06/12/2021

PROGRAMMING FOR PROBLEM SOLVING

Hardware → System Software → Application Software → User

* COMPUTER → Software + Hardware
  - Software → System, Application

* SOFTWARE → collection of programs that allow hardware to do its job.

System Software
→ interface b/w hardware & users

keep the system operating in an efficient manner

- Operating Systems — services such as interface, file & database access
- System Support — system utilities & other operating services
- System Development — language translators, debugging tools.

Application Software
→ helping users

- General Purpose — word, DBMS.
- Application Specific — used only for the task for which they are designed.

*

Central Computers
- → TERMINALS
- → TIME
- → CENTRAL STORAGE
- → SHARING.
- → SHARED PRINTERS

* Types of computing environments:

1) Personal 2) Time Sharing 3) ~~Owner~~ Cluster
4) Client Server 5) Distributed 6) Cloud

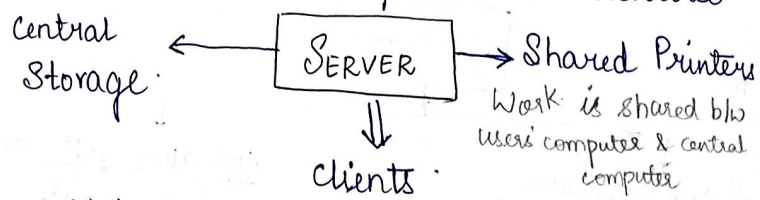
08/12/2021

* Computing Environment:

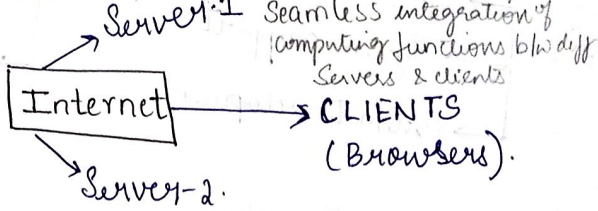
→ Personal: Single system & Single user.

→ Time Sharing: Multiple users to share simultaneously

→ Client Server: Client requests for resource & Server provides the resource.



→ Distributed Computing: contains multiple node.



→ Cloud Computing: done by pooling all the computer resources and then managing them using a software.

→ Cluster Computing: Similar to parallel computing environment as they both have multiple CPUs.

Text Editor → Source → Compiler → Object → Linker → Executable code

* COMPUTER LANGUAGES:

Computer Languages have evolved from machine language to natural language.

1) Machine Languages / Low level languages (1940s)
2) Symbolic Languages / Assembly Language (1950s)
3) High-Level Languages (1980s).

* Assembler: Converts assembly language to machine language.

* HIGH-LEVEL LANGUAGES: C; C++; Python; Java; Cobalt;

C-language:

* C-COMPILER: Converts C-language to machine language.

* Language Translators: • Assembler;
• Compiler (C)
• Interpreter (Python)

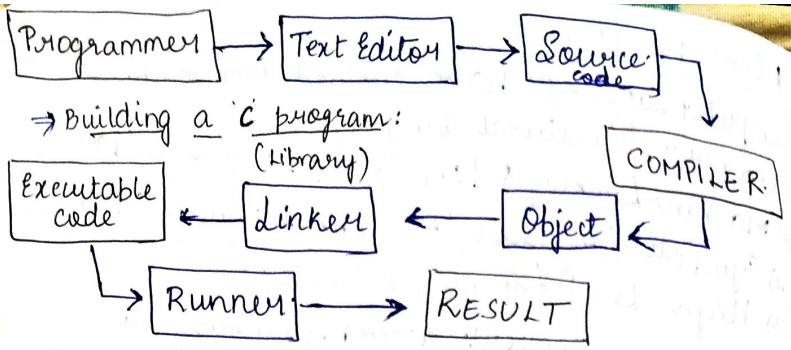
JAVA: uses compiler as well as interpreter.

• compiler converts whole code to machine language & executes

• Interpreter converts the code to machine language line by line and executes only when it is error-free.

* CREATING & RUNNING PROGRAMS:

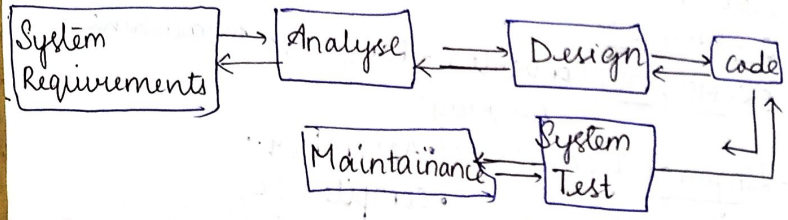
→ Writing & Editing
→ Compiling ; → Linking
→ Executing



⇒ Building a C program:

* SYSTEM DEVELOPMENT:

* System Development Life Cycle.
* Program Development.



* NUMBER SYSTEM:

(Base = 10)

1) Decimal Number System (0,1,2,3,4,5,6,7,8,9)
2) Binary Number System (0,1) [base = 2]
3) Octal Number System (0,1,2,3,4,5,6,7) [base = 8]
4) Hexadecimal Number System [0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F] [base = 16]
   10 11 12 13 14 15

* Decimal to Binary:

Ex1) $(25)_{10} = (11001)_2$

2 | 25
2 | 12 — 1
2 | 6 — 0
2 | 3 — 0
1 — 1

2) $(30)_{10} = (11110)_2$

2 | 30
2 | 15 — 0
2 | 7 — 1
2 | 3 — 1
1 — 1

3) $(25.75)_{10} = (11001.11)_2$

2 | 25
2 | 12 — 1
2 | 6 — 0
2 | 3 — 0
1 — 1

0.75 × 2 = 1.5 — 1
0.5 × 2 = 1 — 1

4) $(30.125)_{10} = (11110.001)_2$

0.125 × 2 = 0.25 — 0
0.25 × 2 = 0.5 — 0
0.5 × 2 = 1 — 1

*) $(25)_{10} =$

8 | 25
8 | 3 — 1

* Decimal to Octal:

1) $(25)_{10} = (31)_8$

8 | 25
3 — 1

2) $(30.75)_{10} = (36.6)_8$

8 | 30
3 — 6

0.75 × 8 = 6.00

Decimal to Hexadecimal

1) $(30)_{10} = (1E)_{16}$

16 | 30
1 — 14

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

* Binary to Decimal

1) 1011.10
$2^3\ 2^2\ 2^1\ 2^0\ .\ 2^{-1}\ 2^{-2}$
$8+0+2+1+0.5+0.25$
$(11.5)_{10}$

