}
Output:
Input three integer data:10   20   30 <enter>
All data using link list:-->10-->20-->30
```

Memory Allocation:



start is initial pointer of link list. Every part of structure contains data and link part. Link is used to store address of next node. Last node holds NULL value in link part to indicate ending of link list.

ptr एक स्ट्रक्चर का पोइंटर है लिंक लिस्ट का प्रारम्भिक एड्रैस स्टार्ट मे है। प्रटेक स्ट्रक्चर मे डाटा और लिंक पार्ट होते है। लिंक पार्ट मे अगली नोड के एड्रैस को स्टोर करते है। अंतिम नोड के लिंक पार्ट मे **NULL** असाइन करते है जिससे नोड के खत्म होने की सूचना प्रपट हो सके।

Passing structure to function(call by vale & call byreference), returning structure from function, copy all members of structure into another :

Pass structure using call byvalue & call by reference: When we pass one structure variable then all members are automatically passed. If we want to changes in actual structure then pass its address.

जब हम स्ट्रक्चर के किसी वेरियबल को फंक्शन मे पास करते हैं तब उसके सभी मेम्बर्स स्वत: ही पास हो जावेंगे। यदि हम एक्चुअल स्ट्रक्चर मे कोई परिवर्तन चाहते है तब उसका एड्रैस पास करते है।

Returning Structure: When a function return structure variable then all members are return.

जब कोई फंक्शन किसी स्ट्रक्चर वेरियबल को रिटर्न करता है तो उसके सभी डाटा मेम्बर्स भी एक साथ रिटर्न हो जाते हैं।

Copy structure: When a structure variable or returning structure is copy to same type of structure variable using one = (assignment) opertor then all members are automatically copied to other structure members.

जब एक स्ट्रक्चर वेरियबल या रिटर्निंग स्ट्रक्चर को उसी टाइप के अन्य स्ट्रक्चर वेरियबल मे = (असाइनमेंट) ऑपरेटर द्वारा कॉपी करते है तो सारे डाटा मेम्बर्स स्वत: कॉपी हो जाते है।

Example:

```
#include<stdio.h>
struct student
{
 int id;
 char name[15];
 float fee;
```

```
};
typedef struct student STD;
STD structfun(STD, STD*);
void main()
{
 STD s1={101,"Rahul",1000.5},s2={102,"nilesh",2000.5},s3;
 S3=Structfun(s1,&s2);
 printf("\ns1 record:%d\t%s\t%f",s1.id,s1.name,s1.fee);
 printf("\ns2 record:%d\t%s\t%f",s1.id,s1.name,s1.fee);
 printf("\ns3 record:%d\t%s\t%f",s1.id,s1.name,s1.fee);
}
STD structfun(STD s,STD *ps)
{
 s.id=103;
 strcpy(s.name,"rakesh");
 s.fee=3000.5;

 ps->id=104;
 strcpy(ps->name,"mukesh");
 ps->fee=4000.5;

 return s;
}
Output:
s1 record:101  Rahul     1000.5
s2 record:104  mukesh    4000.5
s3 record:103  rakesh    3000.5
```

Here s is copy of s1 and ps is pointer of s2. We made changes in s but members of s1 are unchanged whereas changing using ps are made on s2. When we return structure s then all members are copied into structure s3.

यहाँ **s** मे **s1** की कॉपी है और **ps** मे **s2** का एड्रैस है। जब **s** के बलवाव **s1** अप्रभावित है जबकि **ps** का बदलाव **s2** पर ही हुआ है। जब स्ट्रक्चर **s** को रिटर्न करते है तो उसके सभी मेम्बर्स को **s3** मे = के द्वारा कॉपी कर लेते है।

PART8

File Handling:
Properties of file
Operation on file
Types of file
File Pointer and mode
C libray function for file handling:
- fopen() and fclose()
- fputc() & fgetc()
- putw() & getw()
- fputs() & fgets()
- fprintf() & fscanf()
- fwrite() & fread()

Sequential v/s Random access of file:
- feof()
- ftell()
- rewind()
- fseek().

File Handling:

File is a collection of characters and store into secondary storage device like hard disk.
फ़ाइल बहुत सारे कैरक्टर का एक समूह होती है और किसी भी सेकंडरी स्टोरेज डिवाइस जैसे हार्ड डिस्क मे स्टोर की जाती है।

Properties of file:

1. File has a name with extension. Ex file1.txt, file2.c, file3.cpp
   फ़ाइल का एक नाम और एक्सटेंशन होता है जैसे **file1.txt,file2.c, file3.cpp**
2. File has a path. Ex: c:\tc\bin\file1.txt.
   फ़ाइल का एक पाथ होता है।
3. File has creting date & time.
   फ़ाइल मे बनाई गई दिनांक और टाइम दोनों होते हैं।
4. File has a size.
   फ़ाइल की साइज़ होती है।
5. File has read / write option.
   फ़ाइल मे रीड / राइट विकल्प होते हैं।

Operation on file:

Following operations can be apply on any file.
किसी भी फ़ाइल पर निम्न ऑपरेशन अप्लाई किए जा सकते हैं।

1) Create new file
   नई फ़ाइल बनाना
2) Open and close file
   फ़ाइल को ओपन और क्लोज़ करना
3) Read and write data to the file
   फ़ाइल से डाटा को रीड / राइट करना
4) Move read/write pointer in file
   फ़ाइल मे रीड / राइट पोइंटर को मुव करना।
5) Search , update and delete selected data.
   फ़ाइल से किसी डाटा को सर्च, अपडेट और डिलीट करना।

Types of file:

Operating system support following two type of files.

Sno	Text file	Binary file
1	All source code of program file store in ASCII called Text file. प्रोग्राम की सभी सोर्स कोड फ़ाइल जो **ASCII** मे स्टोर हो वे सभी टेक्स्ट	All machine coded file called binary file. सभी मशीन कोड युक्त फ़ाइल बाइनरि फ़ाइल होती है।

	फ़ाइल होती हैं।	
2	Newline character convert into combination of carriage return and linefeed characters. इसमे न्यू लाइन कैरक्टर सबसे पहले केरीज़ रिटर्न और लाइनफीड मे परिवर्तित होता है।	Newline character stores in as it is form. इसमे न्यू लाइन कैरक्टर उसी रूप मे होता है।
3	File is terminated using EOF (26) character. फ़ाइल के अंत मे **EOF(26)** कैरक्टर होता है।	File termination is identified using actual size of file. फ़ाइल के अंत को उसकी सीजे के द्वारा पहचाना जाता है।
4	In which any type of data are store in character form therefore file size is large than actual. इसमे किसी भी टाइप का डाटा कैरक्टर के रूप मे ही स्टोर होता है इसलिए फ़ाइल की साइज़ वास्तविक साइज़ से अधिक हो जाती है।	In which any type of data are store in as it is form इसमे किसी भी टाइप का डाटा उसी रूप मे स्टोर होता है।
5	Example: .c/.cpp/.java/.txt etc	Example: .obj/.exe/.com/.dat

File Pointer and mode:

we can read or write data of a file where file pointer is set. When a file is open then file pointer is set on a fixed position depends on following some mode.

फ़ाइल मे जिस स्थान पर पोइंटर सेट होता है वहीं से डाटा को रीड / राइट किया जा सकता है। जब कोई फ़ाइल ओपें होती है तब फ़ाइल पोइंटर एक निश्चित स्थान पर सेट होता है जो निम्न निम्न तीन मोड पर निर्भर करता है।

a. **Read mode:** read pointer set at the beginning of file and allow only read data from file.
   रीड पोइंटर फ़ाइल के प्रारम्भ मे सेट होता है और फ़ाइल से डाटा को केवल रीड कर सकते है।
b. **Write mode:** write pointer set at the beginning of file and allow only write data to the file.
   राइट पोइंटर फ़ाइल के प्रारम्भ मे सेट होता है और फ़ाइल मे डाटा को केवल कर सकते है।
c. **Append mode:** write pointer set at the end of file and allow only write data to the file.

राइट पोइंटर फ़ाइल के अंत मे सेट होता है और फ़ाइल से डाटा को केवल राइट कर सकते है।

In C language, file pointer is declared as-
सी भाषा मे फ़ाइल पोइंटर को निम्न प्रकार से डिक्लैर करते हैं।

```
FILE *fp;
```

Here FILE is structure data type defined in stdio.h
यहाँ **FILE** एक स्ट्रक्चर डाटा टाइप है जो **stdio.h** मे परिभाषित है।

C libray function for file handling:

1. fopen() and fclose()
2. fputc() and fgetc() / putc() and getc()
3. putw() and getw()
4. fputs() and fgets()
5. fprintf() and fscanf()
6. fwrite() and fread()
7. feof(), rewind() and fseek()

fopen() and fclose():

fopen():
This file function is used to open text file or binary file in read / write / append mode.
इस फ़ाइल फंक्शन का उपयोग टेक्स्ट फ़ाइल / बाइनरि फ़ाइल को रीड / राइट / अपेंड मोड मे ओपन करने के लिए करते हैं।

Syntax:

```
fp = fopen("filename","mode");
```

Where-

fp	: file pointer
filename	: file name
mode	:
	"rt" / "rb" read mode
	"wt" / "wb" read mode
	"at" / "ab" read mode
	t for text file (default)
	b for binary file

Events when file is open:
When file is open in "w" or "a" mode and given file name is not found then a new file is created of that name then open.
जब फ़ाइल को **"w"** या **"a"** मोड मे ओपन करते हैं और दी गई फ़ाइल बनी हुई नहीं है तो उसी नाम की फ़ाइल बन जाती है उसके बाद ओपन होती है।

If file is open in "w" mode and given file name is found then all data removes from the file.
जब फ़ाइल को **"w"** मोड मे ओपन करते हैं और दी गई फ़ाइल बनी हुई है तो उस फ़ाइल के सारे डाटा नष्ट हो जाते हैं।

When file is open is "r" mode and given file is not found then fp has NULL value.
जब फ़ाइल को **"r"** मोड मे ओपन करते हैं और दी गई फ़ाइल बनी हुई नहीं है तो **fp** मे **NULL** वैल्यू होती है।

fclose():

This file function is used to unload a file from memory so that all file data can saved successfully and we can reuse that file pointer to open another file.

इस फ़ाइल फंक्शन का उपयोग फ़ाइल को मेमोरी से अनलोड करने के लिए करते हैं जिससे कि फ़ाइल का सभी डाटा सफलता पूर्वक स्टोर हो जाए और हम उसी फ़ाइल पोइंटर का उपयोग किसी अन्य फ़ाइल को ओपन करने के लिए कर सके।

Syntax: `fclose(fp);`
Where-
fp :file pointer.

Example of fopen() and fclose():

