SIMATS

School of Engineering

# C Programming

Computer Science and Engineering

Saveetha Institute of Medical And Technical Sciences,Chennai.

# INDEX

# INTRODUCTION TO C

## Programming basics
## C Programming history

### PROGRAMMING BASICS

Irrespective of the programming language chosen for learning, the basic concepts of programming are similar across languages. Program languages are also made of several elements. We will take you through the basics of these elements And make you Comfortable to use them in various programming languages.

Those basic elements include-

- * Programming Environment
- * basic syntax
- * Data types
- * Variables
- * keywords
- * basic operators
- * Decision Making
- * Loops
- * Numbers
- * Characters
- * Arrays
- * strings
- * Functions
- * File I/O

### C PROGRAMMING HISTORY

C Programming language was developed in 1972 by demis Ritchie at bell laboratories of AT&T (Amercion Telephone & Telegraph), located in the USA. Dennis Ritchie is known as the founder of C language.
It was developed to Overcome the problems of previous languages such As B, BCPL, etc..

| Language | Year | Developed by |
|---|---|---|
| Algol | 1960 | International group |
| BCPL | 1967 | Martin Richard |
| B | 1970 | Ken Thompson |
| Traditionalc | 1972 | Dennis Ritchie |
| K&RC | 1978 | Kernighan & Dennis Ritchie |
| ANSIC | 1989 | ANSI Committee |
| ANSI/ISOC | 1990 | ISO Committee |

## Structure of C Program

Documentation

list Section

Definition Section

Global Declaration

Main Function

Sub Program Section

```
//Name of Program
    ↳ (Document Section)

#include<stdio.h>   }
# include <conio.h> } → Pre Processor
# include max 100

                    → Def section

void add(c)    }→ Global
int x = 100;   }  declaration Section

int main( ) → Main ( ) function
{
   int a=100; → Variable declaration
   Printf("Hello"); } → Body of
   return0;         }   Main func
{
void add(c)
{Printf("Hello add"); } → Fun declaration
```

## Pre Processor Directives

The C preprocessor is not a Part of Compiler, but it is seperate step in the Compilation process. In simple terms, a C preprocessor is just a text Substitution tool And it instructs the Compiler to do required pre-processing before the actual Compilation. All preprocessor begin with a hash symbol (#). It must be first nonblank character, And for readability. The following sections lists down all the important preprocessor directives:-

| S.no | Directive & description |
|---|---|
| 1. | #define (substitutes a Preprocessor macro) |
| 2. | #include (inserts a Particular header from another file |
| 3. | #undef (undefines a Preprocessor macro) |
| 4. | #ifdef (returns true if this macro is not defined |
| 5. | #if (Tests if a Compile time Condition is true |
| 6. | #else (The alternative for #if |
| 7. | #elif (else And #if in one statement) |
| 8. | #endif (Ends Preprocessor Conditional) |

## Constants, Variables, Data types

Constants:- * Values Cannot be changed
* Const keyword Can be used.

Eg: Const double PI = 3.14
(This value never changed)

- It Can be used # Preprocessor directive

Const double PI = 3.14; PI = 2.9; //Error

Variables:- * Container to hold data
* symbolic reply Memory allocation

Eg: int player score = 95,
(This Can be changed)

Rules * letters, digits and (_)
* 1st letter [letter (or) (_)]
* Var name any length

C is strongly type long
Var type Cannot be changed
Const once it is declared

Eg: int number = 5; // (Integer Variable)
number = 5.5; -// Error
double number; -// Error

Data types: * Types of data
* Used while defining var/fun
* Tells Computer to interpret value

Eg: Company stores data
Name: string  ID: Integer
salary: Float or double
Phone no: string

Data types
- basic → int, char, float, double
- Derived → array, Pointer, structure, union
- Enumeration → Enum
- void → void

## Operators and Expressions

* Symbols used to perform Specific tasks.

### ① Arithmetic Operators:

+ - Addition
- - Subtraction
* - Multiply
/ - Division
%. - Modulo Division

Eg: A+B, A-B, A*B, A/B, A%B

### ② Relational Operators:

< lessthan
<= lessthan or equal to
> greater than
>= greater than or equal to
== is equal to
!= Not equal to.

Eg: 1 < 5, 9 != 8, 2 > 1

### ③ logical operators:

&& Logical AND
|| logical OR
! Logical NOT

Eg: a < 8 || a > 60

### ④ Assignment operator.

• '=' is used for Assignment Operator.

### ⑤ Increment & Decrement:

++m Or m++ ⇒ Increment
--n Or n-- ⇒ Decrement

### ⑥ Conditional Operator:

* Uses "?" & ":" for condition Check.

Syntax
Emp1 ? Emp2 : Emp3

Emp1 is condition check if true Emp2 prints else Emp3 prints.

### ⑦ Bitwise Operator

& Bitwise AND
| Bitwise OR
^ Bitwise exclusive OR
<< Shift left
>> Shift Right

### ⑧ Special Operator

* Comma Operator
* Sizeof() operator.

### C operator precedence

Determines which operator is performed first in Exp.

Eg: 10 + 20 * 30

10 + 20 * 30
* ⇒ Higher Precedence
10 + 600
+ ⇒ lower Precedence
610

② 100 + 200 / 10 - 30 * 10
③ ① ④ ②

100 + 200 / 10 - 30 * 10
100 + 20 - 300
120 - 300
-180

| operator | Description | Associativity |
|---|---|---|
| ()<br>[]<br>.<br>-><br>++ -- | Parantheses: grouping or function call<br>Brackets (array Subscript)<br>Member selection via Object name<br>Member selection via Pointer<br>Postfix increment / Decrement | Left to Right |
| ++ --<br>+ -<br>! ~<br>(type)<br>*<br>&<br>Sizeof | Prefix increment / Decrement<br>Unary Plus / Minus<br>Logical negation / bitwise Complement<br>cast (convert value to temporary value of typ)<br>Dereference<br>Address (of operand)<br>Determine Size in bytes on this implementation | Right to left |
| * / % | Multiplication / Division / Modulus | left to Right |
| + - | Addition / Subtraction | left to Right |
| << >> | Bitwise Shift left, Bitwise Shift Right | left to Right |
| < <=<br>> >= | Relational less than / less than or equal to<br>Relational greater than / greater than or equal to | Left to Right |
| == != | Relational is equal to / is not equal to | left to Right |
| & | Bitwise AND | left to Right |
| \| | Bitwise OR [inclusive] | left to Right |
| ^ | Bitwise exclusive OR | Left to Right |
| && | Logical AND | left to Right |
| \|\| | logical OR | left to Right |
| ?: | Ternary Conditional | Right to left |
| =<br>+= , -=<br>*= /=<br>%= &=<br>^= \|=<br><<= >>= | Assignment<br>Addition / Subtraction assignment<br>Multiplication / division assignment<br>Modulus / Bitwise AND assignment<br>Bitwise exclusive / inclusive OR assignment<br>Bitwise Shift left / Right assignment | Right to left |
| , | Comma (seperate expression) | left to Right |

Find the output of the following Programs

```
1) #include<Stdio.h>
int main(){
    int a= -3,
    a+=!a-a-a;
   Printf("%d\n", a);
    return 0;
   }
```

```
2.) #include<Stdio.h>
 main(void){
  int a= 2, b=1, c,d;
    c = a<b;
    d=(a>b) && (c<b);
 Printf("c = %d, d=%d",
                c,d );
  }
```

```
3.) #include <stdio.h>
 main( ){
 int a=9, b=15, c=16,
     d=12, e,f;
 e= !(a<b || b<c);
 f= (a>b) ? a-b : b-a;
 Printf("e=%d, f=%d\n",
              e,f);
   }
```

```
4.) #include<stdio.h>
 main( ){
  int a= 5, b=5;
Printf("%d, %d\t",++a, b--);
Printf("%d, %d\t", a,b);
Printf("%d,%d\t",++a,
              ++b);
Printf("%d,%d\t",
              a,b);
 }
```

# C PROGRAMMING - DECISION MAKING / BRANCHING STATEMENTS

## ⇒ Branching statements

- if-else statement
  - ↳ Simple if
  - ↳ if-else
  - ↳ else if
  - ↳ Nested if else
- The switch statement

