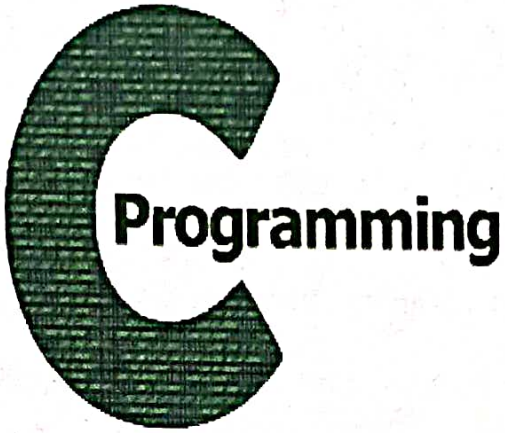
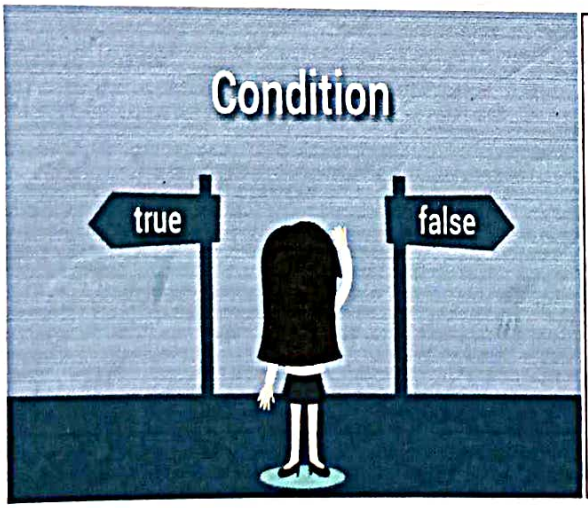
RADIANT
SOFTWARE SYSTEMS

D9, 3RD FLOOR,
GOEL COMPLEX,
LAXMI NAGAR, DELHI-92
M-9999161644
9899112639
WEB-www.radiant.org.in

[A Premier Institute For IGNOU BCA/MCA Coaching]



MCS-011: PROBLEM SOLVING & PROGRAMMING IN 'C'




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

{
 clrscr();
 printf("I am a new programmer");
 getch();

}
```

PROGRAM- A **computer program** is a collection of instructions that performs a specific task when executed by a **computer**. A **computer** requires **programs** to function, and typically executes the **program's** instructions in a central processing unit.

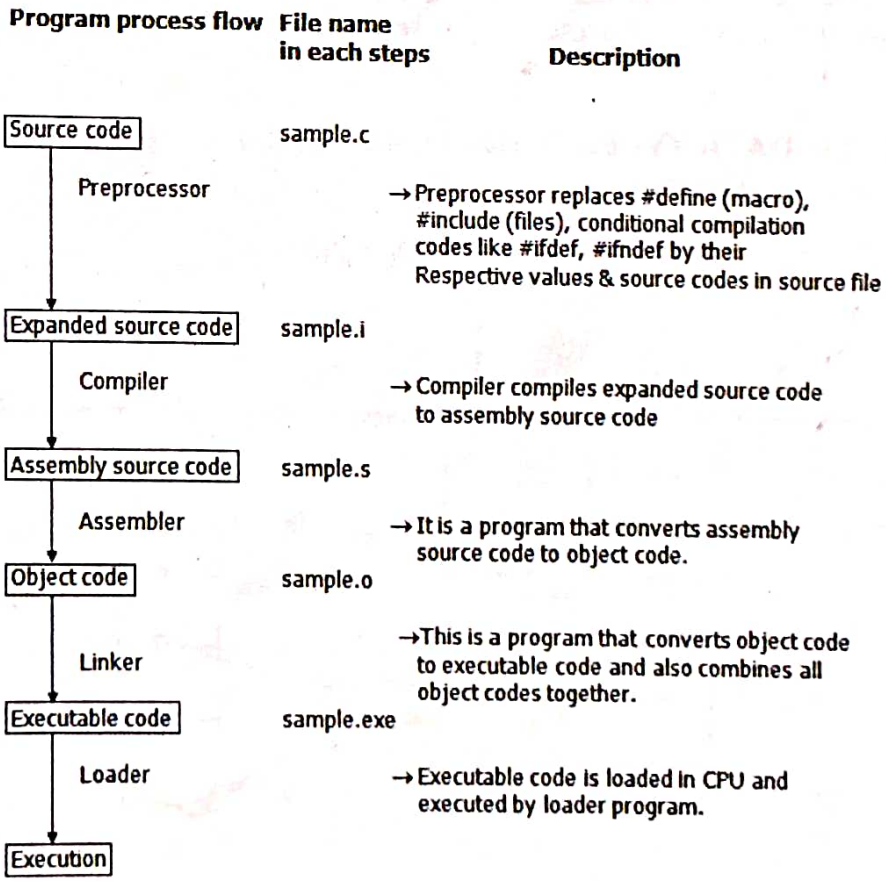
Software- A **software** is a collection of instructions that enable the user to interact with a computer, its hardware, or perform tasks. Without software, computers would be useless. For example, without your Internet browser, you could not surf the Internet.
Example:Tally, Operating System, MS-Word

Source Program

Source program is set of instruction of the high level language used to code a problem to find its solution is referred to as source program.

Object program

If there is no error in program the source programs transformed into the machine language program called object program.
A program in C language involves into different processes. Below diagram will help you to understand all the processes that a C program comes across.



Syntax of C language program