```
#include<stdio.h>
void main()
{
 FILE *fp1,*fp2,*fp3;
 fp1=fopen("file1.txt","r");
 fp2=fopen("file2.txt","w");
 fp3=fopen("file3.txt","a");
 if(fp1==NULL)
 {
  printf("\nfile1.txt not exist");
  return;
 }
 //perform read / write / append operation
 fclose(fp1);
 fclose(fp2);
 fclose(fp3);
}
Output:
file1.txt not exist
<file2.txt and file3.txt creates in current location>
```

fputc() & fgetc()

fputc():

It is used to write one character in a file that open in write or append mode. After then file pointer automatically move to next character.

इसका उपयोग राइट या अपेंड मोड मे ओपन की गई फ़ाइल मे एक कैरक्टर को राइट करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `fputc(ch,fp); / putc(ch,fp);`

Where-

fp = write file pointer
ch = character
putc() is macro of fputc()

Example:

```
#include<stdio.h>
void main()
{
 char ch;
 FILE *fp;
 fp=fopen("file1.txt","w");
 printf("\nInput a character:");
 ch=getchar();
 fputc(ch,fp);
 fclose(fp);
 printf("\nWrite Success!");
}
Output:
Input a character:L <enter>
Write success!
```

fgetc():

It is used to read one character from a file that open in read mode. After then file pointer automatically move to next character.

इसका उपयोग रीड मोड मे ओपन की गई फ़ाइल से एक कैरक्टर को रीड करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `ch=fgetc(fp); / ch=getc();`

Where-

fp = read file pointer
ch = character variable
getc() is macro of fgetc()
fgetc() return EOF if on end of file or error.

Example:

```
#include<stdio.h>
void main()
{
 char ch;
 FILE *fp;
 fp=fopen("file1.txt","r");
 if(fp!=NULL)
 {
   ch=fgetc(fp);
 }
 fclose(fp);
 printf("\nFile data is:%c",ch);
}
```

Output:

File data is L

putw() & getw()

These two are most applicable for binary file.
ये दोनों बाइनरि फ़ाइल के लिए सबसे अधिक उपयुक्त हैं।

putw():

It is used to write one **integer** in a file that open in write or append mode. After then file pointer automatically move to next position.
इसका उपयोग राइट या अपेंड मोड मे ओपन की गई फ़ाइल मे एक इंटीजर को राइट करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `putw(n,fp);`

Where:

fp = write file pointer
n = integer value

Example:

```
#include<stdio.h>
void main()
{
 int n;
 FILE *fp;
 fp=fopen("file1.txt","wb");
 printf("\nInput an integer:");
 scanf("%d",&n);
 fputw(n,fp);
 fclose(fp);
 printf("\nWrite Success!");
}
```

Output:

```
Input an integer:223 <enter>
Write success!
```

getw():

It is used to read one integer from a file that open in read mode. After then file pointer automatically move to next position.

इसका उपयोग रीड मोड मे ओपन की गई फ़ाइल से एक इंटीजर को रीड करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `ch=getw(fp);`

Where-

fp = read file pointer
n = integer variable
getw() return EOF if on end of file or error.

Example:

```
#include<stdio.h>
void main()
{
 int n;
 FILE *fp;
 fp=fopen("file1.txt","rb");
 if(fp!=NULL)
 {
   n=getw(fp);
 }
 fclose(fp);
 printf("\nFile data is:%d",n);
}
Output:
File data is 223
```

fputs() & fgets()

fputs():

It is used to write one line of text / string in a file that open in write or append mode. After then file pointer automatically move to next position.

इसका उपयोग राइट या अपेंड मोड मे ओपन की गई फ़ाइल मे एक इंटीजर को राइट करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `fputs(str,fp);`

Where-

str= string
fp = write file pointer

Example:

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

```
void main()
{
 char str[30];
 FILE *fp;
 fp=fopen("file1.txt","w");
 printf("\nInput an string:");
 scanf("%s",str);
 fputs(str,fp);
 fclose(fp);
 printf("\nWrite Success!");
}
Output:
Input an string:LRsir.net <enter>
Write success!
```

fgets():

It is used to read one one line of text / string from a file that open in read mode. After then file pointer automatically move to next position.

इसका उपयोग रीड मोड मे ओपन की गई फ़ाइल से एक स्ट्रिंग को रीड करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `ch=fgets(str,n,fp);`

Where-

str = name string variable
n = size of string
fp = read file pointer
fgets() return NULL if on end of file or error.

Example:

```
#include<stdio.h>
void main()
{
 char str[30];
 FILE *fp;
 fp=fopen("file1.txt","r");
 if(fp!=NULL)
 {
   fgets(str,30,fp);
 }
 fclose(fp);
 printf("\nFile data is:%d",n);
}
Output:
File data is LRsir.net
```

fprintf() & fscanf()

fprintf():

It is used to write one or more data of any type in a file that open in write or append mode. After then file pointer automatically move to next position.

इसका उपयोग राइट या अपेंड मोड मे ओपन की गई फ़ाइल मे एक या अधिक किसी भी टाइप के डाटा को राइट करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `fprintf(fp,"Formatted code",var1,...);`

Where-

fp = write file pointer
formatted code like %d %f %c %s
var1,…are list of variable

Example:

```
#include<stdio.h>
struct student
{
 int id;
 char name[10];
 float fee;
};
void main()
{
 struct student std;
 FILE *fp;
 fp=fopen("file1.txt","w");
 printf("\nInput id,name and fee of student:");
 scanf("%d%s%f",&std.id,std.name,&std.fee);
 fprintf(fp,"%d\t%s\t%f",std.id,std.name,std.fee);
 fclose(fp);
 printf("\nWrite Success!");
}
Output:
Input id,name and fee of student:101 rahul 1000.5 <enter>
Write success!
```

fscanf():

It is used to read one or more data of any type from a file that open in read mode. After then file pointer automatically move to next position.

इसका उपयोग रीड मोड मे ओपन की गई फ़ाइल से एक स्ट्रिंग को रीड करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `fscanf(fp,"Formatted code",&var1,...);`

Where-

fp = write file pointer

formatted code like %d %f %c %s
var1,…are list of variable
fscanf() return EOF on end of file and 0 if unable to read.

Example:

```
#include<stdio.h>
struct student
{
 int id;
 char name[10];
 float fee;
};
void main()
{
 struct student std;
 FILE *fp;
 fp=fopen("file1.txt","r");
 if(fp!=NULL)
 {
   fscanf(fp,"%d%s%f",&std.id,std.name,&std.fee);
 }
 fclose(fp);
 printf("\nFile dat is:%d\t%s\t%f", std.id,std.name,std.fee);
}
Output:
File data is 101    rahul     1000.5
```

Remark: If we have to any changes in structure then we have to also made changes in all fprintf() and fascanf().

यदि स्ट्रक्चर मे कोई बदलाव करना हो तो सभी **fprintf()** और **fscanf()** मे भी बदलाव होगा।

fwrite() & fread()

These two are most applicable for binary file.
ये दोनों बाइनरि फ़ाइल के लिए सबसे अधिक उपयुक्त हैं।

fwrite():
It is used to write one or more data of any type in a file that open in write or append mode. After then file pointer automatically move to next position.
इसका उपयोग राइट या अपेंड मोड मे ओपन की गई फ़ाइल मे एक या अधिक किसी भी टाइप के डाटा को राइट करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `fwrite(ptr,sizeof(data),n,fp);`
Where-

ptr = pointer of data
sizeof = operator to evaluate data size
n = number of data (in case of array)
fp = file pointer

Example:

```
#include<stdio.h>
struct student
{
 int id;
 char name[10];
 float fee;
};
void main()
{
 struct student std;
 FILE *fp;
 fp=fopen("file1.txt","wb");
 printf("\nInput id,name and fee of student:");
 scanf("%d%s%f",&std.id,std.name,&std.fee);
 fwrite(&std,sizeof(std),1,fp);
 fclose(fp);
 printf("\nWrite Success!");
}
Output:
Input id,name and fee of student:101 rahul 1000.5 <enter>
Write success!
```

fread():

It is used to read one or more data of any type from a file that open in read mode. After then file pointer automatically move to next character.

इसका उपयोग रीड मोड मे ओपन की गई फ़ाइल से एक स्ट्रिंग को रीड करने के लिए करते हैं। इसके बाद फ़ाइल पोइंटर स्वत: ही अगली पोसिशन पर मुव हो जाता है।

Syntax: `fread(ptr,sizeof(data),n,fp);`

Where-

ptr = pointer of data
sizeof = operator to evaluate data size
n = number of data (in case of array)
fp = file pointer
fread() return 0 if on end of file or unable to read.

Example:

```
#include<stdio.h>
struct student
{
 int id;
 char name[10];
```

```
 float fee;
};
void main()
{
 struct student std;
 FILE *fp;
 fp=fopen("file1.txt","rb");
 if(fp!=NULL)
 {
   fread(&std,sizeof(std),1,fp);
 }
 fclose(fp);
 printf("\nFile dat is:%d\t%s\t%f", std.id,std.name,std.fee);
}
Output:
File data is 101    rahul     1000.5
```

Remark: If we have to any changes in structure then we have to not made any changes in fread() and fwrite().

यदि स्ट्रक्चर मे कोई बदलाव करना हो तो किसी भी **fwite()** और **fread()** मे बदलाव नहीं किया जाता।

Sequential v/s Random access of file:

Sequential access: When a file is open then file pointer moves from beginning to ending of the file while read/write operation then it is called sequencial access of file.
जब किसी फ़ाइल को ओपन करते हैं तब रीड/राइट ऑपरेशन करते समय फ़ाइल पोइंटर प्रारम्भ से अंत तक मुव करता है, इसे फ़ाइल का सीक्वेंशियल एक्सैस कहते हैं।

Random access: If we want to access data from any position of the file then we have to move file pointer on that position and then apply read/write operation. It is called random access of file.
यदि हम फ़ाइल की किसी भी पोजीशन पर डाटा को एक्सैस करना हो तब हमे फ़ाइल पोइंटर को उस पोजीशन पर ले जाना होगा और फिर उस पर रीड/राइट ऑपरेशन को अप्लाई करेंगे। इसे ही फ़ाइल का रैनडम एक्सैस कहते हैं।

For random access of file we need following file functions that perform on file pointer.
फ़ाइल के रैनडम एक्सैस के लिए हमे निम्न फ़ाइल फंक्शन्स की आवश्यकता होती है जो फ़ाइल पोइंटर पर कार्य करे।

1. feof()
2. ftell()
3. rewind()

4. fseek()

feof():

This function checks file pointer position. If it is at EOF then return true(non zero) value otherwise false(0).

यह फंक्शन है फ़ाइल पोइंटर की पोजीशन को चेक करता है। यदि यह **EOF** पर हो तो **true** वैल्यू रिटर्न करता है अन्यथा **false**।

Syntax: `feof(fp)`
Where- fp=file pointer
return true / false

```
Example: show all file data up to end of file
#include<stdio.h>
void main()
{
 char ch;
 FILE *fp;
 fp=fopen("file1.c","r");
 while(!feof(fp))
 {
  ch=fgetc(fp);
  putchar(ch);
 }
 fclose(fp);
}
Output: www.LRsir.net download free ebooks
```

ftell():

returns current file pointer position in long integer value.

यह फ़ाइल पोइंटर की करेंट पोजीशन को लॉन्ग इंटीजर वैल्यू मे रिटर्न करता है।

Syntax: `pos=ftell(fp);`
Where- pos=long integer value

```
Example: show file pointer position after reading data
#include<stdio.h>
void main()
{
 char ch;
 long pos;
 FILE *fp;
 fp=fopen("file1.c","r");
 ch=fgetc(fp);
 putchar(ch);
 pos=ftell(fp);
 fclose(fp);
 printf("\tCurrent file pointer at:%ld",pos);
}
Output: w Current file pointer at:1
```

rewind():

repositions file pointer at the beginning of file so that read/write operation can be done from beginning.

यह फ़ाइल पोइंटर को फ़ाइल के सबसे प्रारम्भ मे सेट कर देता है जिससे कि रीड/राइट ऑपरेशन पुन: शुरुआत से कर सकें।

Syntax: `rewind(fp);`

Where- fp=file pointer

```
Example: Read file data two times
#include<stdio.h>
void main()
{
 char ch;
 FILE *fp;
 fp=fopen("file1.c","r");
 while(!feof(fp))
 {
  ch=fgetc(fp);
  putchar(ch);
 }
 rewind(fp);
 while(!feof(fp))
 {
  ch=fgetc(fp);
  putchar(ch);
 }
 fclose(fp);
}
Output: www.LRsir.net download free ebooks
www.LRsir.net download free ebooks
```

fseek():

repositions file pointer at anywhere in file so that read/write operation can be done from that position.

यह फ़ाइल पोइंटर को फ़ाइल मे किसी भी पोजीशन पर सेट कर देता है जिससे कि रीड/राइट ऑपरेशन उस पोजीशन से कर सकें।

Syntax: `fseek(fp,n,position);`

Where- fp=file pointer
n= number of bytes to move from position of file pointer.
Position: set file pointer when
0/SEEK_SET :At beginning position
1/SEEK_CUR :At current position
2/SEEK_END :At ending position

Example:

1. Set file pointer at the beginning:
   `fseek(fp,0,SEEK_SET);`

2. Set file pointer at the ending:
   `fseek(fp,0,SEEK_END);`
3. Set file pointer to next 2 byte from current position:
   `fseek(fp,2,SEEK_CUR);`
4. Set file pointer to back 2 bytes from current position:
   `fseek(fp,-2,SEEK_CUR);`
5. Set file pointer to next 2 byte from begining:
   `fseek(fp,2,SEEK_SET);`
6. Set file pointer to back 2 byte from ending:
   `fseek(fp,-2,SEEK_END);`

Example: print reverse file data

```
#include<stdio.h>
void main()
{
 char ch;
 FILE *fp;
 fp=fopen("file1.c","r");
 fseek(fp,0L,SEEK_END);
 printf("\nSize of file is %ld bytes",ftell(fp));
 fclose(fp);
}
```

Output: Size of file is 13 bytes

PART9

Preprocessor:

Preprocessor statements

File include

Macro Directives

Passing argument to macro

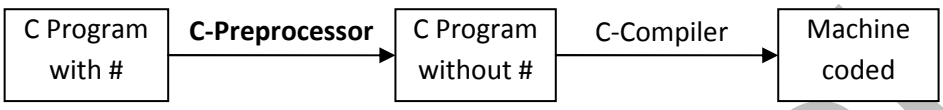
Conditional compilation:

#ifdef, #ifndef, #if, #elif, #else, #endif.

Preprocessor:

A program that processes all # statements (directive) of C program called preprocessor. During compilation, any C program is first processed by preprocessor then compiled into machine code.

यह एक प्रोग्राम होता है जो सी प्रोग्राम के केवल **#** स्टेटमेंट (डायरेक्टिव) को ही प्रोसैस करता है उसे प्रीप्रोसेसर कहते हैं। कंपाइलेशन के समय कोई भी सी प्रोग्राम सबसे पहले प्रीप्रोसेसर द्वारा प्रोसैस होगा उसके बाद ही मशीन कोड मे कंपाइल होता है।



Preprocessor statements: (Directives)

A statement that begins with # but not terminated by semi colon(;) called directives. It supports following type if directives.

ऐसे स्टेटमेंट जो **#** से होते है किन्तु अंत ; से ना हो उसे डायरेक्टिव कहते हैं। यह निम्न प्रकार के डायरेक्टिव को सपोर्ट करता है।

a. File include(#include)
b. Macro(#define)
c. Conditional compilation(#ifdef, #ifndef, #if, #else,#endif)
d. String preprocessor (#, ##, pragma etc)

File include preprocessor:

#include is a preprocessor statement that is used to include content of given filename into current program code before comilation.

#include एक ऐसा प्रीप्रोसेसर स्टेटमेंट है जिसके द्वारा करंट प्रोग्राम कोड मे किसी अन्य फ़ाइल के कंटैंट को कंपाइल होने से पहले जोड़ सकते हैं।

It is used in following two forms.

इसका उपयोग निम्न दो प्रकार से कर सकते हैं।

1) **#include<filename>**
   Search filename only in predefine path.(c:\tc\include\).
   यह **filename** को केवल पूर्वपरिभाषित पाथ(**c:\tc\include\)** मे ही खोजता है।
2) **#include"filename"**
   Search filename first in current working directory(c:\tc\bin\). When not found then search in predefine path.(c:\tc\include\)
   यह **filename** को पहले उसी डाइरैक्टरि मे खोजता है जिसमे हम कार्य कर रहे हों(**c:\tc\include\)**। जब वह नहीं मिलती तब पूर्वपरिभाषित पाथ(**c:\tc\include\)** मे ही खोजता है।

```
Example: include header files
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
#include<alloc.h>
#include<stdlib.h>
```

Macro Directives:

#define is called macro that work as find and replace. If we want to made many changes of same thing then we can use macro.
#define को ही मैक्रो कहते हैं एवं यह फाइंड और रिप्लेस की तरह कार्य करता है। यदि हमे एक जैसे कई सारे बदलाव करना हो तब मेक्रो का उपयोग कर सकते हैं।

Syntax: `#define MACRO expansion`

MACRO is a macro name that replace by expansion by preprocessor.
MACRO एक मेक्रो नाम है जो प्रीप्रोसेसर द्वारा **expansion** से रिप्लेस हो जाता है।

```
Example:
#define PI 3.14
```

Here PI is macro. Every PI will be replacing by 3.14 in a program.
यहाँ **PI** एक मेक्रो है और प्रोग्राम का प्रत्येक **PI, 3.14** से रिप्लेस हो जाएगा।

Macro v/s Function :(passing argument to macro)

Macro can be used just similar to function for single line of statement. It means we can also pass arguments into macro just like function. Macro is different than function because macro performs before compilation whereas function performs after compilation at runtime.
एक ही लाइन के स्टेटमेंट मे मेक्रो का उपयोग फंक्शन के समान भी कर सकते है अर्थात मेक्रो मे भी आर्गुमेंट्स को पास किया जा सकता है। सिर्फ फर्क इतना रहता है कि मेक्रो कि प्रोसैस कंपाइल होने से पहले पूरी हो जाती है जबकि फंक्शन कंपाइल होने के बाद रन टाइम पर प्रोसैस होता है।

Syntax: `#define MACRO(var1,var2,..) expansion`

```
Example: Find area and circumference of circle for given
radius.
#include<stdio.h>
#define PI 3.14
#define AREA(x)     PI*x*x
#define CIRCUM(x) 2*PI*x
void main()
{
 floar r,a;
 printf("\nInput radious of circle:");
 scanf("%d",&r);
```