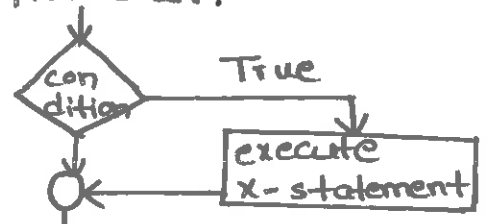
## → Simple if statement

* It is to control the flow of execution of statement and test logically whether condition
  - True (execute)
  - False (Skipped)

* Syntax:

```
if (condition)
{
  statement ;
}
```

* flow chart:



* sample program

```
#include <stdio.h>
voidmain()
{ int a,b;
 clrscr();
 Printf("enter A&B value");
 scanf( " %d",&a);
  if (a>b)
 { printf("A is big");
 }
  getch();
}
```

Sample output:-

```
20   10
A is big
```

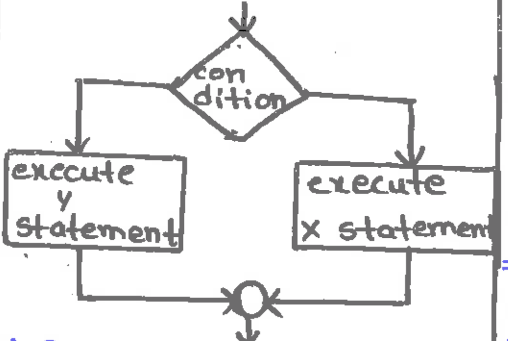
## → if - else statement

- condition True: statement execute
- condition False: this statement block execute

* Syntax:-

```
    if (condition)
    {statement 1;
    }else
    {statement 2;
    }
```

* flowchart:-



* Sample program

```
#include<stdio.h>
void main()
{int a,b;
 clrscr();
 Printf ("enter A&B value:");
 Scanf ("%d", &a);
 if (a>b)
 {printf("A is big");
 }
 else
 {
  printf ("Bis big");
 }
 getch();
}
```

Sample Output:-

```
10  20
B is big
```

## → else - if statement

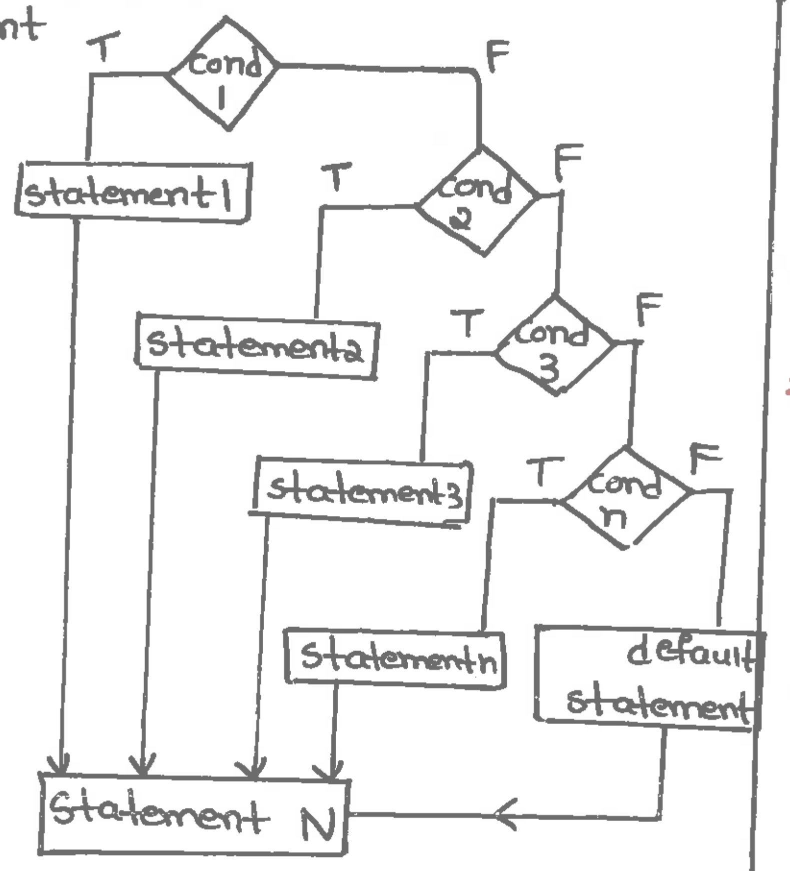
* Similar to if - else
* else if need for multipath decision
* Also called else if ladder.

* Syntax:-

```
if (condition 1)
{ statement 1;
 }
elseif (condition2)
{ statement 2;
 }
elseif (condition 3)
{ statement 3;
 }
 ...
else {stmt 4; }
```

Sample program:-

```
#include<stdio.h>
#include <conio.h>
int main(){
int  num1, num2;
printf("Enter 1st value :");
Scanf( "%d", &num1);
printf("Enter 2nd value:");
Scanf ("%d", &num2);
if (num1 = = num2)
{
printf("both 1st & 2nd value equal");
}
elseif (num1 >num2)
{
printf("1st value greater than 2nd");
}
else
{
 printf("1st value smaller than 2nd");
}
return 0;
}
```

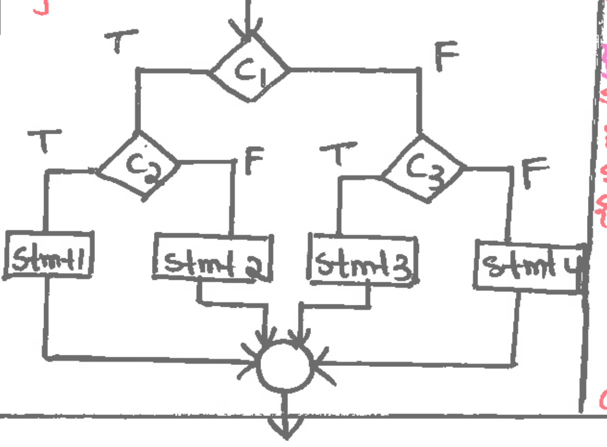


## → Nested if else Statement

* 'if' is placed inside another 'if' or else.

Syntax:

```
if (condition1){
        if(condition 2)
        stmt 1;
        else
        Stmt 2;
}
else {
     if (condition 3 )
     stmt 3;
     else
     stmt 4;
}
```



Sample program:-

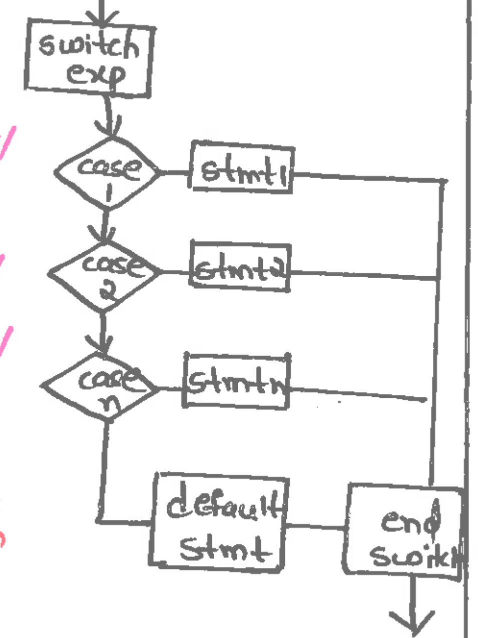
```
#include <stdio.h>
int main(){
int  num1, num2, num3;
printf ("Enter 3 numbers: ");
scanf("%d %d %d",&num1, &num2, &num3);
if(num1 > num2)
{ if (num1 >num3)
 {printf("Num1 is max");
 }
 else{
    printf("Num3 is max");
   }
}else
{ if (num2 > num3)
{ printf("num2 is max"); }
else{
    Printf ("num3 is max"); }
}
 return 0;
}
```

## ⇒ Switch case statement.

It gives ability to make decisions from fixed available choices

Syntax:

```
switch(exp)
{case1:
  /*stmts*/
 break;
Case2:
 /*stmts*/
casen:
  /*stmts*/
  break;
default:
 /*stmts*/
}
```



sample program

```
int num=2;
switch( num)
{
 case1: printf ("I am one");
        break;
 case2: printf("I am two");
        break;
Case3: Printf("I am Three");
        break;
default: Printf ("I am an integer");
}
```

## Looping in C:

→ To Execute the block of Code

→ Several times according to the Condn.

→ Executes Same Code multiple times.

**3 types:**

→ While Loop.