2 | 15
2 | 7 — 1
2 | 3 — 1
1 — 1

2) $(11110) = (30)_8$
$2^4\ 2^3\ 2^2\ 2^1\ 2^0$
$16+8+4+2+0$

* Octal to decimal

$(36)_8$
$8^1\ 8^0$
$24+6 = (30)_{10}$

$(25)_8 = (21)_{10}$
$8^1\ 8^0$
$16+5$

* Hexadecimal to decimal

$(1E)_{16} = (30)_{10}$
$16^1\ 16^0$
$16+14$

* Binary to Octal (grouping of 3)

~~$(101110)_2$~~ $= (011\ 010\ 110)_2 = (326)_8$

* Binary to Hexadecimal (grouping of 4)

$(1101\ 0110)_2 = (D6)_{16}$

* Octal to Binary

$(250)_8 = (010\ 101\ 000)_2$
$(53)_8 = (101\ 011)_2$

* Hexadecimal to Binary

$(1E)_{16} = (0001\ 1110)_2$

* Octal to Hexadecimal.

Octal to Binary

Binary to Hexadecimal.

15/12/2021

* ALGORITHMS & FLOW CHARTS:→ 2 ways to represent the solution for given problem.

→ Algorithm: Step by step procedure.

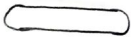
Eg: Adding 2 nos say - a, b

1) Start / Begin
2) Read no. 'a' / Read the value of a
3) Read no. 'b' / Read the value of b
4) Calculate Sum = a+b
5) Write sum / Print Sum / Display Sum.
6) Stop. / End.

→ Flow charts: Graphical / pictorial representation of the algorithm.

Eg: ~~Adding 2 nos say a, b.~~

* SYMBOLS:

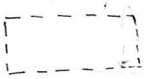
1)  → Terminal

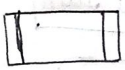
2)  → Read / write

3)  → Calculation / assignment / process.

4)  → Flow line

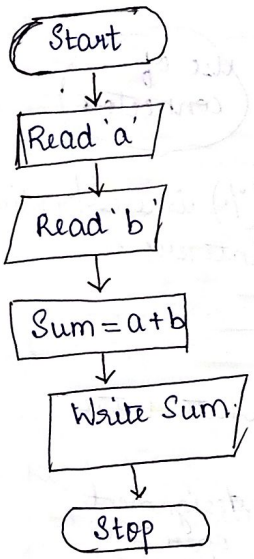
5)  → Decision / to check certain condition.

6)  → Compound Statements

7)  → Module / Procedure Call.

8)  → Connector

Eg: Adding 2 numbers say - a; b



The data on which operator works are called operands.

Arithmetic Operators:

+, −, *, /, %

* Calculate the S.I. by using algorithm & flowchart.

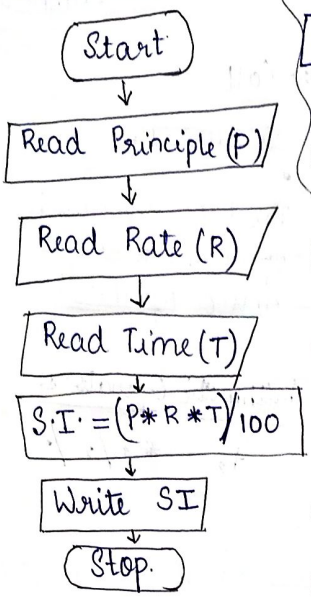
→ Algorithm:

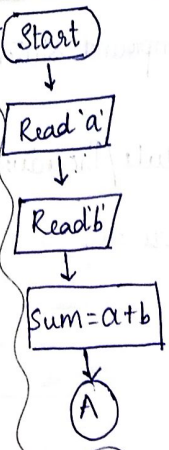
1) Start / Begin
2) Read the principle 'P'
3) Read time 'T'
4) Read rate 'R'
5) Calculate the simple interest by formula

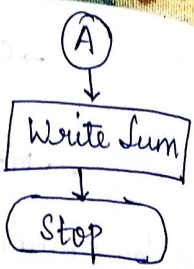
$$(P * R * T)/100 = SI$$

6) Print the SI
7) Stop

* Flow chart







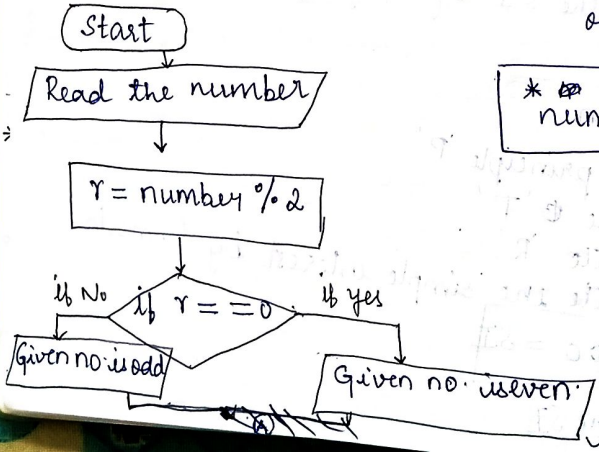
Use of connector

Modulus (%) is used/applied only on integers.

* Write an algorithm to check whether the given number is even or odd.

* → ~~Algorithm~~

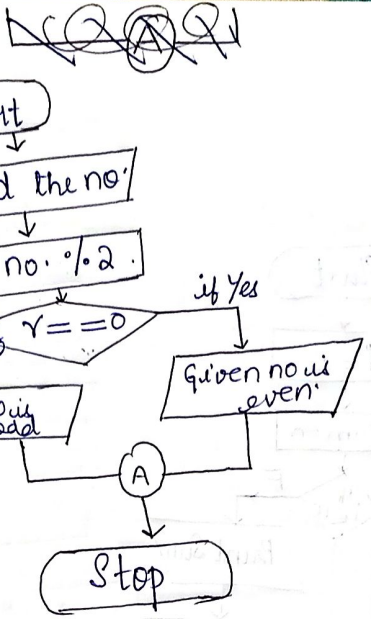
→ FLOW CHART:



"=" → Assignment operator.

"==" → Equality operator.

* num % 2 == 0



→ ALGORITHM:

1) Start
2) Read the number
3) Calculate remainder r = num % 2.
4) If r is equal to 0 then no. is even; Go to step 6.
5) If r is not equal to 0 then no. is odd.
6) Stop.

* Write the algorithm & flowchart for the sum of first 10 natural numbers.