```
1. #include <stdio.h>
2. Main ( )
3. {
4. // Code
5.
6. }
```

Explanation of the Syntax:

# (Hash)	:	Pre-processor directive
Stdio.h	:	It contains the definition of all terms used in program.
Include	:	It includes/attaches the header file with program.
Compiler	:	It is used in c Language as a translator

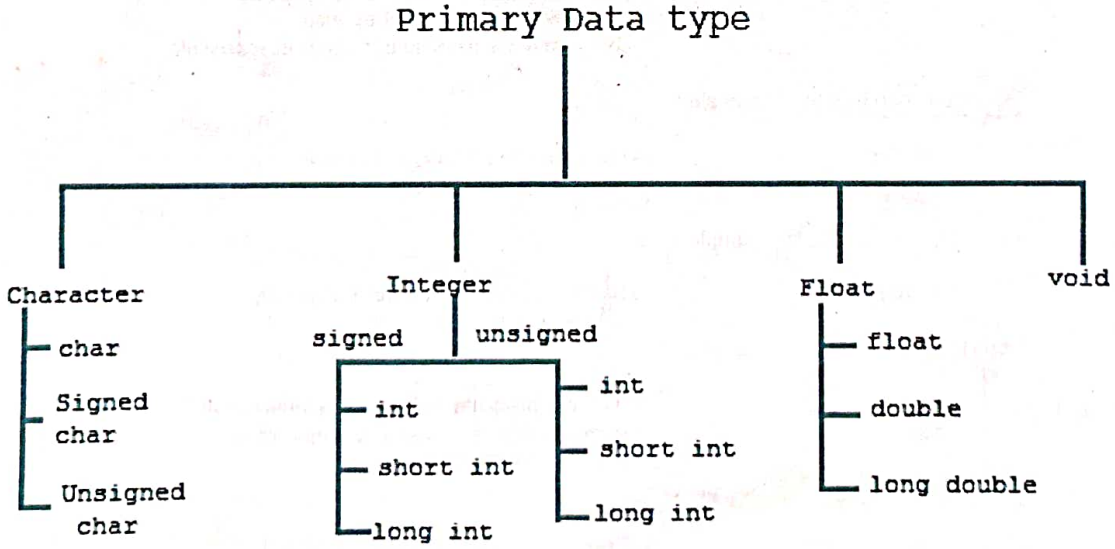
Program→translator→object code

DATA TYPES : It is used to define the type of data to be stored in the variables

Types of Datatypes:-

- **PRIMARY DATA TYPES -**These are fundamental data types in C namely integer(**int**), floating(**float**), charater(**char**) and **void**.

- **DERIVED DATA TYPES-**Derived data types are like arrays, functions, structures and pointers.



Types of Primary DataTypes:

a).INTEGER TYPE -Integers are used to store whole numbers.

Size and Range of Integer type

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

---

b).FLOATING TYPE -Floating types are used to store real numbers.

Size and range of Integer type

Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
Double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

c).CHARACTER TYPE -Character types are used to store characters value.

Size and range of Integer type

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

d).VOID TYPE - Void type means no value. This is usually used to specify the type of functions.

VARIABLE

Variables are simply names used to refer to some location in memory – a location that holds a value with which we are working. It may help to think of **variables** as a placeholder for a value. You can think of a**variable** as being equivalent to its assigned value

Types of variable

The variable can be characterised by their data type and by their storage classes. Data type means the type of value represented by a variable and the storage class means the scope of a variable within a program.
There are four types of storage classes:-

- **Automatic variables/storage classes-**

These are declared inside a function where they are used. They have local scope within the function and destroyed automatically when the function ended.

- **External variables/storage classes-**

These variables are called global variables and are declared outside the function. They can be accessed by any function in the program.

- **Static variables/storage classes-**

A static variable is initialised only once when the program is compiled. It can be either automatic or external. It is mostly used in recursion.

- **Register variable/storage classes-**

A value stored in CPU register can always be accessed faster than main memory. So, we can tell a compiler that a variable should be kept in register in place main memory.

Basic I/O in C:

- **Printf() -** It is used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen. We use **printf**() function with %d format specifier to display the value of an integer variable.
- **Scanf()-**This is the function which can be used to to read an input from the command line.

DESIGN METHODOLOGIES:

A). TOP DOWN DESIGN/APPROACH

If a design methodology that proceeds from the highest level to the lowest level and from the general to the particular, and that provide a formal mechanism for breaking complex process design into functional description, reviewing progress and allowing modification.

Top down design provides the way of handling the logical complexity and detail encountered in computer algorithm. It allows building solutions to problems in step by step.

Top-down design suggests taking the general statement about the solution one at a time, and then breaking them down into a more precise subtask/sub-problem. These sub-problems should more accurately describe how the final goal can be reached. The process of repeatedly breaking a task down into a subtask and then each subtask into smaller subtasks must continue until the sub-problem can be implemented as the program statement. With each spitting it is essential to define how sub-problems interact with each other.

B).BOTTOM UP DESIGN/APPROACH

Bottom up Desgin begins the design at the lowest level module or subsystem and progresses upwards to the design of the main program, main module or main subsystem.

Difference b/w Top Down Design & Bottom up Design

	Top Down Design/Approach	Bottom Up Design Approach
1.	In this approach an overview of the system is first formulated, specifying but not detailing any first level subsystem. Each subsystem is than refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduce to base element.	In this approach the individual base element of the systems are first specified in greater details. These elements are than linked, sometimes in many levels, until a complete top down system is formed.
2.	It proceeds from the abstract entity to get to concrete design	It proceeds from the concrete design to get to abstract entity
3.	It is most often used in designing brand new system.	It is sometimes used when is reverse engineering a design.
4.	It is begins the design with the main or top level module and to the lowest level module or subsystem.	It is begins the design with the lowest level module or subsystem and progresses upward to the main program module or subsystem and progress upward to the main program module or subsystem.

ALGORITHM

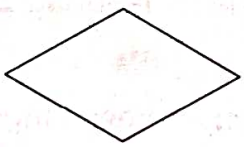
An Algorithm is a finite set of steps defining the solution of a particular problem. An algorithm is expressed in pseudo code- something resembling C language or Pascal, but with some statements in English rather than within the programming language.

FLOWCHART

Flowcharts are used in programming to diagram the path in which information is processed through a computer to obtain the desired results.
Flowchart is a graphical representation of an algorithm. It makes use of symbols which are connected among them to indicate the flow of information and processing. It will show the general outline of how to solve a problem or perform a task. It is prepared for better understanding of the algorithm.

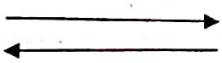
- Basic symbols used in flowchart design are-

Start/Stop

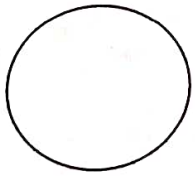


Question, Decision

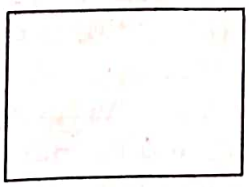
Input/Output



Lines or arrows represent the direction of the flow of control



Connector (connect one part of the flowchart to another)



Process Instruction

Q. Why C is called middle level language?

A. C is a general purpose, structured programming language. Among the two types of programming languages (high level & low level) C lies in b/w two categories. C is called middle level language because it combines the elements of high level language with the functionality of assembly language. It provides relatively good machine efficiency as compared to high level language.
As a middle level language, 'C' allows the manipulation of bits, bytes and address-the basic elements with which the computer executes the inbuilt and memory management functions.

'C' code is very portable, that it allows the same 'C' program to be run on machines with different hardware configurations. The flexibility of C language allows it to be used for systems programming as well as for application programming.

Types of programming language

I. **Low Level Languages or Machine Oriented Languages: -** The language whose design is governed by the circuitry and the structure of the machine is known as the machine language. This language is difficult to learn & use. It is specific to a given computer and is different for different computers i.e. these languages are machine-dependent.

These languages have been designed to give a better machine efficiency, i.e. faster program execution. Such languages are also known as Low level languages. Another type of Low-Level language is Assembly language.

II. **High Level Languages or Program Oriented Languages: -** These languages are particularly oriented towards describing the procedures for solving the problem in a concise, precise and unambiguous manner. Every high level language follows a precise set of rules. These are developed to allow application programs to be run on a variety of computers i.e. these languages are Machine independent.
eg:- FORTRAN,BASIC etc.

These are easy to learn and programs may be written in these languages with much less effort.

The main drawback of these languages is that the computer can't understand them and they need to be translated into machine language with the help of other programs known as compilers or translators.

Q. - Why C is called structured Language?

A. - C is commonly called a structured language because of structural similarities to ALGOL and Pascal. Structured language is one that divides the entire program into modules using top down approach where each module executes one job or task. it is easy for debugging, testing and maintenance if a language is a structured one.
C supports several control structures such as while, do while and for and various date structures such as strucs, files, arrays etc. The structural component of C makes the programming and maintenance easier.

ERROR:

Any mistake done by the programmer in the syntax or in the logic of the program produces an error.

TYPES OF ERROR-There are different types of error in c

1. **Syntax Error-** When the rules of grammar are not followed in program, the program show the error. These errors are called syntax error.

Every language has an associated grammar, and the program written in that language has to follows the rules of that grammar.

C also follows certain syntax rules. When a 'C' program is compiled, the compile will check that the program is syntactically correct. If there are any syntax errors in the program, those will be displayed on the screen with the corresponding ten numbers.
Eg:- program to print a message on the screen.