→ Do - While Loop.

→ For - Loop.

### ⇒ While - Loop:

Code Executes until Condn is False.

Syn:
```
while (Condn)
{
  //Code
}
```

Example:
```
#include <stdio.h>
#include <conio.h>
Void main()
{
 while (i<=20)
 {
  Printf ("%d",i);
 i++;
 }
 getch();
}
```

Output: ←
```
20
```

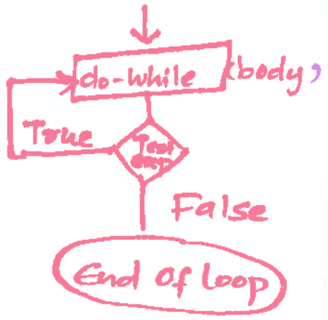
### Do - While Loop:

→ Executes until Condn gets false

→ Atleast once Code will Execute (Cond true or false)

While → Executed when the Condn is true.

Syn:
```
do
{
  //Code
} while (cond);
```



Example:
```
#include<stdio.h>
#include<conio.h>
Void main()
{
  int i=20;
  do
   {
    Print("%d",i);
     i++; } while (i<=20);
    getch();
   }
```

Output:
```
20
21
```

### For Loop:

→ Code Executes until Condn is false

3 Parameters:

→ Initialization

→ Condition

→ Increment / Decrement.

Syntax:
```
for (initialization; Cond; incr/decr)
  {
    //code
  }
```

Example:
```
#include <stdio.h>
#include <conio.h>
Void main()
{
  int i;
  for(i=20;i<25;i++)
  Printf("%d",i);
  } getch(); }
```

O/p:
```
20
21
22
23
24
25
```

## Loop Control Statement:

→ Change the Execution of loop (from normal Flow)

→ Break → Continue → Goto

### - Break -

→ End or Switch St

Example:
```
Void main()
{
 int i;
 for (i=0; i<10; i++)
 {
  print ("%d",i);
  if (i==5)
   break;
 }
Print("Came outside
     loop i=%d",i);
 }
```

Output:
```
0 1 2 3 4 5
Came outside of loop
      i=5
```

### - Continue -

→ Skips Some St as per Condn

Example:
```
Void main()
{
  int i=0;
  while (i!=10)
  { Printf("%d",i);
Continue;
i++;
}
 }
```

Output:
Infinite Loop

### - Goto -

→ Transfer Control to labelled St.

Example:
```
int main()
{
  int n,i=1;
  Printf("Enter no");
  Scanf("%d",&n);
 table:
   printf("%d,%d=%d \n",n,n*i);
i++;
 if (i<=10)
  goto table;
 }
```

# Problems Using Branching & Control Statement.

### 1) Check Whether the given number is odd or even.

```
# include < Std ie.h >
int main c)
{ int num;
 Scanf (" %d", & num);
 if (num % 2 = = 0 )
  print (" %d", is even ", num);
else
   print ("%d is odd ",num);
  return 0 ;
 }
```

/ ⇒ Quotient
% ⇒ Remainder

### 2) C program to find whether the Person is eligible for Vote or not And if that particular person is not eligible, the print how many years are left to be eligible.

```
#include < Studio.h >
 { int a;
 // input age
 Printf ("Enter the age of the
                        Person")
 Scanf (" %d", & a);
 // Check voting eligility
 if ( a>=18 )
 {
         printf ("Eligibal for voting);
 }
else  Printf ("Not eligibal for voting\n");
     Print (" Has to wait %d years", 18-a);
 }
  return 0 ;
```

### 3) Find the year of the given anniversary is leap year or not. If leap year then print the next anniversary, if not leap year then print the previous anniversary.

```
# include < stadio.h >
int main (){
  int year;
  print f (" Enter a year ");
 Scanf (" %d ", & year);
// leap year if perfectly divisible by 400
if  ( year % 400 = = 0 ){
  print ("%d is a leapyear.", year);
}
// not a leap year of divisible by 100
// but not divisible by 400
else  if  ( year % 100 = = 0 ){
    Print f (" %d is a leap year,", year);
}
// all other years are not leap years
else {
 printf  (" %d is not a leapyear,", year);
 }
  return 0 ;
  }
```

Sample input
1947
Sample ouput
1947 is not a leapyear.

### 4) Program using Switch case

// program to create a Simple calculature

```
# Include < Stdio . h >
Int  main ( ) {
  Char  operation;
 double n1, n2 ;
Printf("Enter an operator (+,-,*,/)");
Scanf ("%c", & operation );
print f ("Enter two operands ");
Scanf ("%.lf %.lf", & n1 & n2);
Switch (operation )
{
case'+'
printf(" %.1lf+%.1lf = %.1lf", n1, n2, n1+n2);
break;
Case '-'
 Printf("%.1lf-%.1lf= %.1lf", n1,n2, n1-n2);
 break;
Case '*'
Printf ("%.1lf * %.1lf = %.1lf", n1,n2
                          n1*n2);
break;
Case '/':
Print ("%.1lf/%.1lf = %.1lf", n1,n2,
                          n1 / n2);
break;
// operator doesn't match any
case constant +, -, *, / default:
 Print (" Error! operator is not
                      correct");
}
 return 0;
}
```

### C continue Statement.

The continue Statement in C language is used to being the program control to the beginning of the loop.

```
# include < Stdio.h >
int man (){
int i = 1; // initalizing a local
                        variable
// Starting a loop from 1 to 10
For(i =1; i <=10; i++){
if (i = = 5){// if value of i is
   equal to 5, it will continue the
                            loop.
continue
 }
print(" %d\n", i);
 } // end of for loop
return 0;
 }
```

The break Statement is used inside loops or Switch statement. The break statement breaks the loop one by one. ie. in the case of nested loop, it breaks the inner loop first and then proceeds to outer lopes. The break Statement in C can be used in the ~~ste~~ following two Scenarios.

```
for(i =0; i<10, i++)
{
print f(" %d ", i);
if ( i = = 5)
```

```
break
}
```

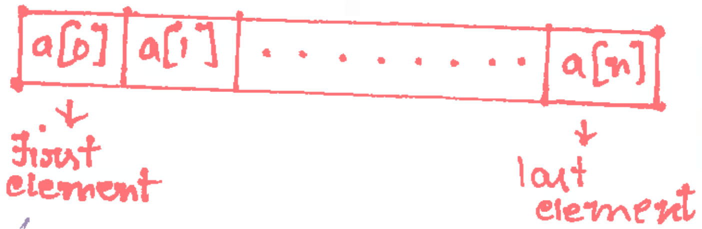
# Defining And Processing Array

## ARRAY & Types

=) Finite Orderd Collection of data

=) Stored in Contegious Memory location

=) Collition of var of Same data type



## Array declaration and Acces to arrays

```
datatype arrname [Size [Subscript]];
```

Datatype - int, float, double

arryname - Name of arry by user

Size/Subscript - No.of value of array

a [5] = a = arryname , S= Subscript Size of array

## Types of array

Array
One dimentional
two dimentional
Multi dimentional

### One dimenional arrey:

Ex: 
```
# include <Stdio.h>
int main ()
{
int i;
int arr [5] = {10,20,30,40,50};
for (i=0; i<5; i++)
{
Printf ["value of arr [%d] is %d\n", i,
  arr [i] ];
```

### Output:

```
Value of arr[0] is 10
Value of arr [1] is 20
value of arr [2] is 30
Value of arr [3] is 40
value of arr [4] is 50
```

## Two Dimentional array

```
Datatype arrname [No.of row, No.of Cols];
```

Eg:
```
# include <Stdio.h>
int main ()
{
 int  i,j;
int arr [2][3] = {10,20,30,40};
for (i=0; i<2; i++)
{
for (j=0; j<2; j++)
{
 Print ("value of arr [%d %d], %d/n", i,
  j, arr [i] [j];
  }
 }
}
```

### Output :

```
value of arr [0][0] is 10
Value of arr [0][1] is 20
Value of arr [0][0] is 30
Value of arr [1][1] is 40
```

## Multi Dimensionl Array :

```
datatype arrname [No.of rows] [Size]
                 [No.of Cols]
```

Eg: int test [2] [3] [4]