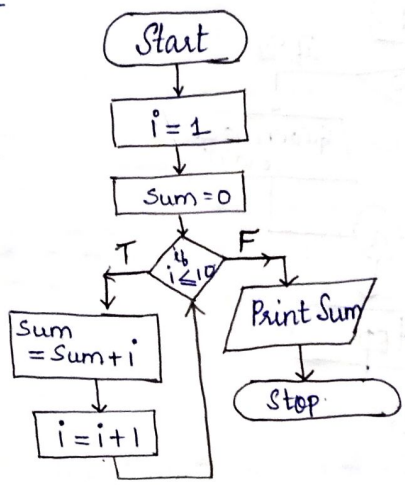
1) Start
~~2) Calculate the sum by sum~~
2) Initialize i = 1 & Sum = 0
3) Check if i > 10, Go to Step-6.
4) Compute Sum = Sum + i
5) Compute i = i + 1; Go to step - 3

done again.

6) Display the Sum.

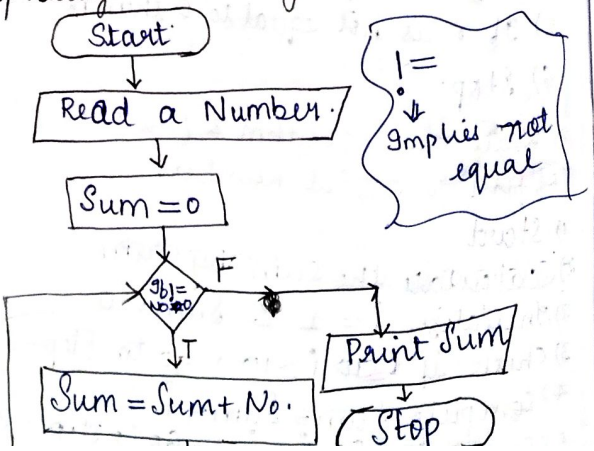
7) Stop.

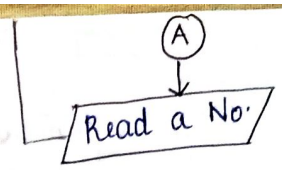
*FLOWCHART



17/12/2021

*Draw the flowchart to calculate the sum of given numbers. Stop adding when the given no is 0.





*Relational operators: <, <=, >, >=, ==, !=

* → Dennis Ritchei developed C-Language in 1972.
→ C language is a structured programming language.

* Documentation section: to write specific comments. Specify title, name, date.

⇒ /* program to add 2 numbers.
Date: 17/12/2021.
Author: Sivani */ } Multi-line comment

⇒ // program to add 2 numbers
// Date: 17/12/2021
// Author: Sivani } Only single line comment

*Pre-processor Section: write pre-processing commands → Link section
→ definition section

⇒ Starts with # Symbol.

#include standard input
#include <stdio.h> → Link section

#define PI 3.14. → Definition section to define constants.

* Global declarations:

```
int main ( )                    // main function
{
   // local declarations
   // program statements
}
```

User defined functions

function - 1

function - 2.

*

18/12/2021

C- Programming Language :/

Designed & developed by Dennis Ritchie in 1972 @ AT&T Bell Labs, USA.

ALGOL → BCPL → B → C

- Easy to understand
- Supports structured programming
- Rich set of features of programming components making it to us in developing a variety of applications
- portable.

C - language character Set:

- All alphabet < uppercase — A to Z
  lowercase — a to z.
- digits :— 0 to 9
- Symbols : +, −, *, /, #, * - - - -
- escape characters: \n (new line character), \t (tab (horizontal)), \a, \\, \", \', \v (vertical tab)
- white spaces
- * Words < keywords
  Identifiers

"% %" → %%

\" → "

\b, \r → carriage return

* Keywords: is a fixed name given by the c-developers for a specific purpose.
We can just use the keywords but we cannot modify them.
Eg: int, float, void, auto, char, if, static, else, why, for, return etc.

* identifier: are the names given by the users for a programming components
Eg: Variables, array, Structure, function etc.

* Identifier naming rules:

1) Name can have alphabets, digits and underscore.
2) First character should be an alphabet (or) underscore.
3) white spaces are not allowed.
4) Keywords cannot be used as identifier name.
5) Names are case - sensitive [upper differs with lower]
   Name / name → different.
6) Usually the length should be as small as 8 - characters (preferably)
7) Name should be meaningful.

Ex: Num, x, sum; average; dob; rno; r_no; dob2001;

* Identify the valid & invalid names:

1) VCE ✓
2) VCE HYD X
3) VCE_1981 ✓
4) myname ✓
5) 1CSE X — CSE ✓
6) sum_average ✓
7) roll no X
8) eid@vce.com X
9) section -A X
10) X
11) a ✓
12) Section ✓
13) 2021_courses X
14) return X return_rno

* C - constants:

Fixed value which cannot be changed.

2 types → Numeric → integers - 21, 63, 45 (int)
→ real nos - 1.25, 3.76, (float & double)
→ character → single - 'A', 'x', '3' (char)
→ string - "cse-A"; "A";

* TOKENS:

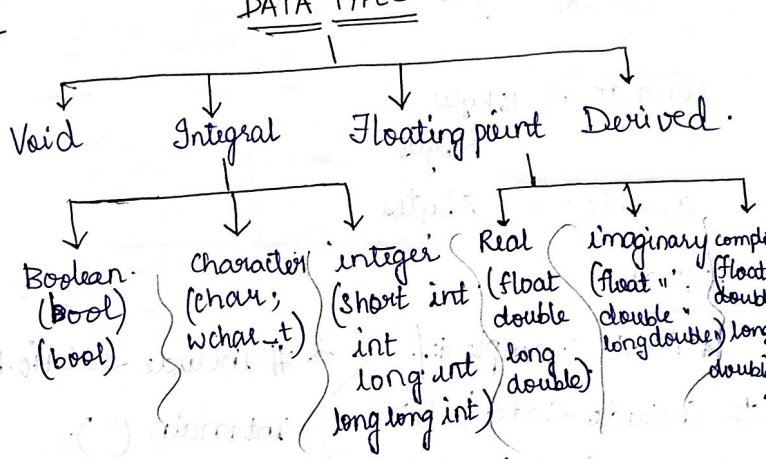
- Key words → reserve words.
- constants / literals.
- operators
- data-types
- identifiers (variable; function)

* Datatypes: / Type:

Primitive | Derived.

data-type / type defines a set of values that a variable might take and a set of operations that can be applied to those values.