```
#include <stdio.h>
main()
{
printf ("Hello, how are you in")
error test.C1: No file name ending
error test.C5: Statement missing;
error test C6: Component Statement missing
}
```

Correct Program

```
# include <stdio.h>
main()
{
printf("Hello,how are you\n");
}
```

2. **Semantic Error:** - The errors that are shown white compilation and displayed as warnings, called semantic errors. These errors are shown if a particular statement has no meaning. The program does compile with these errors, but it is always advised to correct them also, since they may create problems while execution.

   The example of such error is that when a variable is declared in a program but have not used it, the compiler show a warning "code has no effect" these variables are unnecessarily occupying the memory.

3. **Linker error:-** If a program contain syntax error then the program does not compile, but it may happen when the program compiles successfully but we are unable to get the executable file, this happens when there are certain linker errors in the program. For example, the object code of certain standard Libra the definition for this function is present in the header file that is why we don't get a compiler error, such kind of errors are called linker errors. The executable file would be created successfully only if these linker errors are corrected.

4. Logical Error:- The error which are occur due to some logical mistake in program are called logical error.
Some times after compiling the program executes and gives wrong result, it means there are some logical errors in our program.

Example:
```
#include <stdio.h>
main()
{
int  a=2, b=5 avg;
printf ("%d%d",a.b);
avg =(a+b)/2;
printf ("%d",avg);
 }
```

5. Run time Error:- Some time when we compile program the program does not execute completely & abort in b/w it is due to run time error.
The error which are detected during execution are called run time error.
These error don't produce the result at all, the program execution stop in b/w & run time error message is flashed on the screen.

Example:-
```
#include <stdio.h>
main ( )
{
 int a=10, b=10, c;
 printf ("%d%d,a.b);
 C= (a+b/(a-b);
 Printf ("%d,c);
 }
```
Output => divide by zero error

EXPRESSIONS AND OPERATORS

Operators

Operators are symbols which take one or more operands or expression and perform Logical & Arithmetical computation.
In other words operators perform an operation on one or more operands.

Types Of Operators:

1. **Arithmetic operator:-** The operators which are used to perform arithmetical operation are called arithmetic operators.

   Q. Program to add 2 nos.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modular Division

2. **Logical Operator:** - These are used to evaluate expressions which may be true or false. Expressions which involve logical operations are evaluated and found to be one of two values: true or false.A logical operator combine the result of two conditions component to prduce a single result based on them or inward to the result of a single condition.

   eg:=> && => true on both conditions
   !! => true on any one condition
   ! => if condition is false then answer is true.

Q. Program to check the given alphabet is vowel or consonant.

3. **Relational Operator:-** The operator through which we can compare two variable in the program are called Relational operators. It use if control Structure that allows a program to make a decision based on the result of some condition. If the condition is true then the statement in the body of if is executed else if the condition is false, the statement is not executed.

Relational operation	Condition	Meaning
==	x==y	x is equal to y
!=	x!=y	x is not equal to y
<	x<y	x is less than y
<=	x<=y	x is less than or equal to y
>	x>y	x is greater than y
>=	x>=y	x>=y x is greater than and

equal to y

Q-Program to find greater no. b/w two nos.

4. **Assignment Operator-** The function of this operator is to assign the values or values in a variable on right hand side of an expression to variables on the left hand side.

   Q. program to swap two values.

5. **Conditional Operator-** The conditional operator evaluate an expression returning a value if that expression is true & a different one if the expression is evaluated as false.
   the syntax is as follows:-
   (condition) ? (expression) : (expression 2)
   example:- (a>b)? printf ("1st no. greater");
   printf ("2nd no. greater");

6. **Comma Operator-** A Comma operator is used to separate a pair of expression. A pair of expression separated by a comma is evaluated left to right, and the type and value of the result are the value of the type & value of the right operands. All side effects from the evaluation of the left oprand are completed before beginning evaluation of the right operand. The left side comma operator is always evaluated to void. This means that the expression on the right hand side becomes the value of total comma separated expression.

   Eg:- x= (y=2, y-1) ⇒ 1
   a= (b=3, b+2) ⇒ 5

CONDITIONAL STATEMENT

A conditional statement is a feature of programming language, which allow it to perform actions depending upon some conditions provided by the programmer. Conditional statement control the sequence of statements depending on the condition.

Types of conditional statements:

1. **I**f **s**tatement
2. **I**f-**e**lse-**s**tatement
3. **E**lse **i**f **s**tatement
4. **N**ested **i**f-else
5. **S**witch **c**ase
6. **G**oto **s**tatement

1. **If statement:** This is the most simple form of the branching statements.It takes an expression in parenthesis and an statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

   NOTE: Expression will be assumed to be true if its evaulated values is non-zero.

if statements take the following for

```
if (expression)
  statement;

or

if (expression)
  {
    Block of statements;
  }
```

2. **If–else statement-----**C language also let one choose between two statements by using the if-else structure
   This is the most simple form of the branching statements.
   It takes an expression in parenthesis and an statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

   NOTE: Expression will be assumed to be true if its evaulated values is non-zero.