Size 1 = Row, Size = Col,

Size 3 = No. of elements

# Program Using Arrays

```
# include <Stdio.h>
int main()
{
 int i, j, k, test [2][3] [2];
Printf ("Enter 2 values :\n");
for (i=0; i<2; i++)
{
 for (j=0; i<2; +j)
 {
  for (k=0; k<2; ++k)
  {
   Scanf ("%d", & test [i] [j] [k]);
  }
 }
}
 Printf (" in Displaying Value : \n");
  for (i=0; j<2; ++i)
   {
    for (j=0; j<2; ++j)
     {
      for (k=0; k<2; ++k)
       {
Printf ("test [%d] [%d] [%d] = %d /n"]
 i, j, k, test [i] [j] [k]);
 } } }
return 0;
}
```

## Array Application :

① linear Search

② Binary Search

Single Dimentional array to input Sorting algorithem like :

- ✱ Inserting Sort
- ✱ Bubble Sort
- ✱ Selection Sort
- ✱ Quick Sort

## Example : C Program to reverse an arry elements

=) Reverse the element present in the array and display them

Following Steps to Solve:

=) for initialize i=0, whenie quationt of n/2, update :

- =) temp:- arr [i]
- =) arr [i] = arr [n-i-1]
- =) arr [n-i-1] = temp

=) for initialize i=0, when i<n, update (increses i by 1)

Eg:
```
# include < Stdio.h >
# include < Stdio .h >
# define n 6
• int main () {
• int arr [n] = {9,8,7,2,4,3};
• int temp;
• for (int i=0; i<n/2; i++){
• temp = arr [i];
• arr [i] = arr (n-i-1] ;
   arr [n-i-1] = temp
 }
   for (int i=0; i<n; i++){
     Printf ("%d," arr (i]);
 }
}
```

## Real time Appliations of Array

=) CPU Scheduling

=) Store image in Speific Size

=) Managing Contact on Mobiles

=) Viewing Screen as Multi demenhinal array of Pixel

=) Book title in LMS

# ARRAY APPLICATIONS

## LINEAR Search

=) linear Search also known as Sequential Search.

=) Method of finding elements within the list

✱ A Simple approch to implement liner Search

=) Begins with left most element of a[] and one by one compare val with elements

=) If val matches with an element, then Set flag to 1 and Store position.

=) if val does not match with any of the elements, display not found.

Example :

```
# include <Stdio.h>
int main() {
int a[]= {70,40,30,11,57,41,25,14,52};
int val,i,flag=0,P;
int n= Sizeof(a) / Size of(a[0]); // find
no. of elements.
Scanf ("%d , & val); // element of Search
for (i=0; i<n; i++)
 if (a[i] ==Val)
   {
   flag =1;
   P=i; // Position finding
   }
if (flag==1)
  Printf ("Given nomber %d is
 found at %d", Val, P+1);
  else
   Print ("Given nomber %d not found);
}
```

## Binary Search

=) Search technique that works efficienlly on Sorted arry

=) list must be Sorted

=) follows the divide and Conqur approch

=) list is divided into two halfes Item is Compared with the middle element of the list

Example :-

```
# include <Stdio.h>
int main()
{
int c, first, last, middle, n, Search,
array [100];
Printf ("Enter number of elements\n")
Scanf ("%d", &n);
Printf (" Enter value \n", n);
for (c=0; c<n; c++)
  Scanf("%d",& array [c]);
Printf ("Enter value to find /n");
 Scanf ("%d", & Search);
first = 0;
last = n-1;
middle = (first + last)/2;
While (first <= last) {
 if (array [middle ]== Search) {
Printf ("%d found at location%d\n",
Search, middle +1)
break;
}
else
last = middle -1;
middle = (first + last)/2;
}
 if (first > last)
Printf ("Not found ! %d isn't Prent /n",
 return 0;
}
```

## Bubble Sort

=) Bubble Sort is a basic algorithm for arranging a String of number or other elements in the Correct order

=) The method works by examination each set of adjacent element.

=) from left to right , Switching their positions

Example :-

```
# include <Stdio.h>
int main()
{
 int arry[100], n,c,d,Swap;
Printf ("Ender number of ele\n");
Scanf ("%d", &n);
Print f ("Enter %d integer\n", n);
for (c=0,c<n; c++)
  Scanf("%d", & array [c]);
for (c=0 ;c<n-1 ;c++)
  {
  for (d=0; d<n-c-1; d ++)
   {
   if (array [d]> arry [d+1])
     {
     Swap = array [d];
     array [d] = array (d+1)
     arrg [d+1] = Swap
     }
   }
}
 Printf ("Sorted list in asce Ord:\n")
 for (c=0; c<n; c++)
     Printf ("%d\n", array [c]);
return 0;
}
```

## Merge Sort

=) best eg of Divide & Counquer algorithm

=) Middle index of the array two halves m=(l+r)/2.

=) Call MergeSort to first half

=) Call Merge Sort to Second half

Example :

```
Void merge (int arr[], int l, int m, int r)
{
int i,j,k;
int n1 = m-l+1;
int n2 = r-m;
int L[n1], R[n2];
for (i=0; i<n1; i++)
L [i] = arr [l+i];
for (j=0; j<n2; j++)
R [j]= arr [m+1+j];
i=0;
j=0;
K=1;
while (i<n1 && j<n2 )
 {
 if (L[i]<= R[j])
 {
 arr [K] = L[i];
 i ++;
 }
else
{
ark[K]=R[j];
j++;
}
K++;
}
while (i<n1)
{
arr[K] = L[i];
i++;
K++;
}
while (j<n2)
{
arr[K]=R[j];
j++;
K++;
}
```

# FUNCTIONS

## => C Function.

* Large program divide into basic blocks

## => Use of C function

* Avoid rewriting same code
* improve understanding
* No limit in calling Function

## => Types of Functions

Library Function | user-defined Function

## => library function

* These functions are written by C Library.

* eg: printf(), Scanf(), sqrt(), math(), strcat(), rand(), etc.

*example:

```
#include<stdio.h>
# include <math.h>
main ( ) {
  float x,y;
  scanf ("%f"~~&x~~);
  y = sqrt(x);
  Printf("sq.root of %f
   is %f\n", x,y);
```

## => User defined function.

* These functions are defined by user at the time of writing program.

* example/Program using Functions

```
#include <stdio.h>
int multiply (int a,int b); //function declaration
int main() {
  int i, j, result;
  Printf ("enter 2 no.s");
  Scanf ("%d %d", &i, &j);
  result = multiply (i,j); //function call
  Printf("multiplication is
    %d", result);
  return 0;
int multiply (int a, int b)
{ return (a*b); //Function definition
}
```

## =>elements of user defined functions

Function definition | Function call | Function declaration

## =>Function definition:

Syntax:

```
return type functionname
             (argumentlist)
  {
     // function body
  }
```

funtion name, Function type, list of parameters } Function header

local variable dalaration, Function statement, Return statement } Function body

## => Function Call.

Syntax:

Function name (argument list);

```
main()
{
 ...
 func1(); ----> datatype func1()
 ...             {....
}                ....
                 }
Calling          Called
function         function
```

To Call a Function

call by value | call by reference

## => Call by value:

Fn. call

The value of actual Parameter copied to formal parameter

Fn declaration

example:

```
#include<stdio.h>
int sum (int a, int b){
  int c = a+b;
  return c;
}
int main(){
  int var1 = 10;
  int var2 = 20;
  int var3 = Sum(var1,var2);
  Printf("%d", var3);
  return0;
}
```

The operation performed on formal parameter don't reflect in actual parameter.

## => call by reference

The address of variable is Passed to function as parameter

example:

```
#include<stdio.h>
void increment (int *var){
  *var = *var+1;
}
int main(){
  int num=20
  increment (num);
  Printf("value of num=%d",
             num);
  return 0;
}
```

The operation performed on formal parameter affects the actual parameter.

## => Function declaration/Prototype

Syntax: returntype functionname (argument list);

* Also reffered as Function prototyping.

* 4 Parts
  - Return type
  - function name
  - parameter list
  - terminating

## => Category of Function:

1. With argument with return type.

```
int funct (int);
funct (a);
int funct(int a){
  stmt;
  return a;
}
```

2. with argument without return value

```
Void func (int);
func (a);
Void func (int a){
  statements;
}}
```

3. without argument without return type.

```
void func ();
 func ();
void func(){
  stmts,
}
```

4. without argument with return value.

```
int func(); //fn declaration
func(); //fn call
int func() // fn definition
{
  stmts;
  return a;
}
```

## Realtime Application of Functions in C:

→ Avoid rewritting Codes

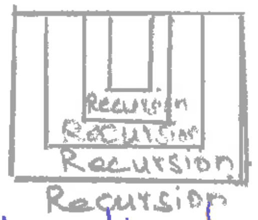
→ Predicting the natural disaster

→ Access to As many number of times from anywhere in the program.