```
 a=AREA(r);
 printf("\nArea=%d",a);
 printf("\nCircumference=%d",a);
}
Output: Input radius of circle: 10
Circumference=62.8
```

During compilation, preprocessor first replace all PI by 3.14 then AREA(x) replace by 3.14*r*r. argument x is replace by r.

कंपाइलेशन के पहले प्रीप्रोसेसर सबसे पहले सभी **PI** को 3.14 से रिप्लेस करता है उसके बाद **AREA(x)** 3.14∗**r**∗**r** से। आर्गुमेंट **x** को **r** रिप्लेस करता है।

Conditional compilation:

If we want to skip some part of code for compilation then it is done using following syntax.

यदि हम प्रोग्राम कोड के कुछ हिस्सों को ही कंपाइल करना चाहते हैं तब इसे निम्न कंडीशनल डायरेक्टिव से कर सकते हैं।

Syntax1:

```
#ifdef MACRO
     Code1
#else
     Code2
#endif
```

When MACRO is defined then code1 compiled otherwise code2.

जब **MACRO** को परिभाषित किया जाता है तब ही **code1** कंपाइल होगा अन्यथा **code2**।

Syntax2:

```
#ifndef MACRO
     Code1
#else
     Code2
#endif
```

When MACRO is not defined then code1 compiled otherwise code2.

जब **MACRO** को परिभाषित नहीं किया जाता है तब ही **code1** कंपाइल होगा अन्यथा **code2**।

Syntax3:

```
#if condition
     Code1
#else
     Code2
#endif
```

When condition (like 4>3) is true then code1 compiled otherwise code2.

जब **condition (**जैसे 4**>3) true** होगी तब ही **code1** कंपाइल होगा अन्यथा **code2**।

Syntax4:

```
#if condition1
     Code1
#elif condition2
     Code2
#else
     Code3
#endif
```

When condition1 is true then code1 is compiled otherwise if condition2 is true then code2 compiled otherwise code3.

जब **condition1 true** होगी तब ही **code1** कंपाइल होगा अन्यथा **condition2 true** होगी तब ही **code2** कंपाइल होगा अन्यथा **code3**।

```
Example:
#include<stdio.h>
#define PI 3.14
void main()
{
  #ifdef PI
     printf("\n#ifdef part compiled");
  #else
     printf("\n#else part compiled");
 #ifdef
}
Output: #ifdef part compiled
```

C-Programs

1. Read any two integer number then perform all arithmatic operation
2. Swapping of any two number using temp variable
3. Calculate Area and Circumference of a Circle.
4. Area of triangle
5. Even odd status of given number
6. Solve a Quadratic Equation - Ax^2 + Bx + C = 0
7. Read any three integer number then find greatest number
8. Read any 5 subject marks then print total, percentage and division
9. Find input character is small case or capital or digit or special symbol.
10. Develop menu driven program for all arithmatic operation
11. Find addition of all number from 1 to 10
12. Display multiplication c number of any given number
13. Find factorial of any given number
14. y=x^n
15. Print all even numbers up to 100
16. Find sum of all digits, reverse and palindrome status of an input number
17. Display pyramid using (*).
18. S=1+ x + x^2/2! + x^3/3! +----+ x^n/n!
19. Print all combination of 1,2 and 3
20. Prime numbers up to 100
21. LCM
22. Find large and small from 10 data.
23. sorting array
24. Transpose Matrix of a given Matrix of order 3 X 2.
25. Addition of two matrices of order 3 X 2.
26. Multiply matrices of order 3 X 2 by matrices of order2X3
27. Swapping of any two number using call by values.
28. Swapping of any two number using call by reference / pointer
29. Find factorial of input number(n!) using recursion.
30. Display the Fibonacci series up to 20 th term.
31. Create an array of structure to hold records of 20 cricketers that define age, number of played matches and average runs. Read these records and display them in ascending order by average runs.(Define functions for each job)
32. Write any union program.
33. Copy one string into another and count number of characters.
34. use Pointer and access operators
35. accessing an integer variable through pointer of pointer.
36. Calculate the sum of Array elements through pointers.

37. Create dynamic array for n size and access all elements.
38. Write a Program that allocate memory dynamicaly for given number of students then enter total marks for all, display each marks with grand total
39. Write a Program that stores adrress of any 5 integer address in pointer array then display all values using pointer array
40. Define a structure book with members name, author and price. Access using pointers.
41. Write a Program that sorting a list of array by passing entir array in to another function
42. Write a Program that call a function using function pointer for calculating factorials of any number
43. Write a Program that calulate length of any string without standard function
44. Write a Program that converts all character of string to upper case without standard function
45. reverse a string
46. Write a Program that copy one string to another without standard function
47. Read some text from keyboard then write to and read from file.
48. count file character, space, tab and lines
49. write and read file
50. file copy
51. Read content of one file and copy to another file from command line
52. use fprintf
53. use fscanf
54. use fprintf for store student records
55. use fscanf to read student records
56. use fwrite to store student records
57. use fread to read student records
58. Define a structure item with members particular, rate and price. Read these data from keyboard to item variable then write to and read from file.
59. search string in file
60. display calendar
61. Convert Decimal to Binary, Hexa_Descimal and octal
62. Convert Decimal to Binary, Hexa_Descimal and octal
63. Convert Binary, Hexa_Descimal and octal to Decimal
64. Convert Decimal to Binary, Hexa_Descimal and octal

```
/* Read any two integer number then perform all arithmatic
operation*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int a,b;
 clrscr();
 printf("\nInput any two numbers:");
 scanf("%d%d",&a,&b);
 printf("\nAddition=%d",a+b);
 printf("\nSubtraction=%d",a-b);
 printf("\nMultiplication=%d",a*b);
 printf("\nDivision=%d",a/b);
 printf("\nRemainder=%d",a%b);
 getch();
}
/* Swapping of any two number using temp variable*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int a,b,temp;
 clrscr();
 printf("\nInput any two numbers:");
 scanf("%d%d",&a,&b);
 printf("\nBefore swapping a=%d,b=%d",a,b);
 temp=a;
 a=b;
 b=temp;
 printf("\nAfter swapping a=%d,b=%d",a,b);
 getch();
}
/* Calculate Area and Circumference of a Circle.*/
#include<stdio.h>
#include<conio.h>
void main()
{
 float r,a,c;
 clrscr();
 printf("\nInput radius of circle:");
 scanf("%f",&r);
 a=3.14*r*r;
 c=2*3.14*r;
 printf("\nArea=%f Circumference=%f",a,c);
 getch();
}
/*Area of triangle*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

```
void main()
{
 float a,b,c,s,area;
 clrscr();
 printf("\nInput value of a,b,c:");
 scanf("%d%d%d",&a,&b,&c);
 s=(a+b+c)/2;
 area=sqrt(s*(s-a)*(s-b)*(s-c));
 printf("\nArea=%f",&area);
 getch();
}
```

/* Even odd status of given number*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int n;
 clrscr();
 printf("\nInput any number:");
 scanf("%d",&n);
 if(n%2==0)
      printf("\n%d is even",n);
 else
      printf("\n%d is Odd",n);
getch();
}
```

/*Solve a Quadratic Equation - Ax^2 + Bx + C = 0*/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
 float a,b,c,d,x1,x2;
 clrscr();
 printf("\nInput value of a,b,c:");
 scanf("%d%d%d",&a,&b,&c);
 d=(b*b)-(4*a*c);
 if(d<0)
     printf("\nImaginary Result");
  else
   {
    x1=(-b+sqrt(d))/(2*a);
    x2=(-b-sqrt(d))/(2*a);
    printf("Root1=%f,Root2=%f",x1,x2);
   }
 getch();
}
```

/* Read any three integer number then find greatest number*/

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

```
void main()
{
 int a,b,c;
 clrscr();
 printf("\nInput any three numbers:");
 scanf("%d%d%d",&a,&b,&c);
 if(a>b)
 {
     if(a>c)
          printf("\n%d is greatest",a);
     else
          printf("\n%d is greatest",c);
 }
 else
 {
     if(b>c)
          printf("\n%d is greatest",b);
     else
          printf("\n%d is greatest",c);
 }
 getch();
}
```

/* Read any 5 subject marks then print total, percentage and division*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int m1,m2,m3,m4,m5,total;
 float avg;
 clrscr();
 printf("\nInput five subject marks:");
 scanf("%d%d%d%d%d",&m1,&m2,&m3,&m4,&m5);
 total=m1+m2+m3+m4+m5;
 avg=(float)total/5;

printf("\n\nTotal=%d\nPercentage=%f\nDivision=",total,avg);
 if(avg>=60)
     printf("First");
 else  if(avg>=45)
          printf("Second");
       else if(avg>=33)
               printf("Third");
         else
               printf("Fail");

 getch();
}
```

/*Find input character is small case or capital or digit or special symbol.*/

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

```
#include<conio.h>
void main()
{
 char ch;
 clrscr();
 printf("\nPress any key:");
 fflush(stdin);
 ch=getche();
 if(ch>='A'&&ch<='Z')
    printf("\nCapital Letter");
  else if(ch>='a'&&ch<='z')
       printf("\nSmall Letter");
       else if(ch>='0'&&ch<='9')
            printf("\ndigit");
         else
            printf("\nSpecial Symbole");
 getch();
}
```

/* Develop menu driven program for all arithmatic operation*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a,b;
 char ch;
 clrscr();
 printf("\nInput any two numbers:");
 scanf("%d%d",&a,&b);
 printf("\nOperator:");
 ch=getch();
 switch(ch)
 {
  case '+':
       printf("=%d",a+b);
       break;
  case '-':
       printf("=%d",a-b);
       break;
  case '*':
       printf("=%d",a*b);
       break;
  case '/':
       printf("=%d",a/b);
       break;
  case '%':
       printf("=%d",a%b);
       break;
  default:
       printf("\nInvalid Operator");
 }
```

```
 getch();
}
/* Find addition of all number from 1 to 10*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,sum=0;
 clrscr();
 for(i=1;i<=10;i++)
     sum+=i;

 printf("\nSum from 1 to 10=%d",sum);
 getch();
}
/* Display multiplication number of any given number*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int n,i;
 clrscr();
 printf("\nInput any number:");
 scanf("%hd",&n);
 for(i=1;i<=10;i++)
      printf("\n%d*%d=%d",n,i,n*i);
 getch();
}
/* Find factorial of any given number*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int n,i;
 long f;
 clrscr();
 printf("\nInput any number:");
 scanf("%d",&n);
 f=1;
 for(i=n;i>0;i--)
     f=f*i;
 printf("\nFactorial of %d is %ld",n,f);
 getch();
}
/*y=x^n*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int x,n,y=1,i;
 clrscr();
```

```
 printf("Input value of x and n:");
 scanf("%d%d",&x,&n);
 for(i=1;i<=n;i++)
     y=y*x;
 printf("y=%d",y);
 getch();
}
```