*



* Primitive / basic / built-in data-types:

→ int → double
→ char → void
→ float

1 byte - 8 bits

- character requires 1 byte space

- void: do not pass any value.
- bool: Only 2 values - T/F
  non-zero value → T = (1)
  zero → F = (0)

	Space	Min Value	Max Value
Short int	2 bytes	-32,768	32,767
int	4 bytes	-2147423648	2,14,748,3647
char	1 bye		
long int	4 bytes		
long long int	8 bytes		
long double	16 bytes		
float	4 bytes		
double	8 bytes		

20/12/2021

```
/* Print Example */
# include <stdio.h>
 int main( )
{
int  b=10;
 int a;
 a = b+35;
 printf ("a= %d", a);
return 0;
}
```

const :→ cannot change it further

2 byte = 8 bit

```
# include <stdio.h>
 int main ( ).
 {
 int, cpi
 const float  pi = 3.145
 pi = pi + 0.006;
 printf ("pi = %f", pi);
 return 0;
 }
```

keyword

```
# include <stdio.h>
# define pi 3.14        → macro name
  int main( )
{
float  radius = 2.5;
float  area; P
   area = pi * r * r / pi * radius * radius;
   P = 2 * π * radius
printf ("area = %f", area);
 return 0;
 }
```

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

* Format Specifier;

%c
%ld
%d

%ld.

%lld

%f.

%lf (or) %g. → 15 decimals.

%Lf → 19 decimals

* OPERATORS:

1) Arithmetic Operators: +, −, *, /, %

2) Relational Operators: >, >=, <, <=, ==, !=

3) Logical Operators: &&; || ; ! → logical not

(a>b)&&(a>c)

↓ logical and ↓ logical or

- Increment & decrement operators → Apply only on the variables.

(++) (−−)

```
*
int a=10
a++;
printf("%d", a);
```

a=a+1

```
* int b=10
b--
printf("%d", b)
```

b=b-1

*5

- Post increment ; Pre-increment (++a)
- Post decrement (b++) ; Pre-decrement (−−c)

(d−−)

} Only for variables.

```
* int a=10;
  x=a++;
printf("x=%d a=%d", x, a)
```

x=a; a=a+1

x = b+++a
x = 5 16 11

x = b+a++
x = 15 10

```
* int a, x
  a=10
  x=++a
printf("x=%d, a=%d", x, a).
```

a=a+1. x=a;

Size of → requires 1 operand → unary.

```
* int a, x
  a=10
  x=a--
```

4) Assignment Operators: =;

Shorthand assignment operators:

+=, −=, *=, /=, >>=, <<=

a−=10 ⇒ a=a−10.

a+=1 ⇒ a=a+1

* x+=y*10 ⇒ x=x+(y*10)

Bitwise operators

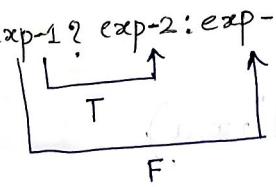
22/12/2021

6) Conditional operators / ternary (3) → Requires 3 operands

(?:)

```
int a=5; b=6; x;
x = a>b ? a : b;
```

Expression is a sequence of operators & operands which evaluates to a single value.



Logical and & or are binary operators

* Greatest no. of 3 nos.

```
int a,b,c, max, temp;

    a=10;
    b=5;
    c=12;

temp=a>b?a:b;
max = temp>c ? temp:c;
    printf("greatest=%d", max);
a>b ?(printf("%d is greater")a);
```

7) Bitwise Operators:

& → Bitwise AND operator.
| → Bitwise OR operator.
^ → Bitwise XOR operator.
~ → Bitwise Negation Operator.
<< → left shift operator (x2)
>> → Bitwise Right Shift Operator (÷2)
} Binary operators.

CHECK:

```
printf("%d", a&&b);  scanf("%d",a)
printf("%d", !a);
```

*int a=5; b=2; → printf("%d", a&b);.

```
a  00000101
b  00000010
a&b 00000000
```

*int a=5 b=b

```
a  00000101
b  00000110
a|b 00000111
```

*int a=5; b=6

```
a  00000101
b  00000110
a^b 00000011
```

```
*a  00000101
~a  11111010
```

* a: 00000101 → 5.
a<<1: 00001010 → 10

a<<2: (left shift) ⟹ Multiplied by 2

00010100 → 20

```
*   a     00000101
    a>>1  00000010 → 2
```

⟹ divide by 2 (integral value) (Right shift

* & → address of

EXPRESSION → Simple ⇒ 5
Unary. ⇒ a+1
Binary
Ternary.
Postfix
Prefix.

$1 * \frac{(3+4)}{2} - 6 * 4$

$= 1 * \frac{7}{2} - 6 * 4$

$= \frac{7}{2} - 6 * 4 = 3 - 24 = -21$

24/12/2021

Precedence is used to determine the order in which different operators in a complex expression is evaluated.

Associativity: is used to determine the order in which operators with same precedence are evaluated in a complex expression.

Side effect: is an action that results from an evaluation of an expression.

Eg: changing the value of ~~a~~ left operand in an assignment expression.

Variables: are named memory locations that have a type; the type determines the values that a variable may contain and the operations that may be used with its values.

- **Variable declaration**:

```
datatype variablename;
" int no1;
  float cgpa;
  int no1, no2;
(or) int no1;
     int no2;
```

- **variable initialization**:

```
datatype varname = value;
eg: float sum = 0.
    charcode = 'g';
• int count, sum = 0;
      (or)
  int count;
  int sum = 0;
```

* **Statements**:

```
Null    ;
Exp     x = a+b;
return  return 0;
```

conditional statement:

* Write a program to read quiz marks of 5 subjects & compute the average.

```
#include <stdio.h>
int main( )
{
   float sub1, sub2, sub3, sub4, sub5;
         float sum = 0;
         float average = 0;
   printf("enter the 5 subject quiz marks") ;
   scanf("%f %f %f %f %f", &sub1, &sub2, &sub3,
                           &sub4, &sub5);
```

```
    Sum = sub1+sub2+sub3+sub4+sub5;
    avg = Sum / 5;
    printf("average quiz marks = %f", avg);
    return 0;
}
```

CONDITIONAL / SELECTION CONTROL STATEMENTS:

→ If statement
→ Nested if statement
→ if else statement
→ else - if statement
→ switch-case statement

1)
```
if (exp/cond)
{
   statements
}
```

* Write a program to check whether the candidate is eligible for voting. Accept candidate age as input.

```
# include <stdio.h>
int main( )
{
   int age;
   printf("enter your age");
   scanf("%d", &age);
   if (age >= 18)
   {
      printf("eligible to vote");
   }
   return 0;
}
```