# Recursion

- The process of Calling the Same function itself again & again until Some Condition is Satisfied.

## Recursive functions:



```
  int recursion (x)
Base {
case  if (x==0)   F
T       return;
T       recursion (x-1);
   }
```

Function being Called again by itself

## Example:

### Factorial of a number

```
#include<stdio.h>
int rec (int);
int main ( )
{
int a;
Printf ("Enter the number:");
Scanf ("%d", &a);
Printf ("The factorial of %d=%d", a, rec (a));
return 0;
}
int rec (int x)
{
int f;
if (x==1)
return 1;
else
f=x*rec (x-1); return f;
}
```

## Output:

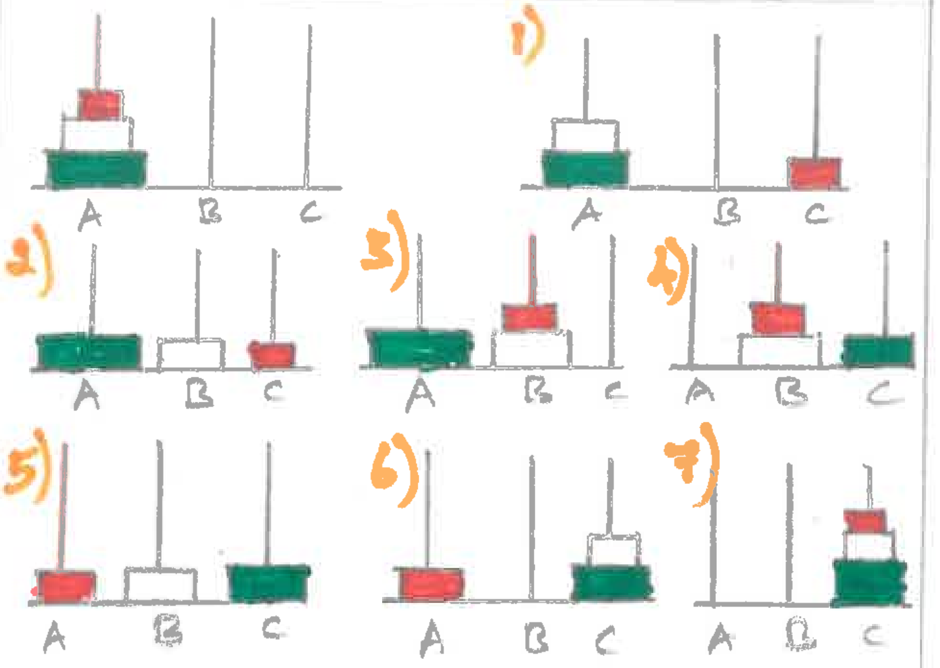
```
Enter the number 5
The factorial of 5=120
```

## Case Study:

### Tower of Hanoi

**Aim:-** To move the entire stack to another rod, obeying the following Simple rules.

- Only one disk can be moved at a time
- Each move consists of taking the Upper disk from one of the stacks and Placing it on top of another stack.
- No disk may be placed on top of a Smaller disk.



## Program:-

```
# include <stdio.h>
void towerofHanoi(int n, char from_rod,
                  char to_rod, char aux_rod)
{
 if (n==1)
 {
  Printf("\n Move disk 1 from rod %c
        to rod %c", from_rod, to_rod);
  return;
 }
 towerofHanoi (n-1, from_rod, aux_rod,
                        to_rod);
 Printf("\n move disk %d from rod %c
        to %c", n, from_rod, to_rod);
 towerofHanoi (n-1, aux_rod, to_rod, from_rod);
}
int main( )
{ int n=4;
 towerofHanoi(n,'A','C',);
 return 0;
}
```

# String Handling

## — STRING IN C —

→ Sequence of Characters.

Example:

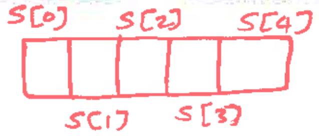
Char c[]="C String";

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| c | | s | t | r | i | n | g | \0 |

[MEMORY DIAGRAM]

Declaration:

Example: Char s[5];



Initialization:

Eg:

i) Char c[]="abcd";

ii) Char c[50]="abcd";

iii) Char c[]={'a','b','c','d','\0'};

iv) Char c[5]="{'a','b','c','d','e','\0'};

| | | | | |
|---|---|---|---|---|
| a | b | c | d | \0 |

Assigning Value to String:

Eg:

Char x[50];

x = "programming" //Error

Array type is not accessible.

Note:

Use strepy() instead of above.

Read String from User:

Scanf() used

Example:

```
#include<stdio.h>
int main()
{
 char name[20];
 printf("Enter name");
 Scanf("%s",name);
 Printf("Your name is %s",name);
 return 0;
}
```

Output: Entername: Dennis Ritchie.
You name is Dennis

Note:

name instead of &name

Scanf("%s",name);

↓

Char array, So & no need.

Read a line of text:

fgets() → To read line of string

puts() → To display.

Example:

```
int main()
{
 Char name[30];
 Printf("Enter name:");
fgets(name, sizeof(name), stdin);
Printf("name");
puts("name);
return 0; }
```

Output:
Entername:
Programming
name: Programming

String Function.

→ Standard Library (string.h)

1) Strlen() - Computes String length

2) Strcpy() - Copies a string to another.

3) Strcat() -
Joins 2 String

4) Strcmp() -
Compares two string

5) Strlwr() - Converts to lower-Case

6) Strupr() - Converts to Upper-Case

Syn:

1) Strlen (String_name)
2) Strcpy(destination, Source)
3) Strcat(firststring, 2nd string)
4) Strcmp (firststring, 2nd string)
5) Strlwr (String)
6) Strupr (String)
7) Strrev (String)

→ Perform Single program using String handling function in c.

Realtime Application Of String:

→ Spam detection [E-mail]
→ Plagiarism detection
→ Search Engine
→ Digital Forensic and information retrival System
→ Spell checker
→ Validation check in Database

# Pointers.

## Pointer:

→ a variable Whose Value is the address of another variables.

→ It must be declared before using it to store any variable address.

### Declaration :

The general form of a pointer variable declaration is

data_typevar_name;

data_type is the pointer's base type, it must a Valid c data type and var_name is the name of the pointer contains the address the result of an arithmetic operati. on performed on the pointer will also lie a pointer if the operand is of type integer.

### Operation on Pointers

Following arithmetic operation are possible on the pointer in c language:

- Increment:

```
int *p; // pointer to int
p = & number // stores the address of
                         number variables
p=p+1; // results+4 to value at p,
an address.
```

- Decrement :

```
int number = 50;
int *p; // pointer to int
P = & number;// Stores the address of no.
                                  variable
P=P-1;// decrement 4 from the address
                                at P.
```

- Addition

```
Scanf("%d%d",& first ,& second); //5,6
                          for first and
                                  Second
P = & First
q  = & Second
Sum =*p+*q; //-11 is stored at Sum.
```

- Subtraction

```
    Sub=*p-*q;//-1 is Stored at Sub
```

- Comparison

```
p2 = & b;
p1 = & a;
if (P1 > P2)
    print (" P1 is greater than p2")
else
    Print (" P2 is greater than p1")
```

## Accessing variable through pointer.

- Declare a normal variable, asign the Value.
- Declare a pointer Variable with the Same type as the normal variable.
- Initialize the pointer variable with the address of normal variable.
- Access the value of the variable by using asterisk (*)-It is known as dereference operator.