```
if (expression)
  statement;
else
statement

or

if (expression)
  {
    Block of statements;
  }
Else
{
    Block of statement;
}
```

3. **Else-if statement—**The else if statement in C generally used when we need to compare more than two option.

```
if (expression)
  {
    Block of statements;
  }
else if(expression)
  {
    Block of statements;
  }
else
  {
    Block of statements;
  }
```

4. **Nested if-else-----**It is conditional statement which is used when we want to check more than one condition at a time in program.

```
if (expression)
  {
   if(expression)
    {
     statements;
    }
   else
    {
     Statement;
    }
```

```
 }

else
 {
 if(expression)
 {
  statements;
 }
else
 {
  statements;
 }
}
```

5. Switch Case: This is multiple or multiway branching decision making statement. When we use nested if-else statement to check more than one condition than the complexity of a program increase in the case of lot of condition. Thus the program difficult to read and maintain. So to overcome this problem, C provide 'switch case'.

Switch case check the value of expression against the case value, if condition matches the case value of a expression against a case value,if conditon matches a case value, then the control is transferred to the point.

```
switch( expression )
   {
     case label1:      statements1;
               break;
     case label2:      statements2;
               break
     case label3:      statements3;
               break
     default :  statements4;
   }
```

Rules for declaring the switch case :

- The case label should be integer or character constant.
- Each compound statement of switch case should contain break statement to exit from case.
- Case label must end with (:) colon.

Advantages of switch case :

- ✓ It is easy to use.
- ✓ It is easy to find out error.
- ✓ Debugging is made easy in switch case.
- ✓ Complexity of a program is minimized.

Limitation of switch case

- ✓ Logical operator can not be used with switch case.
- ✓ Switch case variable can have only 'int' or 'char' data types. So 'float' or no other data types allowed.

6. Goto Statement

The goto is a unconditional branching statement used to transfer control of the program from one statement to another. It is called jumping statement because it force the progam sequence to goto another place in the program.

Syntax

goto label :

label: Statement ;

The goto statement works in two ways:-

- **goto forward/downward jump**
- **goto backward/upward jump**

Forward Jump :If the label: is placed after the goto label:, some statement will be skipped & the jump is known as forward jump.

Syntax:

goto label:
label: statement ;
Example:

```
# include <stdio.h>
main ()
{
int i=2;
While (1)
{
 printf ("%d", i);
 i=i+2;
   if (i>=20)
 goto outside ;
 }
 outside : printf ("over");
 }
```

Backward Jump:- If the label: is before the statement goto label: , a loop will be formed & some statement will be executed repeatedly. Such a jump is known as backward jump.

Example-

```
#include <stdio.h>
main( )
{
 int n=1;
 out : printf ("%d" , n);
 n++;
 if (n<11)
 {
   goto out;
  }

 printf ("program over")
}
```

CONTINUE STATEMENT

The continue statement can be used to skip the rest of the body of all iterated loop. The use of the continue statement violates the rules of structured Programming. However, a section of code which use a continue statement a section of code which use a continue statement can always be rewritten to omit the continue.

example →

```
# include <stdio.h>
main( )
{
  int i ;
  for (i=1; i<=10;i++)
  if (i%5==0)
  continue;
  printf (%d",i);
  }
```

BREAK STATEMENT

The Break statement is used to terminate the execution of the nearest enclosing do, for, while, switch case. Switch statement in which it appears,control passes to the statement that follow the terminated statement.

Q.Program to check the any member is prime or not.

LOOPS

Loops provide a way to repeat commands and control how many times they are repeated. C provides a number of looping way.

- **While loop**

The most basic loop in C is the while loop.A while statement is like a repeating if statement. Like an If statement, if the test condition is true: the statments get executed. The difference is that after the statements have been executed, the test condition is checked again. If it is still true the statements get executed again.This cycle repeats until the test condition evaluates to false.

Basic syntax of while loop is as follows:

```
while ( expression )
{
  Single statement
  or
  Block of statements;
}
```

- **For loop**

For loop is similar to while, it's just written differently. for statements are often used to process lists such a range of numbers:

Basic syntax of for loop is as follows:

```
for( expression1; expression2; expression3)
{
  Single statement
  or
  Block of statements;
}
```

In the above syntax:

- expression1 - Initialises variables.
- expression2 - Conditional expression, as long as this condition is true, loop will keep executing.
- expression3 - expression3 is the modifier which may be simple increment of a variable.

- **Do-While loop**

Do-While is just like a while loop except that the test condition is checked at the end of the loop rather than the start. This has the effect that the content of the loop are always executed at least once.

Basic syntax of do-while loop is as follows:

```
Do
{
  Single statement
  or
  Block of statements;
}while(expression);
```

ARRAY

Array is a contiguous memory allocation of similar/same types of data. It is faster than declaring variables because variables are stored & fetched into the memory in random order but arrays stored/fetched from memory in continous manner.

Types of Array:

- Single Dimension array
- Double Dimension array
- Multi Dimension array
- Character array

Q - WAP to declare a 1-D array.

```
#include<stdio.h>
#include<conio.h>
main()
{
int i;
int s[]={12,24,36} ;
int k[3]={15,30,45};
clrscr();
for(i=0;i<=2;i++)
{
 printf("\n%d",s[i]);
}

for(i=0;i<=2;i++)
{
 printf("\n%d",k[i]);
}

getch();
}
```

//WAP to declare an 2D-array.

```
#include<stdio.h>
#include<conio.h>
main()
{
int i,j;
int s[3][3]={12,24,36,48,60,72 ,84,96,108};
clrscr();
for(i=0;i<=2;i++)
{
 for(j=0;j<=2;j++)
 {
 printf("\t%d",s[i][j]);
 }
```


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

//WAP to declare an 3D-array/Multi-Dimensional array

```
#include<stdio.h>
#include<conio.h>
main()
{
int i,j;
int s[3][3]={
          {12,24,36},
          {48,60,72} ,
          {84,96,108}
          };
 clrscr();
 for(i=0;i<=2;i++)
 {
  for(j=0;j<=2;j++)
  {
  printf("\t%d",s[i][j]);
  }
  printf("\n");
 }
 getch();
 }
```

```
//WAP to declare an 3D-array.
#include<stdio.h>
#include<conio.h>
main()
{
int i,j,k;
int s[3][3][3]={
          {
          {12,24,36},
          {48,60,72} ,
          {84,96,108},
          },
          {
          {2,4,6},
          {8,10,12},
          {14,16,18},
          } ,
           {
          {5,10,15},
```