2)
```
if (exp/cond)
{
   statements 1;
}
else
{
   statements 2;
}
```

```
# include <stdio.h>
int main( )
{
   int age;
   printf("enter your age");
   scanf("%d", &age);
   if (age >= 18)
   {
      printf("eligible to vote");
   }
```

```
else
 {
 printf("Not eligible to vote, wait for %d years",
                                        (18-age));
    }
   return 0;
 }.
```

27/12/2021

*

3) if (cond1)

```
   {
     if (cond2)
      {
        statements;
      }
    }
   else
     {
     statements;
     }
```

Write a program to check whether Bhupathi's height is less than, equal to (or) greater than Sai Krishna's.

```
/* Print the Heights */
#include <stdio.h>
int main ()
{
float Bhupathi_height, SaiKrishna_height;
printf ("Enter Bhupathi's height");
scanf ("%f", &Bhupathi_height);
printf(" enter SaiKrishna's height");
scanf ("%f", &SaiKrishna_height);
if (Bhupathi_height <= SaiKrishna_height)
  {
    if (Bhupathi_height < SaiKrishna_height)
      {
      printf ("SaiKrishna is taller");
      }
   else
      {
       printf ("Both are of equal height");
      }
  }
else
  {
   printf ("Bhupathi is taller");
  }
return 0;
}.
```

To get rid of Dangling else problem; we need to place flower brackets properly.

4) Else-if ladder:

```
if (cond 1)
  {
  statements 1;
  }
else if (cond 2)
  {
  statements 2;
  }
else if (cond 3)
  {
  statements 3;
  }
else
  {
  statements -4;
  }
```

Ladder form

* Write a program to find the minimum CGPA;
Given the CGPA of 3 students.

```
/* Print Min CGPA */
#include <stdio.h>
int main()
{
float S1_CGPA, S2_CGPA, S3_CGPA;
printf(" Enter 3 students CGPA");
scanf("%f, %f, %f", &S1_CGPA, &S2_CGPA, &S3_CGPA);
if (S1_CGPA < S2_CGPA)
  {
  if (S2_CGPA > S3_CGPA)
    {
    printf("Min : S3");
    }
  else
    {
    printf("Min : S2");
    }
  }
else if (S1_CGPA > S3_CGPA)
  {
  printf("Min : S3");
  }
else
  {
  printf("Min : S1");
  }
}
```

(OR)

```
if (S1_CGPA<S2_CGPA && S1_CGPA<S3_CGPA) -
 {
 printf("Min CGPA : S1");
 }
else if (S2_CGPA<S1_CGPA && S2_CGPA<S3_CGPA)
     {
      printf("Min : S2");
     }
  else
    {
    printf (" min : S3").
    }
    return 0;
  }.
```

29/12/2021

5) Switch case:

```
Switch (exp)
{
 case const 1 : statements
                ==
                break;
 case const 2 : statements
                ==
                break;
 case const n : statements
                ==
                break;
 default : statements
 }
```

constants:
integer constants
character constants
only

"Switch" comes to an end when it encounters the break (or) at the end of switch.

* when no condition matches, it executes default statements.

* Write a program to check whether the given no is even or odd using switch case.

```
/* Print Even(or) Odd */
# include <stdio.h>
int main ( )
{ int no;
 printf (" Enter a no.");
 scanf ("%d", &no);
 Switch (no. % 2)
 {
  case 1 : printf("%d is odd", no);
           break;
  case default : printf("%d is even", no);
 }
```

```
return 0;
}
```

* Write a program to check whether the given alphabet is vowel or consonant.

```
/* Print Vowel & Consonant */
#include <stdio.h>
int main ()
{
 char ch;
printf("Enter the alphabet");
scanf("%c",&ch);
switch(ch)
{
 case 'a': printf("vowel");
           break;
 case 'e': printf("vowel");
           break;
 case 'i': printf("vowel");
           break;
case 'o': printf("vowel");
           break;
 case 'u': printf("vowel");
           break;
  default : printf("The given alphabet is a consonant");
 }
 return 0;
}
```

(OR)

```
switch(ch)
{
 case 'a':
 case 'e':
 case 'i':
 case 'o':
 case 'u': printf("vowel");
           break;
 }
```

* if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))

* if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))

```
 {
  switch (ch)
  {
      case 'A':
       case 'a':
```

```
case 'E':
case 'e':
case 'I':
case 'i':
case 'O':
case 'o':
case 'U':
case 'u': printf("Vowel");
          break;

default : printf("The entered alphabet is a
                  consonant");
  }
}
else
 {
  printf("Not an alphabet");
 }
return 0;
}
```

⇒ isalpha( ) #include <ctype.h>

* if(isalpha(ch)).

{

switch. Remaining program.

```
if (! isalpha(ch))
  {
   printf ("not an alphabet");
  }.
else
{switch (ch)
 {
  case 'a':
  case 'e':
  case 'i':
  case 'o':
  case 'u': printf("vowel");
            break;
  default : printf("entered alphabet is consonant");
  }
 }return 0; }
```

* Write a program to display day name in the week given the day number.

```
# include <stdio.h>
int main( )
{
 int day_no;
 printf("Enter the day number");
 scanf("%d", &day_no);
 switch (day_no)
```

```
{
case 1: printf("Sunday");
        break;
case 2: printf("Monday");
        break;
case 3: printf("Tuesday");
        break;
case 4: printf("Wednesday");
        break;
case 5: printf("Thursday");
        break;
case 6: printf("Friday");
        break;
case 7: printf("Saturday");
        break;
default: printf("Invalid number");
}
return 0;
}
```

*Write a program to accept the marks of student (out of 100) & display the corresponding grade:

```
#include <stdio.h>
int main()
{
int marks, m;
printf("enter marks.");
scanf("%d", &marks);
    m = marks/10;
switch(m)
{
case 10: printf("A");
        break;
case 9: printf("A");
        break;
case 8: printf("B");
        break;
case 7: printf("C");
        break;
case 6: printf("D");
        break;
case 5: printf("E");
        break;
case 4:
case 3:
case 2:
case 1:
case 0: print("FAIL");
        break;
}
```

```
a=5  b=6
a=a+b;
b=a-b;
a=a-b;
```

return 0;
}

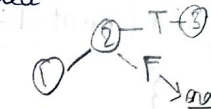
31/12/2021

Q: Write a program to accept 2 numbers & the choice. Perform the arithmetic operations based on the choice using Switch case.

```
/* Print Sum */
#include <stdio.h>
int main ()
{
    int  no1=20;
    int  no2=30;
    char choice;
printf(" Enter your choice");
scanf("%c",&choice);
switch (choice)
{
case '+' : printf( "50");
           break;
case '-' : printf ("-10");
           break;
case '*' : printf ("600");
           break;
case '/' : printf ("1.66");
           break;
}return 0; }
```

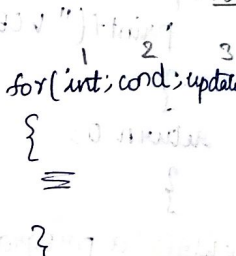
LOOP CONTROL STRUCTURE If something must be repeated.