/*Print all even numbers up to 100*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int n,i;
 clrscr();
 for(i=1;i<=100;i++)
 {
  if(i%2==0)
     printf("%d ",i);
 }
 getch();
}
```

/*Find sum of all digits, reverse and palindrome status of an input number*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int n,r,x,s;
 clrscr();
 printf("Input any number:");
 scanf("%d",&n);
 x=n;
 s=0;
 r=0;
 while(n>0)
 {
  s=s+(n%10);
  r=r*10+n%10;
  n=n/10;
 }
 printf("Sum of digits=%d,Reverse=%d",s,r);
 if(r==x)
   printf("\It is palindrom");
 else
   printf("\It is not palindrom");
 getch();
}
```

/*Display pyramid using (*).*/

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

```
{
 int i,j,k;
 clrscr();
 for(i=1;i<=5;i++)
 {
  printf("\n");
  for(j=1;j<=5-i;j++)
    printf(" ");
  for(k=1;k<=i;k++)
    printf("* ");
 }
 getch();
}
/*S=1+ x + x^2/2! + x^3/3! +----+ x^n/n!*/
#include<stdio.h>
#include<conio.h>
void main()
{
 float x,y,t,s;
 int f,n,i,j;
 clrscr();
 printf("\nInput value of x and last term number:");
 scanf("%f%d",&x,&n);
 s=1;
 for(i=1;i<=n;i++)
 {
  y=1;
  f=1;
  for(j=1;j<=i;j++)
   {
    y=y*x;
    f=f*j;
   }
  s=s+(y/f);
 }
 printf("\ns=%f",s);
 getch();
}
/*Print all combination of 1,2 and 3*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j,k;
 clrscr();
 for(i=1;i<=3;i++)
 {
   for(j=1;j<=3;j++)
   {
    printf("\n");
    for(k=1;k<=3;k++)
```

```
          printf("%d%d%d\t",i,j,k);
    }
  }
  getch();
}
/*Prime numbers up to 100*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j;
 clrscr();
 for(i=1;i<=100;i++)
 {
   for(j=2;j<i;j++)
     if(i%j==0)break;
   if(j==i||i==1)
     printf("%d ",i);
 }
 getch();
}
/*LCM*/
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,max;
clrscr();
printf("\nInput two number:");
scanf("%d%d",&a,&b);
max=a>b?a:b;
while(1)
{
 if(max%a==0 && max%b==0)
 {
  break;
 }
 max++;
}
printf("LCM=%d",max);

getch();
}
/*Find large and small from 10 data.*/
#include<stdio.h>
#include<conio.h>
void main()
{
 int a[10],i,s,l;
 clrscr();
 printf("\nInput 10 data:");
```

```
 for(i=0;i<10;i++)
   scanf("%d",&a[i]);
 l=0;
 s=a[0];
 for(i=0;i<10;i++)
  {
   if(l<a[i]) l=a[i];
   if(s>a[i]) s=a[i];
  }
 printf("\n Large=%d,small=%d",l,s);
 getch();
}
```

/*sorting array*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5];
int i,j,c;
clrscr();
printf("\nInput 5 data");
for(i=0;i<=4;i++)
{
 scanf("%d",&a[i]);
}

for(i=0;i<=4;i++)
{
 for(j=0;j<4-i;j++)
 {
  if(a[j]>a[j+1])
  {
   c=a[j];
   a[j]=a[j+1];
   a[j+1]=c;
  }
 }
}
printf("\nAll array \a\adata in increasing order:");
for(i=0;i<=4;i++)
{
 printf("%d\t",a[i]);
}
getch();
}
```

/*Transpose Matrix of a given Matrix of order 3 X 2.*/

```
#include<stdio.h>
#include<conio.h>
#define ROW 3
#define COL 2
void main()
```

```
{
 int m[ROW][COL],t[COL][ROW];
 int r,c;
 clrscr();
/* Read matrix*/
 for(r=0;r<ROW;r++)
 {
  printf("\nInput Row-%d data:",r+1);
  for(c=0;c<COL;c++)
    scanf("%d",&m[r][c]);
 }
/* logic for transpose*/
 for(r=0;r<ROW;r++)
 {
   for(c=0;c<COL;c++)
      t[c][r]=m[r][c];
 }

/* Print transpose*/
 for(r=0;r<COL;r++)
 {
   printf("\n");
   for(c=0;c<ROW;c++)
      printf("%d\t",t[r][c]);
 }
 getch();
}
```

/*Addition of two matrices of order 3 X 2.*/

```
#include<stdio.h>
#include<conio.h>
#define ROW 3
#define COL 2
void main()
{
 int m1[ROW][COL],m2[ROW][COL],s[ROW][COL];
 int r,c;
 clrscr();

 printf("\nInut First matrix");
 for(r=0;r<ROW;r++)
 {
  printf("\nRow-%d data:",r+1);
  for(c=0;c<COL;c++)
    scanf("%d",&m1[r][c]);
 }

 printf("\nInut second matrix");
 for(r=0;r<ROW;r++)
 {
  printf("\nRow-%d data:",r+1);
  for(c=0;c<COL;c++)
```

```
    scanf("%d",&m2[r][c]);
 }

/* logic for Addition*/
 for(r=0;r<ROW;r++)
 {
   for(c=0;c<COL;c++)
     s[r][c]=m1[r][c]+m2[r][c];
 }

 printf("\nFirst Matrix=");
 for(r=0;r<ROW;r++)
 {
   printf("\n");
   for(c=0;c<COL;c++)
      printf("%d\t",m1[r][c]);
 }

 printf("\nSecond Matrix=");
 for(r=0;r<ROW;r++)
 {
   printf("\n");
   for(c=0;c<COL;c++)
      printf("%d\t",m2[r][c]);
 }

 printf("\nAddition of matrices=");
 for(r=0;r<ROW;r++)
 {
   printf("\n");
   for(c=0;c<COL;c++)
      printf("%d\t",s[r][c]);
 }

 getch();
}
```

/*Multiply matrices of order 3 X 2 by matrices of order2X3 */

```
#include<stdio.h>
#include<conio.h>
#define ROW 3
#define COL 2
#define COL1 3
void main()
{
 int m1[ROW][COL],m2[COL][COL1],p[ROW][COL1];
 int r,c,c1;
 clrscr();

 printf("\nInut First matrix");
 for(r=0;r<ROW;r++)
```

```
 {
  printf("\nRow-%d data:",r+1);
  for(c=0;c<COL;c++)
    scanf("%d",&m1[r][c]);
 }

 printf("\nInut second matrix");
 for(r=0;r<COL;r++)
 {
  printf("\nRow-%d data:",r+1);
  for(c=0;c<COL1;c++)
    scanf("%d",&m2[r][c]);
 }

/* logic for Addition*/
 for(r=0;r<ROW;r++)
 {
   for(c1=0;c1<COL1;c1++)
   {
     p[r][c1]=0;
     for(c=0;c<COL;c++)
          p[r][c1]+=m1[r][c]*m2[c][c1];
   }
}

 printf("\nFirst Matrix=");
 for(r=0;r<ROW;r++)
 {
   printf("\n");
   for(c=0;c<COL;c++)
       printf("%d\t",m1[r][c]);
 }
 printf("\nSecond Matrix=");
 for(c=0;c<COL;c++)
 {
   printf("\n");
   for(c1=0;c1<COL1;c1++)
       printf("%d\t",m2[c][c1]);
 }

 printf("\nMultiplication of matrices=");
 for(r=0;r<ROW;r++)
 {
   printf("\n");
   for(c1=0;c1<COL1;c1++)
       printf("%d\t",p[r][c1]);
 }

 getch();
}
```

/*Swapping of any two number using call by values.*/

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

void swap(int a,int b);

void main()
{
 int a,b;
 clrscr();
 printf("\nInput two values:");
 scanf("%d%d",&a,&b);
 printf("\nBefore  Swap a=%d,b=%d",a,b);
 swap(a,b);
 getch();
}

void swap(int a,int b)
{
 int c;
 c=a;
 a=b;
 b=c;
 printf("\nAfter Swap a=%d,b=%d",a,b);
}
```

/*Swapping of any two number using call by reference / pointer*/

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

void swap(int *a,int *b);

void main()
{
 int a,b;
 clrscr();
 printf("\nInput two values:");
 scanf("%d%d",&a,&b);
 printf("\nBefore  Swap a=%d,b=%d",a,b);
 swap(&a,&b);
 printf("\nAfter Swap a=%d,b=%d",a,b);
 getch();
}

void swap(int *a,int *b)
{
 int c;
 c=*a;
 *a=*b;
 *b=c;
```

```
}
/*Find factorial of input number(n!) using recursion.*/

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

long fact(int n);

void main()
{
 int n;
 long f;
 clrscr();
 printf("\nInput any number:");
 scanf("%d",&n);
 f=fact(n);
 printf("\n%d!=%ld",n,f);
 getch();
}

long fact(int n)
{
 long f;
 if(n==1)
   return 1;
 else
   f=n*fact(n-1);
 return f;
}
/*Display the Fibonacci series up to 20 th term.*/

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

void fibo(int f,int s);

void main()
{
 int f,s;
 clrscr();
 fibo(1,1);
 getch();
}

void fibo(int f,int s)
{
  int sum;
  if(f>32767/2)
    return;
  else
  {
```

```
    printf("%d+",f);
    sum=f+s;
    f=s;
    s=sum;
    fibo(f,s);
    }
}
```

/*Create an array of structure to hold records of 20 cricketers that define age, number of played matches and average runs. Read these records and display them in ascending order by average runs.(Define functions for each job) */