```
        {20,25,30},
        {35,40,45},
         } ,
        };

clrscr();
for(i=0;i<=2;i++)
{
 for(j=0;j<=2;j++)
 {
 for(k=0;k<=2;k++)
 {
 printf("\t%d",s[i][j][k]);
 }
 printf("\n");
}
printf("\n");
}
getch();
}
```

STRING :

Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

STRING FUNCTIONS

1. **Strlen() :** It is used to find the length of any string.
   Syntax: l=strlen(a);

```
//WAP to find the length of a  string .
#include<stdio.h>
#include<conio.h>
main()
{
int i,p=0;
char s[40], t[40];
clrscr();
printf("enter string");
gets(s);
p=strlen(s);
printf(" The length of string=%d",p);
printf(".");
getch();}
```

2. **Strcmp**(): It is used to compare two strings are equal or not.
   Syntax: x=strcmp(s1,s2);

```
//WAP to compare two strings.
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
int i;
char s[40], p[40];
clrscr();
printf("enter first string===");
gets(s);
printf("enter the 2nd string===");
gets(p);
if(strcmp(p,s)==0)
{
```

```
printf("  stringis equal");
}
else
{
printf("string is not equal");
}
getch();
}
```

3. Strcat() : It is used to merge two string.

Syntax: strcat(s1,s2);

```
//WAP to concatenate the source string into destination string .
#include<stdio.h>
#include<conio.h>
main()
{
int i;
char s[40], p[40];
clrscr();
printf("enter string");
gets(s);
gets(p);
strcat(p,s);
printf(" The concatenate string=%s",p);
printf(".");
getch();
}
```

4. Srtupr():It is used to change a string in upper case.
Syntax: strupr(s);

```
//WAP to change a string in upper case.
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
int i;
char s[40], p[40];
clrscr();
printf("enter string");
gets(s);
strupr(s);
printf(" The string=%s",s);
printf(".");
}
```

5. **Srtlwr():**It is used to change a string in lower case.
   Syntax: strlwr(s);

```
//WAP to change a string in lower case.
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
int i;
char s[40], p[40];
clrscr();
printf("enter string");
gets(s);
strlwr(s);
printf(" The string=%s",s);
printf(".");
getch();
}
```

6. **Strrev**(): It is used to reverse a string .
   Syntax: strrev(s);

```
//WAP to  reverse a string.
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
int i;
char s[40], p[40];
clrscr();
printf("enter string");
gets(s);
strrev(s);
printf(" The reverse string=%s",s);
printf(".");
getch();
}
```


FUNCTION

A Function is a self contained block of executable code that can be called from any other function.

Functions are very important tool for modular programming, where we break very large programs into small subprograms/module/functions. The function reduces the complexity and make programming simple and easy to understand.

Function Prototype-Function Prototype required that every function which is to be accessed should be declared in calling function. If we will not declare function prototype in the calling function the "c" compiler give an error.

Example:

```
#include<stdio.h>
main()
{
int n,sq;
int  square(int); ------>        Function prototype
printf("enter number");
scanf("%d", &n);
sq=square(n); ---------->        Function call
printf("%d",sq);
}

int square(int k)   ------>      Passing argument
{
int result;
result=k*k;
return(result);
}
```

TECHNIQUES OF INVOKING/CALLING FUNCTION

a. **Call By Value-** In this technique the copied of the of the actual argument passed passed to the function. So, if any changes are done in formal argument, don't effect the actual argument.

Example

Q:program to print the sum of two numbers.

```
#include <studio.h >
#include<conio.h>
sum (int, int );
main()
{
int a,b;
printf("enter two numbers")
scanf("%d%d & a,& b);
Sum(a,b);
printf("%d%D", a,b);
```

```
getch();
}

sum (int k, int t)
{
int c;
C=K+t;
print f ("%d",c);
}
```

b. Call By Reference- In this technique the address of actual argument passed to the called function. So, if any changes are done in formal argument it will effect the actual argument.
Example

Program to print the sum of two numbers.

```
#include <stdio.h>
#include<conio.h>
sum (int*,int*);
main ()
{
int a,b;
print f ("enter two numbers")
scanf ("%d%d, & a,&b);
sum( &s, &b);
getch();
}
sum (int*k, int*t)
{
C= (*k)+(*t) ;
printf ("%d",c);
}
```

RECURSION

It is a technique in which we can create a loop like situation without using loop. it can't be implemented function. It is called recursion because in this technique a called function call itself and this function is known as recursive function.

Q. program to print table of any number using recursion

```
#include<stdio.h>
#include<conio.h>
table(int);
getch();
}
table (int k)
{
static int c,i=1;
if(k=10)
{
c=K*i;
printf("%d",c);
i++;
table(k);
}
}
```

Q. Write a program to print factorial of any number using recursion.

Types of function Invoking/Calling-Function invoking depending on argument or parameter and there returning a value. A function can be called in following ways:

- With no argument and with no return value
- With no argument and with return value
- With argument and with no return value
- With argument and with return value.

```
K=sum (a,b)
pf ("%d",K)
}
sum (int a, int b)
{
int c
c=a+b;
return c;
}
```

POINTER

A **pointer** is a variable which contains the address in memory of another variable. We can have a **pointer** to any variable type. The unary or monadic operator & gives the "address of a variable". The indirection or dereference operator * gives the "contents of an object pointed to by a **pointer**".

Pointer to array- A pointer points to an array is called pointer to array. The pointer stores the base address of the array.

```
int main()
{
  int i,class[6],sum=0;
  printf("Enter 6 numbers:\n");
  for(i=0;i<6;++i)
  {
     scanf("%d",(class+i)); // (class+i) is equivalent to &class[i]
     sum += *(class+i); // *(class+i) is equivalent to class[i]
  }
  printf("Sum=%d",sum);
  return 0;
}
```

Imp

Array of Pointer: Array can be as array of pointer like as array of int , array of char, or array of float. As a pointer, a variable always contains an address, an array of pointer is a collection of address. The most common use of pointer of array is used to hold the character strings.For example the following declaration creates an array ,named days of the month, that contains pointer to character strings, because it provides the efficient use of memory.

```
#include<stdio.h>
#include<conio.h>
main()
{
char *days[7]={"Sunday" , "Monday" , "Tuesday" ,"Wednesday" , "thrusday" ,
"Friday" , "Saturday"};
int i;
clrscr();
for(i=0;i<=6;i++)
{
printf("\n%s" ,*(days+i));
}
getch();
}
```

//WAP to access 2-D array elements using pointer.