→ for loop } pre-test loop
→ while loop }
→ do-while loop → post test loop.



⇒FOR LOOP:

Syntax: for (exp1; exp2; exp3)
{
loop body
}




```
*
#include <stdio.h>
   int main ()
  {
  int count;                    count++
  for (count= 1; count <6 ; count= count+1);
    {
     printf ("VCE");
    }
   return 0;
   }.
```

* Write a program to print VCE 10 times.

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

```
int count;
for(count=1; count<11; count=+1);
 {
  printf(" VCE");
 }
return 0;
 }
```

* Write a program to print first 20 natural numbers.

```
/* Print Numbers*/
#include <stdio.h>
int main ( )
{
 int count;
 for (count=1; count<21; count +) -
  {
   printf("%d\t"
   ,c);
  }
 return 0;
 }
```

* Write a program to print even numbers from 0 to 20.

```
/* Print Even Numbers*/
#include <stdio.h>
int main ( )
{
 int even_no;
 for (evenno=0; evenno<=21; evenno+=2).
  {
   printf ("%d\t", evenno);
  }
 return 0;
 }
```

* Write a program to display multiples of 5 from 50 to 0.

```
/* Print Multiples of 5*/
#include <stdio.h>
 int main ( )
{
 int num;
for(num = 50; num < 50
{
 int num;
 for(num =50; num>=0&&num<=50; num-=5)
 {
  printf("%d", num);
 }
 return 0;
}
```

05/01/2022

* Write a program to find the sum of the digits of the given integers.

```
# include<stdio.h>
int main ( )
{
int no, num, sum, rem
printf(" Enter an integer ") ;
scanf("%d", &no);
for (sum =0, num =no; num!= 0; num/=10.
    {
    rem = num %10;
    sum += rem;
    }
printf("%d", sum) ;
return 0;
}
```

* Write a program to calculate the factorial of a given number.

```
# include<stdio.h>
int main ( )
int no, p;
printf(" Enter a no ");
scanf ("%d", &no);
for (p=no.; p!=0 ; p+=1); no, f, x.
{
printf("%d", p);
}
return 0;
}
for (p=no; p!=0; p+=1).
{
 p= p*no ;
}
printf("%d", p) ;
return 0;
}
```

for (f=1, x=1 ;
f != no+1,
f+=1).

→ WHILE LOOP:

```
while (expression)
{
 // statements
}
```

* Write a program to find the sum of digits of the given integer.

```
#include <stdio.h>
int main ( )
{
 num=no;
 sum=0;
 while (num!=0)
 {
   sum = sum + num%10;
   num = num/10;
 }
return 0;
}
```

```
# include <stdio.h>
int main
{
 int no, num, sum;
 printf("enter the no");
 scanf("%d", &no);
 num = no;
 sum=0;
 while (num!=0)
 {
   rem = num % 10;
   sum = sum + rem
   num = num/10;
 }
```

* Factorial of a number:

```
#include <stdio.h>
int main ( )
{
 int fact, i, no;
 printf("enter the no.");
 scanf("%d", &no);
 fact = 1;
  i = 1;
 while (i<=no)
 {
  fact = fact * i;
   i++;
 }
 printf("factorial of %d is %d", no, fact);
return 0;
}
```

*Write a program to find the sum of given nos. Stop accepting the nos when it is -1.

```
#include<stdio.h>
int main()
{
int no, sum=0;
printf("Enter the no:");
scanf("%d",&no);
while(no!=-1)
{
Sum=Sum+no;
printf("enter a no");
scanf("%d",&no);
}
printf("sum=%d", sum);
return 0;
}
```

07/01/2022

do-while loop:

```
do
{
//statements;
}
while(condition);
```

Write a program to print first 10 natural numbers.

```
#include<stdio.h>
int main()
{
int i=1;
do
{
printf("%d\n",i);
i++;
}while(i<=10);
return 0;
}
```

* Write a menu driven program to perform arithmetic operations on given 2 nos. based on the users' choice.

```
#include<stdio.h>
 int main ( )
 {
int no1, no2, choice;
printf("enter 2 numbers");
scanf("%d%d", &no1, &no2);
printf("1.add\n 2.sub\n 3.multiply\n 4.deivide\n
          5.exit");
printf(" enter your choice");
scanf("%d", & choice);
switch(choice)
{
 case 1: {printf("sum= %d", no1+no2);
          break;
 case 2: printf("diff.= %d", no1-no2);
          break;
 case 3: printf("product= %d", no1*no2);
          break;
 case 4: printf("divident= %d", no1/no2);
          break;
case 5: printf(" exit");
          break;
default : printf("Wrong choice");
          break;
} while(choice!=5);
 return 0;
}
```

* Menu Driven Program : includes Switch case.

11/01/2021

Printf

```
# include <stdio.h>
 int main( )
 {
 printf("%d", printf("Hello World\n"));
  return 0;
 }
```

Scanf

```
# include <stdio.h>
 int main
 {
  int i
  printf("\n %d", scanf("%d", &i));
  return 0;
 }
```

An infinite loop is a looping construct that does not

*

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

```
for(a=1;a<=5;a++)
 {
int for(b=1;b<=a;b++)
  {
  printf("%d",a);
  }
```

*

```
#include <stdio.h>
int main( )
 {
 int a,b,c;
 for(a=0,b=.12,c=23;a<2;a++)
  {
  printf("%d",a+b+c);
  }
 return 0;
}
```

Syntax of nested loop.

```
Outer loop
 {
 Inner loop
  {
  statements
  }
  statements
 }.
```

Syntax of Break Statement :

```
break;
```

13|01|2021

JUMP STATEMENTS:

1) Break <- switch, loops.
2) Continue
3) goto
4) return

* while(cond)
```
 {
 ≡
 break;
 ≡
 }
 ≡
```

* do
```
 {
 ≡
 break;
 ≡
 }while(cond);
```

* for(cond1;cond2;cond3)
```
 {
 ≡
 break;
 ≡
 }
 ≡
```

* for(——)
{
≡
for(——)
{
≡
break;
≡
}
}

```
*for(i=0; i<10; i++)
{
printf("%d", i);
if(i==5);
break;
}
```

OUTPUT:
012345

```
*for(i=0;i<10;i++)
{
printf("%d", i);
}
```

```
int i=0
* while (1)
{
printf("%d", i);
i++
if(i==10);
break; ——→ not given
}
printf("end");
```

it turns to be infinite loop

OUTPUT
0123456789

* CONTINUE: SYNTAX: continue;

```
while (cond)
{
≡
continue;
≡
}
```

```
do
{
≡
continue;
≡
}while(cond);
```

```
* for(i=0;i<10;i++)
{
≡
continue;
≡
}
```

```
* for(i=0; i<=10; i++)
{
if(i==5)
continue;
printf("%d", i);
}
```

OUTPUT
012346789

* goto:

goto label; ⟹ Syntax

forward jump

```
≡
goto label
≡
label : statements
```

backward jump

```
label : statements;
≡
goto label;
≡
```

```
* int num1=1, num2=5
table:
printf("%d", num1 * num2);
num1++;
if(num1<=10)
goto table;
}
```

OUTPUT: 5 10 15 20 25 30 35 40 45 50.

* STORING INTEGERS:

**

- Unsigned Integers
- Signed Integers

UNSIGNED INTEGERS: 4bits

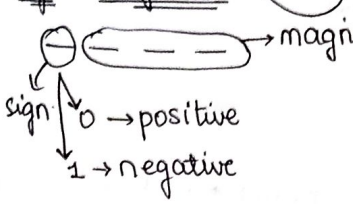
Store in binary values

0000 – 0	0101 – 5
0001 – 1	0110 – 6
0010 – 2	0111 – 7
0011 – 3	1000 – 8
0100 – 4	

SIGNED INTEGERS: (Eg: 3, -3)

1) Sign and magnitude method
2) One's complement method
3) Two's complement method
4) Excess System

Sign & Magnitude (4bits)



Used to store analog & digital signals.

+0 → 0000	−0 → 1000
+1 → 0001	−1 → 1001
+2 → 0010	−2 → 1010
+3 → 0011	−3 → 1011
+4 → 0100	−4 → 1100
+5 → 0101	−5 → 1101
+6 → 0110	−6 → 1110
+7 → 0111	−7 → 1111

One's complement

5 − 0101

one's complement − 1010

+0 → 0000	−4 → 1011	
+1 → 0001	−3 → 1100	
+6 → 0110	−2 → 1101	
+7 → 0111	−1 → 1110	
−7 → 1000	−0 → 1111	
−6 → 10001		
−5 → 1010		

−7 = negation of (One's complement (7))

+3	0011
+(+2)	0010
+5	0101

+3	0011
+(−2)	1100
+1	0000
	+ 1
	0001

Two's Complement:

Two's complement = one's complement + 1

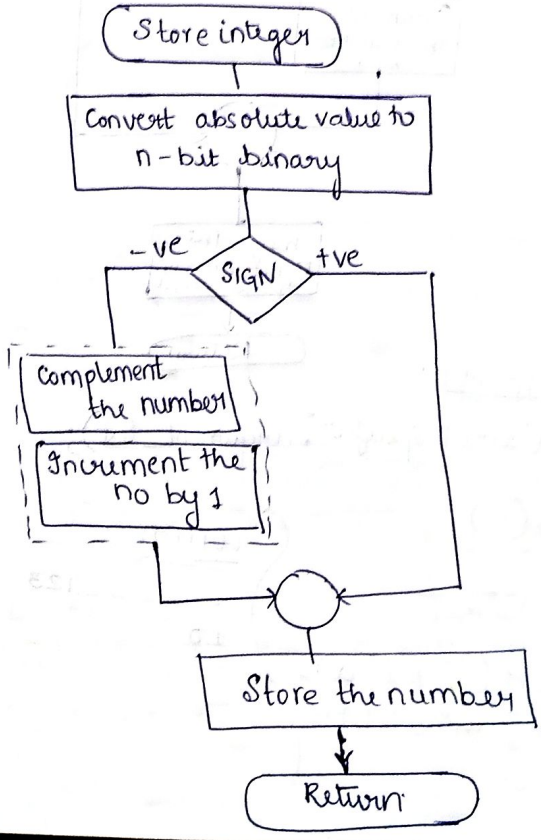
+5 0101

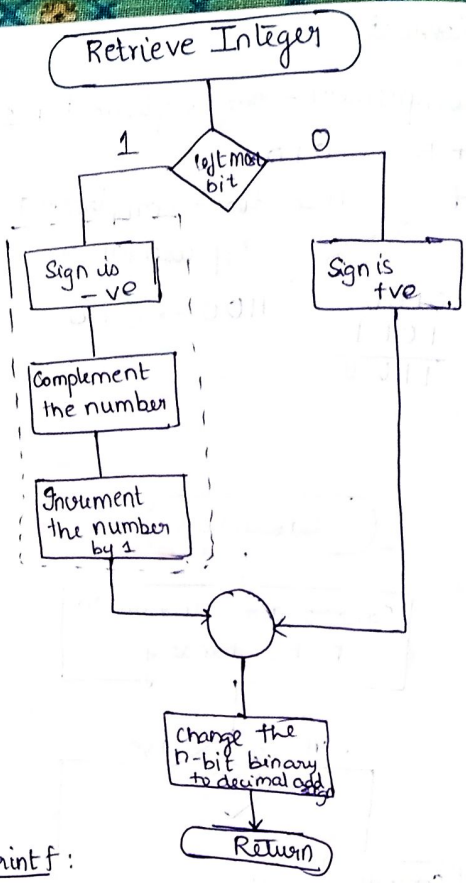
−4 use two's complement representation

4 : 0100 → 1011 → 1100

1100 → −4 in two's complement

18/01/2022





Syntax of printf:

printf ("format specifier", argument_list);

```
int main( )
{
 int a = 123;
 int b;
 b = printf ("%10d", a);
 printf ("\n%d", b);
 return 0;
}
```

OUTPUT:

```
       123
10
```

→ b = printf ("%010d", a); 0000000123

→ b = printf ("%+-010d", a); +123

Syntax of scanf:

scanf("format specifier", argument_list);

```
int main ( )
{
 int a=123;
 float b=25.6789;
 char c;
 printf("%f", b);
 return 0;
}
```