```
#include<stdio.h>
#include<conio.h>
#define SIZE 3

void read();
void display();
void asending();

struct player
{
 char name[15];
 int age;
 int n;
 int avg;
}p[SIZE];


void main()
{
 clrscr();
 read();
 asending();
 display();
 getch();
}

void read()
{
  int i;
  for(i=0;i<SIZE;i++)
  {
   printf("\nInput palyer-%d Record",i+1);
   printf("\nPlayer Name:");
   scanf("%s",p[i].name);
   printf("\nAge:");
   scanf("%d",&p[i].age);
   printf("\nTotal Played matched number:");
   scanf("%d",&p[i].n);
```

```
   printf("\nAverage run:");
   scanf("%d",&p[i].avg);
  }
}

void display()
{
  int i;
  clrscr();
  printf("\nName\t\tAge\tPlay Matchs\tAverage Run");
  for(i=0;i<SIZE;i++)

printf("\n%s\t\t%d\t%d\t\t\t%d",p[i].name,p[i].age,p[i].n,p
[i].avg);
}

void asending()
{
 int i,j;
 struct player t;
 for(i=0;i<SIZE;i++)
   for(j=0;j<SIZE-i;j++)
     if(p[j].avg<p[j+1].avg)
      {
       t=p[j];
       p[j]=p[j+1];
       p[j+1]=t;
      }
}
```

/*Write any union program.*/

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

union abc
{
 char a;
 int b;
 float c;
}ch,i,f;

void main()
{
 clrscr();
 printf("\nInput char,integer and real data:");
 scanf("%c%d%f",&ch.a,&i.b,&f.c);
 printf("%c %d %f",ch.a,i.b,f.c);
 getch();
}
```

/*Copy one string into another and count number of characters.*/

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

void main()
{
 char s1[15],s2[15];
 int c;
 clrscr();
 printf("\nInput Your name:");
 scanf("%s",s1);
 strcpy(s2,s1);
 c=strlen(s2);
 printf("s1=%s s2=%s count characters=%d",s1,s2,c);
 getch();
}
```

/*use Pointer and access operators*/

```
#include<stdio.h>
void main()
{
 int x=10;
 int *p;
 p=&x;
 printf("\nValue of x=%d",*p);
 printf("\nValue of p=%u",p);
 printf("\nAddress of x=%u",&x);
 printf("\nAddress of p=%u",&p);
 printf("\nValue of x=%d",x);
}
```

/* accessing an integer variable through pointer of pointer.*/

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

void main()
{
 int a,*p,**pp;
 clrscr();
 printf("\nInput any number:");
 scanf("%d",&a);
 p=&a;
 pp=&p;
 printf("a=%d,*p=%d,**pp=%d",a,*p,**pp);
 getch();
}
```

/*Calculate the sum of Array elements through pointers.*/

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

```
void main()
{
 int a[10],i,s=0,*p;
 clrscr();
 p=&a[0];
 printf("\nInput any 10 number:");
 for(i=0;i<10;i++)
   scanf("%d",&p[i]);

 for(i=0;i<10;i++)
   s+=p[i];

 printf("sum=%d",s);
 getch();
}
```

/*Create dynamic array for n size and access all elements.*/

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void main()
{
 int a,n,i,*p;
 clrscr();

 printf("\nInput size:");
 scanf("%d",&n);
 p=(int *)calloc(n,sizeof(int));
 printf("Input %d data:",n);
 for(i=0;i<n;i++)
   scanf("%d",&p[i]);

 for(i=0;i<n;i++)
   printf("%d ",p[i]);
 free(p);
  getch();
}
```

/*Write a Program that allocate memory dynamicaly for given number of students then enter total marks for all, display each marks with grand total*/

```
#include<stdio.h>
int main()
{
int i,no,sum,*start,*ptr;
printf("\nEnter number of student:");
scanf("%d",&no);
start=(int *)malloc((sizeof(int))*no);
ptr=start;
for(i=1;i<=no;i++)
{
```

```
printf("\n%d:Enter marks:",i);
scanf("%d",ptr);
ptr++;
}
ptr=start;
printf("Marks:\n");
for(i=1,sum=0;i<=no;i++)
{
printf("%d\n",*ptr);
sum+=*ptr;
ptr++;
}
printf("Total Grand Marks=%d",sum);
return(0);
}
```

/*Write a Program that stores adrress of any 5 integer address in pointer array then display all values using pointer array */

```
#include<stdio.h>
int main()
{
int *ptr_arr[5];
int a,b,c,d,e,count;
printf("\nEnter any 5 integer number:");
scanf("%d%d%d%d%d",&a,&b,&c,&d,&e);
ptr_arr[0]=&a;
ptr_arr[1]=&b;
ptr_arr[2]=&c;
ptr_arr[3]=&d;
ptr_arr[4]=&e;
for(count=0;count<=4;count++)
printf("\nValue=%d",*ptr_arr[count]);
}
```

/*Define a structure book with members name, author and price. Access using pointers.*/

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

struct book
{
 char name[15];
 char aut[15];
 int price;
}b,*p;

void main()
{
 clrscr();

 p=&b;
```

```
 printf("\nInput Book name,writer and price:");
 scanf("%s%s%d",p->name,p->aut,&p->price);

 printf("\nBook name:%s\nwriter: %s\nprice:%d",
       p->name,p->aut,p->price);

 getch();
}
```

/*Write a Program that sorting a list of array by passing entir array in to another function*/

```
#include<stdio.h>
int main()
{
int i,arr[10];
void sortfun(int *ptr,int size);
printf("\nEnter any 10 number:");
for(i=0;i<=9;i++)
scanf("%d",&arr[i]);
sortfun(&arr[0],10);
printf("\nSorted list:\n");
for(i=0;i<=9;i++)
printf("%d\t",arr[i]);
return(0);
}

void sortfun(int *ptr,int size)
{
int i,j,temp;
for(i=0;i<=size-1;i++)
{
 for(j=i+1;j<=size-1;j++)
 {
  if(*(ptr+i)>*(ptr+j))
  {
   temp=*(ptr+i);
   *(ptr+i)=*(ptr+j);
   *(ptr+j)=temp;
   }
   }
 }
}
```

/*Write a Program that call a function using function pointer for calculating factorials of any number*/

```
#include<stdio.h>
int main()
{
int no;
long fact;
long fact_fun(int);
long (*func_ptr)(int);
printf("\nEnter any number:");
```

```
scanf("%d",&no);
func_ptr=fact_fun;
printf("\nAddress of fact_fun() Function is %u",func_ptr);
fact=(*func_ptr)(no);
printf("\n%d!=%ld",no,fact);
}

long fact_fun(int n)
{
long f;
int i;
for(i=1,f=1;i<=n;i++)
f=f*i;
return (f);
}
```

/*Write a Program that calulate length of any string without standard function*/

```
#include<stdio.h>
int main()
{
char *str;
int l;
int strlenx(char *);
printf("\nType any message:");
gets(str);
l=strlenx(str);
printf("\nLength=%d",l);
}

int strlenx(char *s)
{
int count=0;
while(*s!=NULL)
{
count++;
s++;
}
return (count);
}
```

/*Write a Program that converts all character of string to upper case without standard function*/

```
#include<stdio.h>
int main()
{
char *str;
void struprx(char *);
printf("\nType any message:");
gets(str);
struprx(str);
puts(str);
return (0);
```

```
}

void struprx(char *s)
{
while(*s!=NULL)
{
if(*s>='a' && *s<='z')
*s=*s-32;
s++;
}
}
```

/* reverse a string*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[15];
int i,l;
clrscr();
printf("\nInput string:");
scanf("%s",str);
l=0;
for(i=0;i<=14;i++)
{
 if(str[i]=='\0')
 {
   break;
 }
 l++;
}
printf("%d",l);
for(i=l;i>=0;i--)
{
 printf("%c",str[i]);
}
getch();
}
```

/*Write a Program that copy one string to another without standard function*/

```
#include<stdio.h>
int main()
{
char *source,*target;
void strcpyx(char *,char *);
printf("\nType any message:");
gets(source);
strcpyx(target,source);
puts(target);
return 0;
```

```
}

void strcpyx(char *t,char *s)
{
while(*s!=NULL)
{
*t=*s;
++s;
++t;
}
*t=NULL;
}
```

/*Read some text from keyboard then write to and read from file.*/

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

void main()
{
 char ch;
 FILE *fp;
 clrscr();
 fp=fopen("file1.txt","a");
 printf("\nInput some text:");
 while(1)
 {
  ch=getche();
  if(ch==13) break;
  fputc(ch,fp);
 }
 fclose(fp);

 printf("\File data is:\n");
 fp=fopen("file1.txt","r");
 while(feof(fp))
 {
  ch=fgetc(fp);
  putch(ch);
 }
 fclose(fp);
 getch();
}
```

/*count file character, space, tab and lines*/

```
#include<stdio.h>
#include<stdlib.h>
main()
{
char ch=NULL,filename[30];
int chr=0,tab=0,line=0;
```

```
FILE *fp;
puts("Type file name with path and extension");
gets(filename) ;
fp=fopen(filename,"rb");
if(fp==NULL)
{
     printf("Unable");
     exit(0);
}
while(1)
{
     ch=getc(fp);
     if(ch==EOF)
        break;
     if(ch=='\n')
        line++;
     if(ch=='\t')
        tab++;
     chr++;

}
fclose(fp);
printf("Char=%d Tab=%d Line=%d",chr,tab,line);
getch();
return (0);
}
```