```
#include<stdio.h>
#include<conio.h>
main()
{
int * mat[3][3]={1,2,3,4,5,6,7,8,9};
int i,j;
clrscr();
for(i=0;i<=2;i++)
{
  for(j=0;j<=2;j++)
  {
  printf("\t%d" ,*(*(mat+i)+j));
  }
  printf("\n");
}
getch();
}
```

STRUCTURE

Array is used to store large set of data and manipulate them but the element stored in array are to be of same type.
So, if we need to use a collection of different data item of different data types we use structures.
Structure is a method of packing data of different types. It is a convenient method of handling a group of related data items of different data types.

Syntax:

```
        struct tag-name
        {
    data type member;
    data type member;
        };

struct tag-name obj1, obj2;
```

The key word struct declare a structure to hold the detail of all fields. These are members of the structures. Each member may belong to different or same data types. The-name can be used to define object that have the tag tag-name structure.

PREPROCESSOR

A preprocessor is a program that processes its input data to produce output that is used as input to another program.The output is said to be a preprocessed form of the input data, which is often used by some subsequent programs like compliers.

Types of pre-processor

1. **#define directive(macro)-**used to define the definition of statement and that never in the program.It is a special type of directive for defining the macro definition in the program. It is multiple purpose directive.

2. **#include directive(source file inclusion)-**This directive helps to include the file to the program. There are two types of file: first file is "built in file" and second file is "user define" file.

   Syntax is #include "file_name" for user defined file but #include<file name> for built in file.

3. **#undef directive-**This directive is used to undefined a define directive. This is used to undefined a macro which has been define earlier #define.
   Syntax: #undef name_of_macro Example:#undef max

4. **#ifdef #endif(conditional directive)-**"#ifdef" directive checks whether particular macro is defined or not. If it is defined, "If" clause statements are included in source file.Otherwise, "else" clause statements are included in source file for compilation and execution.

5. **#ifndef-**#ifndef exactly acts as reverse as #ifdef directive. If particular macro is not defined, "If" clause statements are included in source file.Otherwise, else clause statements are included in source file for compilation and execution.
6. **#if #elif #endif(conditional directive)-**

Example

```
#include<stdio.h>
#define MARKS 71
void main()
{
#if(MARKS >= 70)
    printf("\nDistinction");
#elif((MARKS >= 60)&&(MARKS < 70 )
    printf("\nFirst Class");
#elif((MARKS >= 40)&&(MARKS < 60 )
    printf("\nSecond Class");
#else
    printf("\nFail");
#endif
}
```


7. #pragma-Pragma is used to call a function before and after main function in a C program.

S.no	Pragma command	Description
1	#Pragma startup <function_name_1>	This directive executes function named "function_name_1" before
2	#Pragma exit <function_name_2>	This directive executes function named "function_name_2" just before termination of the program.
3	#pragma warn – rvl	If function doesn't return a value, then warnings are suppressed by this directive while compiling.
4	#pragma warn – par	If function doesn't use passed function parameter , then warnings are suppressed
5	#pragma warn – rch	If a non reachable code is written inside a program, such warnings are suppressed by this directive.

//use of pragma directive

```
#include<stdio.h>
function1() ;
function2();
#pragma startup function1
#pragma exit function2
void main()
{
printf("\nhello");
}
function1()
{
printf("\nindia");
}
function2()
{
printf("\nbye");
}
```

FILE HANDLING/ FILE IO

A **file** represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a readymade structure.

In C language, we use a structure **pointer of file type** to declare a file.

FILE ***fp**;

Through file handling, one can perform operations like create, modify, delete etc on system files.

C provides a number of functions that helps to perform basic file operations. Following are the functions,

Function	Description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

Opening a File or Creating a File

The fopen() function is used to create a new file or to open an existing file.

General Syntax :

*fp = FILE ***fopen**(const char **filename*, const char **mode*);

Here **filename** is the name of the file to be opened and **mode** specifies the purpose of opening the file. Mode can be of following types,

***fp** is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file.

mode	Description
R	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode
rb	opens a binary file in reading mode
wb	opens or create a binary file in writing mode
ab	opens a binary file in append mode
rb+	opens a binary file in both reading and writing mode
wb+	opens a binary file in both reading and writing mode
ab+	opens a binary file in both reading and writing mode

Closing a File

The fclose() function is used to close an already opened file.

General Syntax :

int **fclose**(FILE *fp);

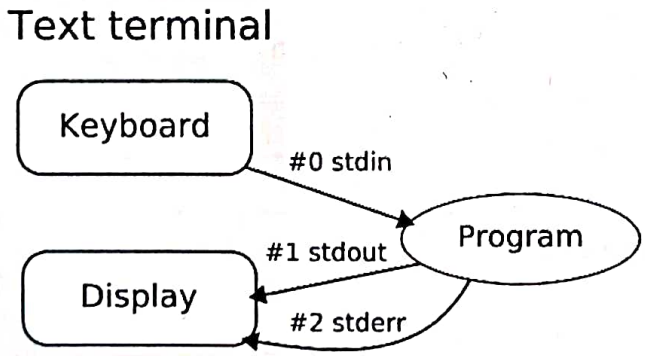
Here fclose() function closes the file and returns **zero** on success, or **EOF** if there is an error in closing the file. This **EOF** is a constant defined in the header file **stdio.h**.

File I/O Streams in C Programming Language :

1. In C all **input and output** is done with streams
2. Stream is nothing but the **sequence of bytes of data or flow of data.**
3. A sequence of bytes flowing into program is called **input stream**
4. A sequence of bytes flowing out of the program is called **output stream**
5. Use of Stream make I/O machine independent.

Predefined Streams :

Stdin	Standard Input
Stdout	Standard Output
Stderr	Standard Error



Standard Input Stream Device :

1. **stdin** stands for (**St**andard **In**put)
2. Keyboard is **standard input device** .
3. Standard input is data **(Often Text) going into a program**.
4. The program requests data transfers by use of the read operation.
5. Not all programs require input.

Standard Output Stream Device :

1. **stdout** stands for (**Stand**ard **Out**put)
2. Screen(Monitor) is **standard output device** .
3. Standard output is data **(Often Text) going out from a program.**
4. The program sends data to output device by using write operation.

Difference Between Std. Input and Output Stream Devices :

Point	Std i/p Stream Device	Standard o/p Stream Device
Stands For	Standard Input	Standard Output
Example	Keyboard	Screen/Monitor
Data Flow	Data (Often Text) going into a program	data (Often Text) going out from a program
Operation	Read Operation	Write Operation

Some Important Summary :

Point	Input Stream	Output Stream
Standard Device 1	Keyboard	Screen
Standard Device 2	Scanner	Printer
IO Function	scanf and gets	printf and puts
IO Operation	Read	Write
Data	Data goes from stream	data comes into stream

Reading and writing function for one character

1. getc()-It is used to read a single character from a file.It is equivalent to getchar().

Syntax