→ 25.678900

⇒ printf("%8.2f", b); ⟶ - - - 25.68

⇒ printf("%5.2f", b); ⟶ 25.68

```
* int main ( )
 {
  int a=123,no;
  flood b=25.6789;
  char c;
  scanf("%d,%d",&a &no);
  printf ("a=%d  no=%d", a,no);
  return 0;
 }
```

12,35

a=12 no=35

scanf("%d-%d",&a,&no); ⇒ 3-5

a=3 no=5.

scanf("%d/%d",&a,&no); ⇒ 3/5

a=3 no=5

```
int main ( )
 {
  int a=123,no,n;
  float b=25.6789;
  char c;
  n=scanf("%d/%d",&a,&no);
  printf("a=%d  no=%d",a,no);
  return 0;
 }
```

3/5

a=3 no=5

18/01/2022

FUNCTIONS:

Syntax of function prototype:

returnType functionName(type1 argument1, type2 argument2,..);

Function definition:

contains the block of code to perform a specific task; or

SYNTAX

returnType functionName(type1 argument1, type2 argument2,..)

25/01/2022

*Write a program to define a function that displays square of a given number.

```
#include <stdio.h>
void square(int n);
int main()
{
  int no;
  printf("Enter a no");
  scanf("%d",&no);
  square(no);
  return 0;
}
void square(int n)
{
  printf("sq=%d", n*n);
}
```

(not compulsory to give; need not return anything; function prototype; function call; actual parameter; can give a diff name (compulsory); Function definition)

OUTPUT
Enter a no 5
sq = 25

*Write a function to print square of a number and return it.

```
#include <stdio.h>
int square(int x);
int main()
{
  int no; int s;
  printf("Enter a no");
  scanf("%d", &no);
  s = square(no);
  printf("sq = %d", s);
  return 0;
}
int square(int x)
{
  return (x*x);
}
```

(or)

```
int square(int x)
{
  int n;
  n = x*x;
  return(n);
}
```

(space is allocated; separate; int s)

function call ⟶ function definition

OUTPUT
Enter a no 3
sq = 9

*Write a program to calculate $(20)^5$ and display. [Hint: Write a function that calculates 20^5 and displays it].

```
#include <stdio.h>
void power();
int main()
{
  power();
  return 0;
}
```

```
void power ( )
{
long int P;
P=20*20*20*20*20;
printf("%ld", p);
return;
}
```

```
#include<stdio.h>
long int power(void);
int main( )
{
long int n;
n = power ( );
printf("%ld",n);
return 0;
}
long int power(void)
{
long int p;
P= 20*20*20*20*20;
return p;
}
```

```
int main()
{
printf("%ld",power());
return 0;
}
```

⇒ no parameters; no return type.
noparameter; returns a value.
parameter; no return type.
parameters, returns.

* Write a program to calculate factorial for a given function.

```
#include <stdio.h>
long int factorial(int x);
int main( )
{
int no; long int f;
printf("Enter a no");
scanf("%d", &no);
f= factorial(no);
printf("factorial of %d is %ld", no,f);
return 0;
}
long int factorial(int x)
{int i; long int f=1;
for(i=1; i<=x, i++)
{
f=f*i;
}
return f;
}
```

27/01/2022

→ A function is an independent or seperate module i.e. called to do a specific task.

→ A called function receives control from calling function when the called function completes its task it returns the control back to calling function.

→ It may / may not return a value to the caller.

* Formal & actual parameters.

• Formal parameters are the variables that are declared in the header of the function definition

Actual parameters are the expressions in the calling statements.

• Formal & actual parameters must match exactly in type, order and number.

Ex.

```
# include <stdio.h>
int getno( );
int cube(int);
void display(int x);
int main( )
{
    int n,c;
    n=getno( );
    c=cube(n);
    display(c);
    return 0;
}
```

c=cube(5);
display(c);
c=cube(2*3);
display(c);

```
int getno( )
{
 int no;
 printf("Enter a number");
 scanf("%d",&no);
 return no;
}
int cube(int x)
{
 return (x*x*x);
}
void display(int x)
{
 printf("%d",x);
}
```

OUTPUT

```
Enter a number 7
343
125
216
```

* Write a program to multiply 2 numbers using functions.

```
# include <stdio.h>
double multiply(double a, double b);
int main( )
{
 double res, a;
 res = multiply(2.5*5.5);
 printf("product = %lf", res);
 return 0;
}
```

OUTPUT

product = 13.750000

```
double multiply(double d1, double d2)
{
 return(d1*d2);
}
```

OUTPUT
13.750000

*Write a program to swap the content of 2 variable using user defined function.

```
#include <stdio.h>
void swap(char ch1, char ch2);
int main()
{
char c1='A';
char c2='F';
printf("Before function call c1=%c c2=%c\n",c1,c2);
swap(c1,c2); / swap(&c1,&c2) → call by reference
printf("After swapping c1=%c c2=%c\n", c1, c2);
return 0;
}
```

→ function header

```
void swap(char c1, char c2) / swap(char *c1, char *c2)
{
 char ch3;
 ch3=*y;
 *y=*x;
 *x=ch3;
 printf("In the swap function, c1=%c, c2=%c\n",*x,*y);
}
```

Function body

c1 = A, c2 = F

*
There are 2 ways of passing the value:
pass by value (call by value)
pass by reference (call by reference).

* Recursive function:
Recursive function is a function which calls itself.

```
int fact(int x).
{
 int i, f=1;
 for(i=1; i<=x; i++)
 {
  f=f*i;
 }
```

⇒

```
int fact(int n)
{
 if(n==0 || n==1)
  return 1;
 else
  return(n*fact(n-1));
```

Recursion is a repetitive process in which a function calls itself.
Every recursive call must either solve part of the problem or reduce the size of the problem.
and every recursive function should have base case.

*TOWERS OF HANOI:

*INTER FUNCTION COMMUNICATION:

Although the calling & called functions are 2 separate entities, they need to communicate to exchange data. The data flow between the calling and called functions can be divided into 3 categories: downward flow, upward flow and a bi-directional flow.

→ Standard functions whose definitions have been written and are ready to be used in program

→ Scope: Scope determines the region of the program in which defined object is visible.

* Variables defined within the block, will have only local scope.

#

* Variables are in scope from declaration until end of the block.

* Global Scope: An object defined in the global area of a program is visible from its definition until the end of the program.

* STORAGE CLASSES:

There are 4 storage classes.
→ Automatic Storage class. (auto).
→ Static Storage class (static)
→ External storage class (extern)
→ Register Storage class (register).