/*write and read file*/

```
#include<stdio.h>
#include<stdlib.h>
main()
{
char ch=NULL;
FILE *fp;
fp=fopen("c:\\file1.txt","wt");
if(fp==NULL)
{
     printf("Unable");
     exit(0);
}
puts("Enter contents:");
while(1)
{
     ch=getche();
     if(ch==26)
        break;
     if(ch==13)
      {
       puts("\n");
       ch='\n';
      }
     putc(ch,fp);
```

```
}
fclose(fp);

fp=fopen("c:\\file1.txt","rt");
if(fp==NULL)
{
     printf("Unable");
     exit(0);
}
puts("contents:");
ch=NULL;
while(1)
{
     ch=getc(fp);
     if(ch==EOF)
        break;
     if(ch=='\n')
        puts("\n");
     else
        putch(ch);

}
fclose(fp);
getch();
return;
}
/*file copy*/
#include<stdio.h>
#include<stdlib.h>
main()
{
char ch=NULL,filename[30];
FILE *fp1,*fp2;
puts("Type exist file name with path and extension");
gets(filename) ;
fp1=fopen(filename,"r");
if(fp1==NULL)
{
     printf("Unable");
     exit(0);
}

puts("Type new file name with path and extension");
gets(filename) ;
fp2=fopen(filename,"w");
if(fp2==NULL)
{
     printf("Unable");
     exit(0);
}
```

```
while(1)
{
     ch=getc(fp1);
     if(ch==EOF)
        break;
     putc(ch,fp2);
}
fclose(fp1);
fclose(fp2);
getch();
return (0);
}
```

/*Read content of one file and copy to another file from command line*/

```
#include<stdio.h>
void main(int argc,char *argv[])
{
 char ch;
 FILE *fp1,*fp2;
 if(argc!=3)
 {
  printf("\nUnsufficient files to copy!");
  return;
 }
 fp1=fopen(argv[1],"r");
 if(fp1==NULL)
 {
  printf("\n%s file not exist",argv[1]);
  return;
 }
 fp2=fopen(argv[2],"w");

 while(!feof(fp1))
 {
  ch=fgetc(fp1);
  fputc(ch,fp2);
 }
 fclose(fp1);
 fclose(fp2);
 printf("\nFile copy success!");
}
```

Output:

Save as-fcopy.c

Make fcopy.exe in C:\

Run from command line:

C:\fcopy.exe file1.txt file2.txt <enter>

File copy success!

/*use fprintf*/

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

```
struct emp
{
 int id;
 char name[15];
 float sal;
};
void main()
{
 struct emp e;
 FILE *fw;
 clrscr();

 printf("\nInput id, name and salary:");
 scanf("%d%s%f",&e.id,e.name,&e.sal);

 fw=fopen("file1.txt","w");

 fprintf(fw,"%d %s %f ",e.id,e.name,e.sal);
 getch();
}
/*use fscanf*/
#include<stdio.h>
#include<conio.h>
struct emp
{
 int id;
 char name[15];
 float sal;
};
void main()
{
 struct emp e;
 FILE *fr;
 clrscr();

 fr=fopen("file1.txt","r");
 fscanf(fr,"%d%s%f",&e.id,e.name,&e.sal);

 printf("%d %s %f ",e.id,e.name,e.sal);
 getch();
}
/*use fprintf for store student records*/
#include<stdio.h>
#include<stdlib.h>
struct student
{
int rno;
char nm[30];
float fee;
};
```

```
main()
{
FILE *fp;
struct student s;
char ch='y';
fp=fopen("c:\\stu_list.dat","w");
if(fp==NULL)
{
printf("\n Unable to create");
exit(0);
}
while(ch=='y' || ch=='Y')
{
printf("\nRoll no, name, fee:");
scanf("%d%s%f",&s.rno,s.nm,&s.fee);
fprintf(fp,"%d %s %f\n",s.rno,s.nm,s.fee);
printf("\nPress Y key to continue:");
fflush(stdin);
ch=getche();
}
fclose(fp);
getch();
return(0);
}
```

/*use fscanf to read student records*/

```
#include<stdio.h>
#include<stdlib.h>
struct student
{
int rno;
char nm[30];
float fee;
};

main()
{
FILE *fp;
struct student s;
clrscr();
fp=fopen("c:\\stu_list.dat","r");
if(fp==NULL)
{
printf("\n Unable to open");
exit(0);
}
printf("\nRoll no   name   fee");
while(fscanf(fp,"%d%s%f",&s.rno,s.nm,&s.fee)!=EOF)
     printf("\n %d\t%s\t%f",s.rno,s.nm,s.fee);

fclose(fp);
getch();
```

```
}
/*use fwrite to store student records*/
#include<stdio.h>
#include<stdlib.h>
struct student
{
int rno;
char nm[30];
float fee;
};

main()
{
FILE *fp;
struct student s;
char ch='y';
fp=fopen("c:\\stu_list.dat","wb");
if(fp==NULL)
{
printf("\n Unable to create");
exit(0);
}
while(ch=='y' || ch=='Y')
{
printf("\nRoll no, name, fee:");
scanf("%d%s%f",&s.rno,s.nm,&s.fee);
fwrite(&s,sizeof(s),1,fp);
printf("\nPress Y key to continue:");
fflush(stdin);
ch=getche();
}
fclose(fp);
getch();
}
/*use fread to read student records*/
#include<stdio.h>
#include<stdlib.h>
struct student
{
int rno;
char nm[30];
float fee;
};

main()
{
FILE *fp;
struct student s;
clrscr();
fp=fopen("c:\\stu_list.dat","rb");
if(fp==NULL)
```

```
{
printf("\n Unable to create");
exit(0);
}
printf("\nRoll no   name   fee");
while(fread(&s,sizeof(s),1,fp)!=0)
     printf("\n %d        %s
%f",s.rno,s.nm,s.fee);

fclose(fp);
getch();
}
```

/*Define a structure item with members particular, rate and price. Read these data from keyboard to item variable then write to and read from file.*/

```
#include<stdio.h>
#include<conio.h>
struct item
{
 char p[15];
 int r,q;
}i,o;
void main()
{
 FILE *fp;
 clrscr();
 printf("\nInput Item name,rate,quantity");
 scanf("%s%d%d",i.p,&i.r,&i.q);
 fp=fopen("file1.dat","w");
 fprintf(fp,"%s %d %d\n",i.p,i.r,i.q);
 fclose(fp);

 fp=fopen("file1.dat","r");
 fscanf(fp,"%s%d%d",o.p,&o.r,&o.q);
 printf("\nItem name=%s,rate=%d,quantity=%d",o.p,o.r,o.q);

 fclose(fp);
 getch();
}
```

/*search string in file*/

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

main()
{
FILE *fp;
int cmp,count=0,len;
char str1[30],str2[30];
clrscr();
fp=fopen("c:\\find.txt","r");
if(fp==NULL)
```

```
{
printf("\n not found");
exit(0);
}
printf("\nenter any searched string:");
len=strlen(gets(str1));

while(fgets(str2,len+1,fp)!=NULL)
{
cmp=strcmp(str1,str2);
if(cmp==0)
{
printf("\nFound at %d",count);
break;
}
fseek(fp,++count,SEEK_SET);
}