Initializing pointer variable.

```
int a=5; int *p, P=&a, *p_*(&a) gives Value
                               5 if printed
```

## program using Pointers with afunction

```
# include < Stdio.h >
void Swap (int* n1, int *n2);
int main()
{
 int num1=5, num2=10;
// address of num1 and num2 is passed
Swap (& num1, & num2);
printf(" num1 = %d\n", num1);
print(" num2= %d", num2);
return 0;
}
void swap (int* n1, int* n2)
{
 int temp;
 temp = *n1;
 *n1 = *n2;
 * n2 = temp;
}
```

## Programs using pointers with arrays.

```
# include < Stdio.h >
int main () {
int n;
int data[] = {1,2,3,4,5};
n = sizeof (data)/ Size of (data [0]);
Printf "( array accessed using
                     pointers:\n");
for ( int i = 0; i < n; ++i)

    Print(" %d\n", *(data + i );

   return 0;

  }
```

## pointers and Functions:

→ Create a pointer of any datatype such as int, char, flow, we can also create a pointer pointing to a function.

Syntax :

```
return type (* ptr - name )(type 1,
                            type 2,.....);
```

Example :

```
  int (*ip ) (int);
```

* ip is a pointer that points to a function which returns as int Value and accepts an integer Values as arguments.

```
float (*fp ) (float);
```

* fp is a pointer to a function that returns a float Value and accept a float value as arguments.

```
# Include < Stdio.h >
int add ( int , int );
int main ()
 {
  int a,b;
  int (* ip ) ( int , int );
  int results;
 Print f ("Enter the value of a & b:");
 Scanf ("%d*%d", &a, &b);
  ip = add;
result= ( * ip )( a,b);
Printf ("values after addition is
                        %d", result);
 return 0;
}  int add ( int a, int b)
 {
    int c=a+b;
    returnc;
 }.
```

## pointers using Array:

Sum of elements in 1-d array:

```
# include < Stdio.h >
int main ()
{
int a[10]; int i,sum=0; int*ptr;
printf ("Enter 10 element: ");
for ( i =0; i<10; i++)
Scanf (" %d", & a[i]);
ptr = a;
 for ( i = 0; i<10; i++)
{
 Sum = Sum + *ptr;
 ptr ++;
}
 Print (" %d", sum);
return 0;
}
```

Displaying the values in 2-d Array:

```
# include< Stdio.h >
int main()
{
int arr [3][4] = {
  {11,22,33,44},
  {55,66,77,88},
  {11,66,77,44}
 };
int i,j;
int (*p )[4]; p= arr;
for (i=0; i<3 ; i++)
{
printf ("Address of %d th array %u
                      \n", i, p+i);
for( j =0; j <4; j++)
{
Printf(" arr[%d][%d]= %d\n",
               i , j , *(*(p+i)+j ));
} print f ("/n\n");
 }  return 0;
  }
```

## Example programs using pointers with function

```
#include < Stdio.h >
Void add one ( int* ptr){
(* ptr)++; // adding 1 to * ptr
}
int main ()
{
int *p, i = 10;
P = & i;
addone( p);
Print f ("%d", *p)// 11
return 0;
}
```

Here the value stored at p, *p is 10 initially. We then passed the pointer p to the addone() function. The ptr pointer gets this address in the addone() function. Inside the function, we increased the value stored at ptr by 1 using *(ptr)++ Since ptr and p pointers both have the same address, *p inside main() is also 11

```
include < stdio.h >
void func1 (void(* ptr)());
void fun2();
int main()
{}
func1 (func2);
return 0;
}
void func1 (void(*Ptr)())
{
Printf("Function 1 is called");
(*ptr)();
{ }
void func2()
{
printf ("\nFunction 2 is called")
}
```

In the above code, we have created two functions; ie, func1() and func2(). The fun1() function contains the function pointer as an argument. In the main() method, the func1() method is called in which we pass the address of fun2. When fun1() function is called, 'ptr' contains the address of 'func2'.

## Example programs using pointers with arrays.

```
# include <stdio.h>
int main ()
{
// array declaration and initialization
int a[5] = {5,6,7,8},i;
// Valid in case of arrays but not valid in case of single integer value.
int *ptr=a;
// All representations prints the base address of the array.
print ("ptr: %u, &a[0]: %u, a: %u, &a: %u\n", ptr&a[0], a & a);
for (i=0;i<5;i++)
{
 // Printing address values.
 printf("[index %d] Address: %u\n", i, (ptr+i));
}
printf ("\n");
for (i = 0; i<5; i++)
// Gives address of next byte after array last element.
printf("&a: %u, &a+1: %u\n", &a, &a+1);
// Gives the address of the next element
printf ("a: %u, a+1: %u\n", a, a+1);
// Gives value at index 1
printf("* (a+1): %d\n", *(a+1));
// Gives (value at index 0)+1
printf("*a+1: %d\n", *a+1);
// Gives (value at index 0)/2 we can't perform *(p/2) or (p*2)
printf ("(* ptr/2): %d\n", (*ptr/2));
return 0;
}
```

```
your code was executed successfully
ptr: 1709381934, &a[0]: 1709381938, a: 1709381984
[index 0] address: 1709381984
[index 1] address: 1709381988
[index 2] address: 1709381992
[index 3] address: 1709381996
[index 0] value: 5555
[index 1] value: 6666
[index 2] value: 7777
[index 3] value: 8888
&a: 1709381984 &a+1: 1709382004
a: 1709381984, a+1: 1709381988
*(a+1): 6
* a+1: 6
(* Ptr/2): 2
```

## Command Line Argument in C

→ Parameter supplied to program when it is invoked

→ Mostly used when you need to Control your program from outside.

→ Arguments passed to main ()

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

→ Argc stands for Argument Count.

→ Argv stands for Argument Values

→ These are arguments passed to the main function when its starts executing

→ Argv [0] holds the name of the program

→ Argv [1] Points to the first Command line argument

→ Argv [n] gives last argument

→ If no augments is supplied argc will be 1

Program:- Write a c program to find the area of circle when the diameter is given. The input diameter is an integer.

```
# include < stdio.h >
# include < stdlib.h>
int main (int argc, char* argv[])
{ int diameter;
 float radius, area;
 if (argc >= 2){
diameter = atoi (argv[1]);
radius = diameter/2;
area = (3.14)* (((float) radius)*
                    (float) radius);
Print f ("%.2f", area);
}
 return 0;
}.
```

## Dynamic Memory Allocation

Allocation memory at runtime

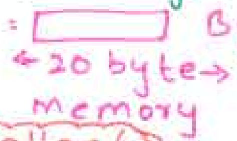
i) malloc() ii) calloc () iii) free () iv) realloc ()

1] Malloc(): → For dynamically allocating single large block of memory with size

Syntax:

Ptr = (Cast-type*) malloc (byte-Size);

Ptr :



Byle = 5

int = 4 x 5 = 20 bytes

2] Calloc(): → Dynamically allocated specified no. of blocks.

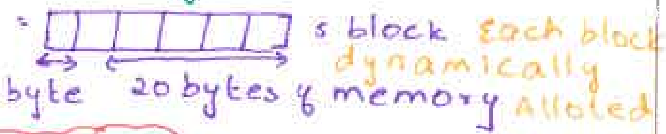
→ Initialized each block with default values.

Syntax:

Ptr = (cast-type*) Calloc (n.element-size)

n = number of elements

Ptr :



3] free(): → Dynamically alloted Memory

→ Helps to reduce wastage of Memory

Syntax: free (Ptr);

int* ptr = (int*) Calloc(5, Size of(int));

Ptr =

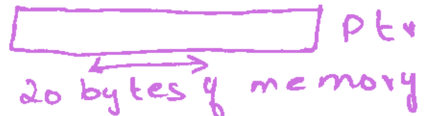
(free ptr)
20 bytes of memory

4] realloc(): → change the Memory allocation

→ change existing to new block

Syntax: Ptr = realloc (Ptr, new size);

int * ptr = (int*) malloc( 5* size of (n);



Ptr = realloc (Ptr, 10* size of (int);

Ptr
40 bytes of memory

# STRUCTURE

## ⇒ Structure definition

* different datatype represented by a Single name.

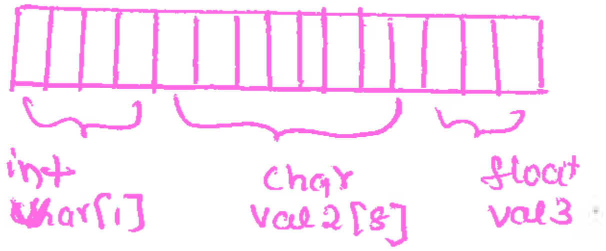
* User defined datatype.