```
getc(in_file);
```

2 .putc()-It is used to write a character to a file identified by its second argument.

Syntax

```
putc(c,out_file);
```

2 .putc()-It is used to write a character to a file identified by its second argument.
Syntax

putc(c,out_file);

Reading and writing one line at a time

1. **fputs()-**This function is used to write one line to a file. To write a string to file is known as fputs(). This function takes two arguments.
   Syntax:
   fputs(string_file , file_pointer);

2. **fgets()-**This function is used to read one line to a file .It reads string file. It takes three arguments.
   Syntax:
   fgets(stringfile, size , filepointer)

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
FILE *p;
char s[20];
clrscr();
p=fopen("my.c","w");
while(strlen(gets(s))>0)
{
  fputs(s,p);
}
fclose(p);
p=fopen("my.c","r");
while(fgets(s,19,p)!=NULL)
{
printf("%c",s);
printf("\n");
fclose(p);
getch();
}
```

Reading and writing one block at a time

1. **fread()---**This function is used to read the data from the disk to be placed in the structure variable. The function fread returns the number of records read
   syntax
   fread(&d ,sizeof(d),1,fp);
   where
   First argument d is a variable of structure.
   Second argument sizeof() operators gives the size of variable in bytes.

Third argument is number of structure that you want to write at one time means you write the structure only one time.
Fourth argument fp is file pointer.

2. **fwrite()---**This function is used to write the data into the disk. This function has the same four argument s to read.
   Syntax
   fwrite(&d ,sizeof(d),1,fp);
   where
   First argument d is a variable of structure.
   Second argument sizeof() operators gives the size of variable in bytes.
   Third argument is number of structure that you want to write at one time means you write the structure only one time.
   Fourth argument fp is file pointer.

Reading and Writing one word at a time

1. **getw()**-It is used to read from file as like as getc but these deals with only integers.
   Syntax
   getw(filepointer);

2. **putw()**---It is used to write from file as like as putc but these deals with only integers.But when I use these for writing integers, it just write different symbol in file(like if i write 65 to file using putw(). It writes A in the file)
   Syntax: Putw(integer value , file pointer);

```
#include<stdio.h>
#include<conio.h>
main()
{
FILE *p;
int val;
int i;
clrscr();
p=fopen("mi.c","w");
for(i=1;i<=5;i++)
{
printf("enter the integer value\n");
scanf("%d",&val);
putw(val,p);
}
fclose(p);
p=fopen("mi.c","r");
while((val=getw(p))!=EOF)
{
printf("%d",val);
printf("\n");
}
```

Random Accessing File

fseek(), ftell() and rewind()

fseek()- It is used to moves the reading control to different positions using fseek function.
ftell()- It tells the byte location of current position in file pointer.
rewind()- It moves the control to beginning of a file.

Program-

```
#include<stdio.h>
void main(){
  FILE *fp;
  int i;
  clrscr();
  fp = fopen("CHAR.txt","r");
  for (i=1;i<=10;i++)
{
     printf("%c : %d\n",getc(fp),ftell(fp));
     fseek(fp,ftell(fp),0);
     if (i == 5)
     rewind(fp);
  }
  fclose(fp);
}
```

fprint and fscanf

These are equivalents to the function printf and scanf which read or write data to a file. These functions are used in same way,except that the fprintf and fscanf take the file pointer as an additional first argument.
fprintf(output file, "this is listing file\n");
fscanf(input_ file,"%d",&counter);

Difference between getc(), getchar(), getch() and getche()

All of these functions read a character from input and return an integer value. The integer is returned to accommodate a special value used to indicate failure. The value EOF is generally used for this purpose.

1. **getc():**-It reads a single character from a given input stream and returns the corresponding integer value (typically ASCII value of read character) on success. It returns EOF on failure.

Syntax:
int getc(FILE *stream);
Example:

```
// Example for getc() in C
#include <stdio.h>
int main()
{
```

```
  printf("%c", getc(stdin));
  return(0);
}
```

Run on Console
Input: g (press enter key)
Output: g

2. getchar():-The difference between getc() and getchar() is getc() can read from any input stream(file), but getchar() reads from only standard input(console). So getchar() is equivalent to getc(stdin).
Syntax:

```
int getchar(void);
```

Example:

```
// Example for getchar() in C
#include <stdio.h>
int main()
{
  printf("%c", getchar());
  return 0;
}
```

Run on Console
Input: g(press enter key)
Output: g

3. getch():-getch() is a nonstandard function and is present in conio.h header file which is mostly used by MS-DOS compilers like Turbo C. It is not part of the C standard library or ISO C, nor is it defined by POSIX Like above functions, it reads also a single character from keyboard. But it does not use any buffer, so the entered character is immediately returned without waiting for the enter key.
Syntax:

```
int getch();
```

Example:

```
// Example for getch() in C
#include <stdio.h>
#include <conio.h>
int main()
{
  printf("%c", getch());
  return 0;
}
```

Run
Input: g (Without enter key)
Output: Program terminates immediately.
But when you use DOS shell in Turbo C, it shows a single g, i.e., 'g'

4. **getche()**-Like getch(), this is also a non-standard function present in conio.h. It reads a single character from the keyboard and displays immediately on output screen without waiting for enter key.
   Syntax:
   int getche(void);

Example:

```
#include <stdio.h>
#include <conio.h>
// Example for getche() in C
int main()
{
  printf("%c", getche());
  return 0;
}
```

Run
Input: g(without enter key as it is not buffered)
Output: Program terminates immediately.
But when you use DOS shell in Turbo C,
double g, i.e., 'gg'

Difference between getc() and fgetc()

The difference between getc and fgetc is that getc can be implemented as a macro whereas fgetc cannot be implemented as a macro. This means three things:

- The argument to getc should not be an expression with side effects be
- Since fgetc is guaranteed to be a function, we can take its address. This allow us to pass the address of fgetc as an argument to another function.
- Calls to fgetc probably take longer than calls to getc, as it usually takes more time to call a function.

Difference between putc() and fputc()

putc is equivalent to fputc but putc could be implemented as a macro and putc may evaluate its stream argument more than once.

Difference between gets() and scanf()

The main **difference** is that **gets** reads until EOF or \n , while **scanf**("%s") reads until any whitespace has been encountered.

Difference between gets() and fgets()

Both fgets() and gets() read a line terminated with a newline terminator, while gets() reads from stdin and fgets() reads from a specified file. Also, gets() strips the trailing newline from the result.

Printf(), sprint() and fprintf()

fprintf writes output to a file handle (FILE *).
sprintf writes output to a buffer that you allocate(**char***).
printf writes output to the standard output stream(stdout).