fclose(fp);
getch();
return(0);
}
```

/* display calendar*/

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


/* function returns last day number of given year*/

int first_day(int month,int year)
{
int prv_year,cent_year=0,hund_year=0,tenth_year=0;
int leap_year=0, ord_year=0;
int odd_day1=0,odd_day2=0,last_day,days;

prv_year=year-1;

cent_year=(prv_year/400)*400;
hund_year=(prv_year-cent_year)/100;
tenth_year=(prv_year-cent_year)%100;
leap_year=tenth_year/4;
ord_year=tenth_year-leap_year;

odd_day1=hund_year*5;
odd_day2=(leap_year*2)+ord_year;
last_day=(odd_day1+odd_day2)%7;

switch(month)
{
  case 1: //jan
     days=1;
```

```
      break;
   case 2://feb
      days=31+1;
      break;
   case 3: //mar
      days=31+28+1;
      break;
   case 4: //apr
      days=31+28+31+1;
      break;
   case 5://may
      days=31+28+31+30+1;
      break;
   case 6://jun
      days=31+28+31+30+31+1;
      break;
   case 7://jul
      days=31+28+31+30+31+30+1;
      break;
   case 8://aug
      days=31+28+31+30+31+30+31+1;
      break;
   case 9://sep
      days=31+28+31+30+31+30+31+31+1;
      break;
   case 10://oct
      days=31+28+31+30+31+30+31+31+30+1;
      break;
   case 11://nov
      days=31+28+31+30+31+30+31+31+30+31+1;
      break;
   case 12://dec
      days=31+28+31+30+31+30+31+31+30+31+30+1;
      break;
 }
if(year%4==0&&month!=1&&month!=2)
   days+=1;

return((last_day+days)%7);

}


/* function for returning day name of given day number*/
char * dayname(int n)
{
      switch (n)
      {
      case 0:
           return("Sunday");
      case 1:
```

```
            return("Monday");
      case 2:
            return("Tuesday");
      case 3:
            return("Wednusday");
      case 4:
            return("Thusday");
      case 5:
            return("Friday");
      case 6:
            return("Saturday");
      }
}

int days_in_month(month,year)
{
 switch(month)
 {
  case 1: //jan
     return(31);
  case 2://feb
     if(year%4==0)
      return(29);
     else
      return(28);
  case 3: //mar
     return(31);
  case 4: //apr
     return(30);
  case 5://may
     return(31);
  case 6://jun
     return(30);
  case 7://jul
     return(31);
  case 8://aug
     return(31);
  case 9://sep
     return(30);
  case 10://oct
     return(31);
  case 11://nov
     return(30);
  case 12://dec
     return(31);
 }
}

char* monthname(month)
{
 switch(month)
```

```
 {
  case 1: //jan
     return("JANUARY");
  case 2://feb
     return("FEBRUARY");
  case 3: //mar
     return("MARCH");
  case 4: //apr
     return("APRIL");
  case 5://may
     return("MAY");
  case 6://jun
     return("JUNE");
  case 7://jul
     return("JULY");
  case 8://aug
     return("AUGUEST");
  case 9://sep
     return("SEPTEMBER");
  case 10://oct
     return("OCTOBER");
  case 11://nov
     return("NOVEMBER");
  case 12://dec
     return("DECEMBER");
 }
}


void display(int day,int month, int year)
{
  int total_day,days[42],i,j,r=16,c=20,count=0;   //days of
the month
  clrscr();
  gotoxy(25,2);
  printf("*******  CALENDER  ******");
  gotoxy(30,8);
  printf("%s: %d",monthname(month),year);
  gotoxy(15,40);
  printf("Developed By: Lokesh Rathore Mob:94250-34034");
  textcolor(RED);
  gotoxy(20,15);cprintf("SUN");
  gotoxy(25,15);printf("MON");
  gotoxy(30,15);printf("TUE");
  gotoxy(35,15);printf("WED");
  gotoxy(40,15);printf("THU");
  gotoxy(45,15);printf("FRI");
  gotoxy(50,15);printf("SAT");

  total_day=days_in_month(month,year);
```

```
  for(i=0;i<day;i++)
     days[i]=NULL;
  for(j=1; j<=total_day;j++)
       days[i++]=j;
  for(;i<42;i++)
       days[i]=NULL;
  for(i=0;i<42;i++)
   {
    if(i%7==0)
     {
      r+=2;
      count=0;
     }
     if(days[i]!=NULL)
     {
      gotoxy(20+(count*5),r);
      if(i%7==0) cprintf(" %2d ",days[i]);
      else   printf(" %2d ",days[i]);
     }
     count++;
  }
}

void main()
{
int year,month=1,day;
char ch;
do
{
 clrscr();
 printf("\nEnter month number:");
 scanf("%d",&month);
 printf("\n Enter year:");
 scanf("%d",&year);
 day=first_day(month,year);
 display(day,month,year);
 gotoxy(25,50);
 printf("\nPress Y key to continue:");
 fflush(stdin);
 ch=getche();
 if(ch=='Y' || ch=='y')
   continue;
 else
   break;
}while(1);
}
```

```
/* Convert Decimal to Binary, Hexa_Descimal and octal*/

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


void oh_b(int *n,int count,int pair)
{
 int arr[40],t[4],i=0,j=0,sum=0;

 for(;count>=0;count--)
   {
      t[i++]=*(n+count)%2;

     }
   }
 count=j-1;

 for( ;count>=0;count--)
   {
    switch (arr[count])
    {
     case 10:
          putch('A');
          break;
     case 11:
          putch('B');
          break;
     case 12:
          putch('C');
          break;
     case 13:
          putch('D');
          break;
     case 14:
          putch('E');
          break;
     case 15:
          putch('F');
          break;
     default:
          printf("%d",arr[count]);
   }
}
}
void main()
{
int arr[20],i=0,count=0;
char ch;
clrscr();
```

```
gotoxy(10,2);
printf("**** CONVERSION: BINARY TO OCTAL & HEXA-DECIMAL
****");
printf("\n\nYou can perform:- \
     \n\n\n Binary to Octal and Hexadecimal \
     \n\nEnter Your Binary number:");
while((ch=getche())!=13)
{
  if (ch<48 || ch>49)
    {
     printf("\nNot valid Binary digit");
     getch();
     exit(0);
    }
  if (ch=='0') arr[i++]=0;
  if (ch=='1') arr[i++]=1;
}
count=i-1;

printf("\n\nOctal Equivalent="); bi_oh(arr,count,3);
getch();

printf("\n\nHexa-Decimal Equivalent="); bi_oh(arr,count,4);
getch();

}
```

/* Convert Decimal to Binary, Hexa_Descimal and octal*/

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


void bi_oh(int *bi,int count,int pair)
{
 int arr[20],i=0,j=0,sum=0;

 for(;count>=0;count--)
   {
     sum+=*(bi+count)*pow(2,i++);
     if(i==pair||count==0)
     {
      arr[j++]=sum;
      i=0; sum=0;
     }
   }
 count=j-1;

 for( ;count>=0;count--)
   {
    switch (arr[count])
```

```
     {
      case 10:
           putch('A');
           break;
      case 11:
           putch('B');
           break;
      case 12:
           putch('C');
           break;
      case 13:
           putch('D');
           break;
      case 14:
           putch('E');
           break;
      case 15:
           putch('F');
           break;
      default:
           printf("%d",arr[count]);
    }
}
}


void main()
{
int arr[20],i=0,count=0;
char ch;
clrscr();
gotoxy(10,2);
printf("**** CONVERSION: BINARY TO OCTAL & HEXA-DECIMAL
****");
printf("\n\nYou can perform:- \
     \n\n\n Binary to Octal and Hexadecimal \
     \n\nEnter Your Binary number:");
while((ch=getche())!=13)
{
  if (ch<48 || ch>49)
     {
      printf("\nNot valid Binary digit");
      getch();
      exit(0);
     }
  if (ch=='0') arr[i++]=0;
  if (ch=='1') arr[i++]=1;
}
count=i-1;

printf("\n\nOctal Equivalent="); bi_oh(arr,count,3);
```

```
getch();

printf("\n\nHexa-Decimal Equivalent="); bi_oh(arr,count,4);
getch();

}
```

/* Convert Binary, Hexa_Descimal and octal to Decimal*/

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

void bi_dec()
{
int arr[10],i=0,j,count=0,deci;
char ch;
clrscr();
printf("\nEnter any Binary Number:");
while((ch=getche())!=13&&i<10)
 {
  if (ch<48 || ch>49)
   {
    printf("\nNot Enter Binary Digits");
    getch();
    return;
   }
   if (ch==48) arr[i]=0;
   if (ch==49) arr[i]=1;
  i++;
  count++;
 }

deci=0;
for(i=i-1,j=0;i>=0;i--,j++)
  deci=deci+(arr[i]*pow(2,j));
printf("\nDecimal Value=%d",deci);
}

void oct_dec()
{
int arr[10],i=0,j,count=0,deci;
char ch;
clrscr();
printf("\nEnter any Octal Number up to 10 digits:");
while((ch=getche())!=13&&i<10)
 {
  if (ch<48 || ch>55)
   {
    printf("\nNot Enter Octal Digits");
    getch();
    return;
```

```
    }
    switch(ch)
    {
     case 48:
         arr[i]=0;
         break;
     case 49:
         arr[i]=1;
         break;
     case 50:
         arr[i]=2;
         break;
     case 51:
         arr[i]=3;
         break;
     case 52:
         arr[i]=4;
         break;
     case 53:
         arr[i]=5;
         break;
     case 54:
         arr[i]=6;
         break;
     case 55:
         arr[i]=7;
         break;
    }
   i++;
   count++;
  }
}
void hex_dec()
{
int arr[10],i=0,j,count=0,deci;
char ch;
clrscr();
printf("\nEnter any Hexa Decimal Number up to 10 digits:");
while((ch=getche())!=13&&i<10)
 {
  if ((ch>=48 && ch<=57)||(ch>=65&&ch<='F'))
   {
    switch(ch)
    {
    case 48:
         arr[i]=0;
         break;
    case 49:
         arr[i]=1;
         break;
    case 50:
```

```
        arr[i]=2;
        break;
    case 51:
        arr[i]=3;
        break;
    case 52:
        arr[i]=4;
        break;
    case 53:
        arr[i]=5;
        break;
    case 54:
        arr[i]=6;
        break;
    case 55:
        arr[i]=7;
        break;
    case 56:
        arr[i]=8;
        break;
    case 57:
        arr[i]=9;
        break;
    case 65:
        arr[i]=10;
        break;
    case 66:
        arr[i]=11;
        break;
    case 67:
        arr[i]=12;
        break;
    case 68:
        arr[i]=13;
        break;
    case 69:
        arr[i]=14;
        break;
    case 70:
        arr[i]=15;
        break;
   }
i++;
count++;
}
else
 {
  printf("\nNot Valid Hexadecimal Digits");
  getch();
  return;
 }
```

```
 }
 deci=0;
 for(i=i-1,j=0;i>=0;i--,j++)
   deci=deci+(arr[i]*pow(16,j));
 printf("\nDecimal Value=%d",deci);
}



void main()
{
int ch;
clrscr();
gotoxy(10,2);
printf("**** CONVERSION: BINARY,OCTAL & HEXA-DECIMAL TO
DECIMAL ****");
printf("\n\nYou can perform following conversion\
     \n\n1:Binary to Decimal\
     \n\n2:Octal to Decimal\
     \n\n3:Hexa-Decimal to Decimal\
     \n\nEnter Your Choice:");
scanf("%d",&ch);
switch(ch)
{
 case 1:
     bi_dec();
     break;
 case 2:
     oct_dec();
     break;
 case 3:
     hex_dec();
     break;
 default:
     printf("\nMade Wrong Choice");
}
getch();
}
/* Convert Decimal to Binary, Hexa_Descimal and octal*/

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


void dec_boh(int deci,int base)
{
 int arr[20],i=0,count=0;

 while(deci!=0)
 {
```

```
  arr[i++]=deci%base;
  deci/=base;
 }
 count=i-1;
 for( ;count>=0;count--)
   {
    switch (arr[count])
    {
     case 10:
          putch('A');
          break;
     case 11:
          putch('B');
          break;
     case 12:
          putch('C');
          break;
     case 13:
          putch('D');
          break;
     case 14:
          putch('E');
          break;
     case 15:
          putch('F');
          break;
     default:
          printf("%d",arr[count]);
   }
}
}


void main()
{
int deci;
clrscr();
gotoxy(10,2);
printf("**** CONVERSION: DECIMAL TO BINARY,OCTAL & HEXA-
DECIMAL  ****");
printf("\n\nYou can perform:- \
     \n\n\n Decimal to Binary,Octal and Hexadecimal \
     \n\nEnter Your decimal number:");
scanf("%d",&deci);
printf("\n\nBinary Equivalent="); dec_boh(deci,2);
getch();

printf("\n\nOctal Equivalent="); dec_boh(deci,8);
getch();

printf("\n\nHexaDecimal Equivalent="); dec_boh(deci,16);
```

```
getch();

}
```