* individual element called members

### Syntax:

```
Struct Structure_name
{
  data-type member 1;
  data-type member 2;
  ....................
  data type member n;
};
```

### memory allocation in structure



### example :-

```
Key    Struct mystruct      Structure tag (or) Structure name
word {
        int var1 ;
        Char var2[8];      } member
        float var 3 ;
     }  Struct_var;        Structure Declaration
```

⇒ To declare variables of a Structure :

Syntax :

```
Struct Struct_name var_name;
```

## ⇒ Access data member of a Structure

```
Varname. member1-name;
Var-name. member2-name;
.....
```

### Example:

```
Struct employee
{ char name[20];
  int age;
Char department[15];
Char gender;
};
Struct employee e1,e2;
```

from above, variables within the definition of Structure.

### Example :-

```
Struct Employee
{
  Char name [20];
  int age;
  Char department [15];
  Chor gender ;
} e1, e2 ;
```

## ⇒ initilization of structure members.

* Structure is defined and Memory is allocated when Structure variables declared.

### Example :-

```
Struct rectangle
{ // Struct definition
  int length = 10;     } compilation
  int breadth = 6;     } error
}
```

data members are initilized inside Structure.

### Example :-

```
Struct rectangle
{ //Struct definition
  int length ;
  int breadth;
};
int main ()
{ Struct rectangle my_rect
  my-rect. length = 10,
  my-rect. bredth = 6 ;
}
```

From above, initialized Structure using Structure variable.

## ⇒ Example of Structure C/ Program using Structure

```
#include<stdio.h>
Struct Studentdata
{  Char* Stu_name;
   int Stu_id ;
   int Stu-age;
};
Int main()
{
  Struct Studentdata Student;
  Student . Stu-name = "Jee";
  Student . Stu-id = 1234;
  Student . Stu-age = 20 ;
Printf ("Student name is : %.s",
   Student . Stu-name);
Printf ("Student id is : %d",
       Student . Stu-id);
Printf ("Student age is : %d",
       Student . Stu-age);
  return0;
}
```

### Sample output:

```
Student name : Jee
Student id   : 1234
Student age  : 20.
```

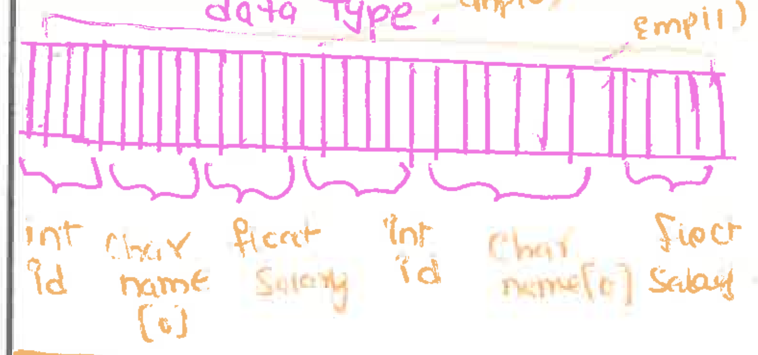
# ARRAY OF STRUCTURE, STRUCTURE POINTER, STRUCTURE FUNCTION

## Array of Structure

*collection of structores

*each variable - different entities

*multiple entities of different data type.



Struct tagname arrayname[i]

declaration struct employee emp[2]

```
Struct Employee
{ int id;                    // Structore
  Char name[5]                  declaration//
  float salary
}; struct Employee emp[2] //array of
                              structore//
int main()
{ size of emp
  size of emp[2]               // 4+5+4=13 bytes
  Emp[0].id =1;                // 26 bytes
strcpy[emp[0].name "Priya");  // direct
                                 declaration/
  Printf(" \n enter salary:");     assignment
  Scanf("%f & emp[0].salary);   // from
for[i=0; i<3; i++)                 users
    Printf("%d %s %f") emp[i].id,       Display;
        emp[i] name, emp[i].salary);    1, Priya,
 }                                      24500.
 return0;
}
```

## Nested Structure

*Structure within a structure

*address of employee - structure contain no; city, pincode.

*address structure within employee structured. - Nested.

ways
- → Separated structure
- → Embedded structure.

### Seperate structure

- Two structure are declared independently.

(Nested) Embedded structure

declare the structure inside the structure.

### Seperate Structure

```
Struct date
{ int dd;                 // Structure
  int mm;                    declaration //
  int yyyy;
} structure Employee
{  int id;                // nested
  Char name[20];             structure
  Struct Data do};           (separate)//
} emp;
```

### Embedded Structure

```
   Struct Employee;
{ int id; char name[20]
Struct date                       // Embedded
 { int dd; int mm; int yyyy          stru//
  } do;
} emp;
```

## Structure Pointer

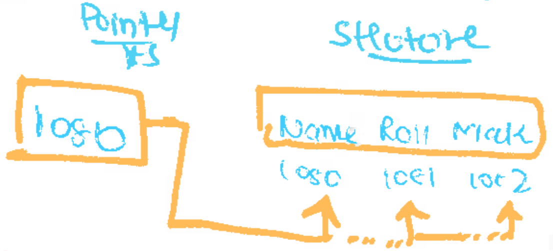
*Pointer to the address of structure memory block.

*used in linked links, trees, graphs.

### Declaration of Structure Pointer

Struct tagname *Pointer Varible-name

eg;- Struct Student *st



### Initilization of Structure Pointer

Pointer name = & struct.variable

or

Struct tagname *ptr = & v-name.

Accessing

- * asstelic production operation and . dot operator
- → array operation / membership operator

Structoperator → member name

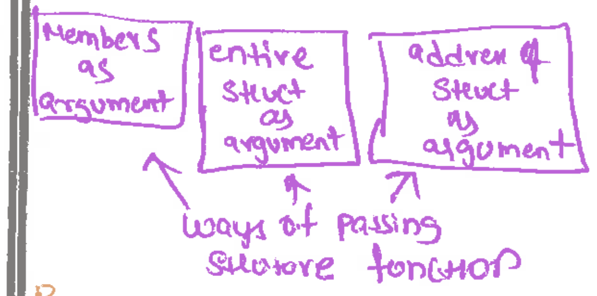
```
eg:-  St → name
      St → Roll
        (or)
      *st.name
      *st.Roll
```

```
#include<stdio.h>
Struct Student {
char name[30]; int Roll; float
   marks;
 };
Main() {
Struct student s;
Struct Student *st
Scanf ("%s, %d, %f", &st-name,
   & st.Roll, &st.marks);
St = &s
Printf ("Name = %s \n" st→name);
Printf ("Roll = %d \n" st→Roll);
Printf ("marks = %f \n" st→marks);
getch()
}.
```

## Structure function

*Structure passed as function arguments

*code efficient, less memory less execution time.


Members as argument
entire struct as argument
address of struct as argument
ways of passing structure function

Passing Members as argument

function name(struct name var name.member)

Passing structure as argument

function name(struct name)

Passing address as argument

functionname(& struct-name)

A **Union** is a special data type that allows to store different data types in the same memory location.

A union can be defined with many members, but only one member can contain a value at any given time.

The format of union statement is

```
union [union tag] {
   member definition;
   member definition;
...
   member definition;
} [one or more union variables];
```

```
union Data {
  int i;
  float f;
  char str[20];
}
```

Here, a variable of Data type can store an integer, a floating point number, or a string of character. The memory occupied by a union will be a large enough to hold the largest number of the union. For example, in the above example, Data types will occupy 20 bytes of Memory space because this is the maximum space which can be occupied by a character string. To access any member of a union, we use the member access operator(.). The member access operator is coded as a period between the union variable name and the union member.