MEMORY MANAGEMENT IN C

1. STATIC MEMORY ALLOCATION

Memory is allocated for the declared variable by the compiler. The address can be obtained by using 'address of' operator and can be assigned to a pointer. The memory is allocated during compile time. Since most of the declared variables have static memory, this kind of assigning the address of a variable to a pointer is known as static memory allocation.

2. DYNAMIC MEMORY ALLOCATION

The exact size of array is unknown untill the compile time,i.e., time when a compier compiles code written in a programming language into a executable form. The size of array you have declared initially can be sometimes insufficient and sometimes more than required. Dynamic memory allocation allows a program to obtain more memory space, while running or to release space when no space is required.
Although, C language inherently does not has any technique to allocated memory dynamically, there are 4 library functions under "stdlib.h" for dynamic memory allocation.

1. malloc()

The name malloc stands for "memory allocation". The function malloc() reserves a block of memory of specified size and return a pointer of type void which can be casted into pointer of any form.
Syntax of malloc(): ptr=(cast-type*)malloc(byte-size)
Here, ptr is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

2. calloc()

The name calloc stands for "contiguous allocation". The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.
Syntax of calloc(): ptr=(cast-type*)calloc(n,element-size);
This statement will allocate contiguous space in memory for an array of n elements.

3.free()

Dynamically allocated memory with either calloc() or malloc() does not get return on its own. The programmer must use free() explicitly to release space.
syntax of free(): free(ptr);
This statement cause the space in memory pointer by ptr to be deallocated.

4.realloc()

If the previously allocated memory is insufficient or more than sufficient. Then, you can change memory size previously allocated using realloc().

Syntax of realloc(): ptr=realloc(ptr,newsize);

Here, ptr is reallocated with size of newsize.

FILE ORGANISATION

A file is a collection of data, usually stored on disk.The term "file organization" refers to the way in which data is stored in a file and, consequently, the method(s) by which it can be accessed.

Types of File Organisation:

A. SEQUENTIAL FILE ORGANISATION

It is the most basic way to organise the collection of records in a file. Records of the file are stored in sequence by the primary key field values. They are accessible only in the order stored, i.e., in the primary key order. It works well for the tasks which need to access nearly every record in a file.

On an average, to search a record in sequential file would require to access half of the records of the file.

The records in the file written consecutively when the file is created and must be accessed consecutively.

Advantages

- It is fast and efficient when dealing with processed periodically (batch system).
- Simplest technique
- Easy to search

Disadvantages

- This method is too slow to handle applications requiring immediate updating or responses.
- Search nth record, travel n-1 records to access any record,on an average half the records are accessed.

B. DIRECT FILE ORGANISATION/RANDOM FILE ORGANISATION

It is the most common form of random access to a file. It uses Hash Algorithm for the file organisation. Hash functions calculate the address of the page in which the record is to be stored based on one or more fields in the record. The Records in a Hash file appear randomly distributed across the available space. It requires some hashing algorithms and the technique. Hashing Algorithm converts primary key

values into a record address. The most popular form of hashing is division hashing with chained overflow.

Advantages

- Insertion or search on hash-key is fast.
- Fast technique

Disadvantages

- It is a complex file organisation method.
- Search is slow
- Sometime Hash functions may generate same address for different key values, which may cause "collisions" as if it is not possible to store different records on a same address. To overcome this problem :-Bucketing, Linked list etc.

Null Pointer

1. NULL Pointer is a pointer which is pointing to nothing.
2. NULL pointer points the base address of segment.
3. In case, if you don't have address to be assigned to pointer then you can simply use NULL
4. Pointer which is initialized with NULL value is considered as NULL pointer.

Example of NULL Pointer

```
#include <stdio.h>
main()
{
  int  *ptr = NULL;
  printf("The value of ptr is %u",ptr);
}
```

Output :

The value of ptr is 0

Command line arguments

It is a technique to run a "c" language program through command prompt.

When we run a c program through command prompt,we have to pass 2 parameters in main()

1 int argc—it is called argument counter.It stores the total no.of parameters passed to the main();

2 char *argv[]—It is called argument vector/array.It stores the parameters passed to the main() through command line.

The syntax of main() with command line argument:-

```
main(int argc , char *argv[])
```

Ex

```
Void main(int argc ,char *argv[])
{
int a,b,c;
a=(int) argv[1];
b=(int ) argv[2];
c=a+b;
printf("%d",c)
}
```

sizeof operator

1. sizeof operator is used to calcualte the size of data type or variables.
2. sizeof operator can be nested.
3. sizeof operator will return the size in integer format.
4. sizeof operator syntax looks more like a function but it is considered as an operator in c programming

Example of sizeof() operator

```
#include<stdio.h>
main() {
  printf("%d" sizeof(int));
  printf("%d", sizeof(float));
}
```

Type-cast operator

The type cast operator is very important in C. Cast operator uses in convert one data type to another data types. Type casting may be two types:

1. Implicit type cast
2. Explicit type cast

Implicit type cast

In C, implicit type cast are automatically handled by compiler i.e. when two or more data types are getting execution then the final data-type will be that data type as it is declared, i.e it is not depend on conversion of data type. It is clear understand by example as:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j;
 float f;
 double d;
 i=d*f+f*j;
}
```

what you think, *what will be data type of i*?
it is double!! No, right answer is int. You see in program that double has high priority or precedence of float and int, so result of data type will be comes in double but when result is assign in i, it will be convert in int because i is declared as int. It is *implicit type casting*.

Explicit type cast

An explicit type cast is a cast that we should specify invoke with either the cast. The compiler does not automatically invoke to resolve the data type conversion.

```
main()
{
int a=65;
char b=(char)a;
printf(%c,b);
}
```

Bitwise Operators in C Programming

A bitwise operator is an operator used to perform bitwise operations on bit patterns or binary numerals that involve the manipulation of individual bits.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Example #1: Bitwise AND

```
#include <stdio.h>
main()
{
   int a = 12, b = 25;
   printf("Output = %d", a&b);
}
```

Output

Output = 8

Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

Example #2: Bitwise OR

```
#include <stdio.h>
main()
{
   int a = 12, b = 25;
   printf("Output = %d", a|b);
}
```

Output

```
Output = 29
```

Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

Example #3: Bitwise XOR

```
#include <stdio.h>
main()
{
   int a = 12, b = 25;
   printf("Output = %d", a^b);
}
```

Output

```
Output = 21
```

Bitwise complement operator ~

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

Shift Operators

There are two shift operators in C programming:

- Right shift operator.
- Left shift operator.

Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

Left Shift Operator

Left shift operator shifts all bits towards left by certain number of specified bits. It is denoted by <<.