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

union Data {
 int i;
 float;
 char str[20];
};

  int main() {

   union Data data;
data.i = 10;
data.f = 220.5;
strcpy(data.str, "C programming");

Printf("data.i: %d/n", data.i);
Printf("data.f: %f/n", data.f);
Printf("data.str: %s/n", data.str);
return 0;
```

When the above code is compiled and executed, it produces the following result -

```
data.i: 1917853763
data.f: 4122360580327794860
  -452759994368.000000
data.str: C programming
```

See that the values of i and f members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of str member is getting printed very well.

**Storage classes**: storage classes are used to describe the features of a variable/function. These features basically include the scope, visibility & life-time which help us to trace the existence of a particular variable during the runtime of a program

| storage specifier | storage | initial value | scope | life |
|---|---|---|---|---|
| Auto | stack | Garbage | within block | End of block |
| extern | Data segment | Zero | global Multiple files | Till end of Program |
| static | Data segment | Zero | within block | Till end of Program |
| register | CPU register | garbage | within block | End of block |

## Automatic storage class

It is also known as the auto storage class, and it acts as the default storage class for all the variables that are local in nature

For example,

```
{
 int mount;
 auto int month;
}
```

look at the example that we used above. It defines two of the variables in the very same storage class. One can use 'auto' only within the functions - or the local variables

## External storage class

It is also known as external storage class & we use it for giving a reference of any global variable which is visible to all the files present in a program. However, note that it points to the name of the variable at any storage location that we have already defined.

## Static storage class

This type of storage class gives an instruction to a compiler to keep the given local variable around during the program's lifetime - instead of creating it and then destroying it every time it comes into a scope and goes out of it. It allows the variable to maintain the values that are available b/w various function calls.

## Register storage class

We use the register storage class for defining the local variable that must be stored in any register and not in a RAM. It means that the maximum size of the this variable is equal to that of the register size.

For example,

```
{
register int miles;
}
```

We must only use the register in the case of those variable which requires quick access, such as counters. It rather means that this variable MIGHT or might not be stored in a register. It totally depends on the hardware & also the restrictions of implementation.

# FILE HANDLING IN C

## FILE STRUCTURES & POINTERS:-

⇒ data structure of a FILE is defined as FILE in the library of standard I/o function definitions. Therefore all filkes should be declared as type FILE before they are used.

⇒ FILE is a defined data type.

⇒ when we open a file we must specify what we want to do with the file.

⇒ For example, we may write data to the file or read the already existing data.

```
FILE *p;
fp = fopen("filename", "mode");
```

## operations done in file handling:-

* The primary operations that can perform on file in c

→ opening a file that already exists.

→ create a new file

→ Reading content/data from existing file.

→ writing more data into file

→ deleting the data in file

→ opening a file - To create/edit

→ fopen() function is defined in header file - stdio.h

↳ syntax:

```
ptr = fopen ("openfile", "openmode");
```

↳ example:

```
fopen ("E:\\myprogram\\one.txt", "w");

fopen ("E:\\myprogram\\one.txt", "rb");
```

→ opening mode of c

| mode in program | meaning of mode |
|---|---|
| r | open file for reading |
| rb | open file for binary reading |
| w | open file for writting |
| wb | open file for binary writing |
| a | open file for appending |
| ab | open file for appending binary |
| r+ | open file for reading & writing |
| rb+ | open file for read & write in binary |
| w+ | open file for writing & reading |
| wb+ | open file for write, read in binary |
| a+ | open file for appending, writing |
| ab+ | appending & writing in binary. |

→ close a file

* once we write /read a file, need to close it

⇒ To close a function, fclose() function.

```
fclose (fptr);
```

fptr refers to the file pointer associated with file needs to close in program.

→ Read and write data to the text file

* writing data

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int val;
 FILE *fptr;
 fptr = fopen("c:\\program.txt", "w");
 if (fptr == NULL)
 {
  printf("File type invalid");
  exit(1);
 }
 printf ("Enter value:");
 scanf ("%d", &val);
 fprintf (fptr, "%d", val);
 fclose (fptr);
 return 0;
}
```

* Reading information from Text file in program

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
 int val;
 FILE *fptr;
 if ((fptr=fopen ("c:\\prgm.txt",
      "r")) = NULL) {
 printf("variable error detected
      cannot open file");
 exit(1); }
 fscanf (fptr, "%d", &val);
 printf ("The val of the
    integer is %d", val);
 fclose (fptr);
 return 0;
}
```

## CASE STUDY:

case1: If his basic salary is less than Rs.1500, then HRA = 10% of basic salary and DA = 90% of basic salary. If his salary is either equal to or above Rs.1500, then HRA = Rs.500 & DA = 98% of his basic salary. If the employees salary is input through keyboard write a program to find his gross salary.

```
/*Calculation of gross salary */
main()
{
 float bs, gs, da, hra;
 printf("enter basic salary");
 scanf("%f", &bs);
 if( bs < 1500)
 {
 hra = bs * 10/100;
 da = bs * 90/100;
 }
 else
 { hra = 500;
 da = bs * 98/100;
 }
 gs = bs + hra + da;
 printf("gross salary= Rs. %f", gs);
}
```

case2:- A travels company insures its driver is not insured. If the marital status, gender and age of driver are inputs, write program to determine whether the driver it to insured or not.

* Driver is married
* Driver is unmarried male above 30 years age
* driver is unmarried female above 25 years age

```
main()
{
 char gender, ms;
 int age;
 printf( "Enter age, gender, marital status");
 scanf("%d %c %c", &age, &gender, &ms);
 if ((ms=='m') || (ms=='U' && gender=='m' && age>30) ||
 (ms=='U' && gender=='F' && age>25))
  printf ("Driver is insured");
  else
  printf()("Driver is not insured");
}
```

# Advanced Topics

## Cognitive agents

* Cognitive agents are born out of one of the major elements of aI : Cognitive Computing.

→ it's the simulation of human thought Processes in a computerized model. and it includes Self learning systems that leverage data mining, pattern recognition, and natural language Processing (NLP) to mimic patterns of human brain

* while cognitive agents are a great way to Save money, but they have other benifits as well.

* This technology can also improve data Security, customer and employee Experience, and Visibility over business Processes.

## Mobile bots :-

* computer bots and Internet bots are essentially digital tools and like any tool, can be used for both good and bad.

* Mobile bots can run off of real mobile devices, but often off servers, attempting to Stimulate Specific tasks, such as clicks, installs and in app engagement.

* Another form of bots can be Identified as malware located on a user's device.

in C graphics, the graphics.h functions are used to draw different shapes like Circles, rectangles etc---

* display text in a different format (different font & colour).

* by using functions in the header graphics.h, programs, animations and games.

## Functions used :

line ($x_1, y_1, x_2, y_2$) : it is a function Provided by graphic.h header file to draw a line.

circle ($x, y, r$) : it is a function provided by graphic.h header file to draw circle.

rectangle ($x_1, y_1, x_2, y_2$): Provided by graphic.h header file to draw rectangle

delay (n) : it is used to hold the Program for Specific time period.

Cleardevice () : it is used to clear the screen in graphic mode.

Close graph() : it is used the graph.

```
//c implementation for circle
include <graphics.h>
int main()
{
    Int gd = DETECT, gm;
    intgraph (&gd, &gm, "");
    circle (250,200,50);
    getch()
    closegraph();
    return 0;
}
```

## Design a Car and its moving

```
#include <graphics.h>
# include <dos.h>
# include <conio.h>
main ()
{ int i,j=0, gd = DETECT, gm;
initgraph (&gd, &gm, "C:\\TURBOC3\\BGI");
Settextstyle (DEFAULT_FONT, HORIZ_DIR, 2);
getch();
Setviewport( 0,0, 639, 440, 1);
for(i=0; i<=420; i=i+10; j++)
{
  rectangle(50+i, 275, 150+i, 400);
  rectangle( 150+i, 350, 200+i, 400);
 circle( 75+i, 410, 10);
circle( 175+i, 410, 10);
Setcolour(j);
delay (100);
if (i==420)
 break;
 clearviewport();
} getch();
Cleardevice();
Settextstyle(SANS_SERIF_FONT,
        HORIZ_DIR, 2);
Outtextxy(100,200,"mini project");
delay (5000);
Closegraph();
 return 0;
}.
```

## Low Level Programming features

⇒ C also supports low level programming features which enable the programming to carry out bit-wise operations.

⇒ These features are normally Provided in assembly language or machine language.

## Register Variables :-

* earlier, we have seen that C supports four different Storage classes, viz. Static, auto, extern and register.

* As such general purpose register c are special storage areas within cpu.

* in C, the content of register Variables reside inside registers.

* A Program that uses register variables execute faster since values are stored inside registers within cpu.

## Bitwise operator :-

* C allows the manipulation of individual bits within a word of computer memory.

## Masking :-

* The masking operation transforms the bit patterns of an operand with the help of a Specially Selected bit pattern called mask.

# SIMATS
## SCHOOL OF ENGINEERING

Engineer to Excel

Approved by AICTE | IET-UK Accreditation

Saveetha Nagar, Thandalam, Chennai - 602 105, TamilNadu, India