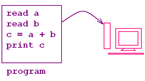
C Language Notes

By Sagar Uppuluri

***Note : Few operations like clrscr, getch, conio.h etc may not relevant or need to removed. You need to replace with standard library functions where ever applicable. Content is written based on 16-bit machine architecture.

Q: What is a program ?
Ans:

A program is a sequence of instructions given to computer to achieve certain goal.



Computer goes through these instructions one after another and executes each instruction.

Q: How it executes the above program?
Ans:

Step 1: first instruction is read a. Then it waits for one value when given computer stores it in a.

Step 2: second instruction is read b Then it waits for one value when given computer stores it in b.

Step 3: third instruction c = a + b then it adds value in a and value in b and stores it in c.

Step 4: fourth instruction print c it prints the value in c on the monitor.

This is how we have to provide instructions to the computer to achieve our targets.

Q: In which language we have to instruct computer?
Ans:

There are many languages such as C,C++,Java etc. But computer understands only one language i.e. Machine Language.

Q: What is Machine Language ?
Ans:

A language which is composed using 0's and 1's. Instructions would look like this

10001010000101101

It is tough to learn !

Yes, hence languages such as C,PASCAL etc are invented. These languages use instructions that we use normally.

e.g. c = a + b;

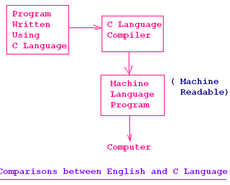
Does computer understand this ?

No, it is humans who understand it easily.

What about computer ?

Translators such as compilers are designed for the sake of computer. Compiler will translate the program written in highlevel language into machine language.

e.g.



Comparisons between English and C Language

English	C Language
1)It has alphabet (A...Z)	1) It has character set. Set of recognized characters
2)It has meaning ful words.	2) It has meaningful instructions
3) A meaningful combination of words forms a sentence.	3) Here,it is a Statement that ends with semi-colon(;)
4) Sentence has formation rules.	4) For statement it is syntax that defines how it should be formed.
5) Combination of sentences form a document which serves some purpose.	5) Combination of sentences form a program which serves some purpose.

Your first C Language Program :-

First.c

```
#include <stdio.h>

void main()
{
    printf("Hello");
}
```

Entry point for program execution.

Statement ends with semi-colon.

When we submit this program for execution we will find Hello displayed over the monitor.

Output:-

Note: This First.c is submitted to the C Language compiler which produces executable code and this executable code is submitted to computer for execution.

Exercise :-

1) Write a program to print "Hello how r u"

Next DatatypesAndDeclarations

Datatypes :-

Datatype represent the type of value that we are going to store. It also defines the data and operations over the data.

Data is computer is broadly divided into three types

* integers
* real number (floating types)
* characters

Datatypes supported in C language :-

Various data types supported in C language are

1) char occupies 1 byte memory
2) int occupies 2 bytes in memory (machine dependent)
3) short int : 2 bytes
4) long int : 4 bytes
5) float : 4 bytes
6) double : 8 bytes
7) long double : 10 bytes.

Note: byte represents an 8 bit data and one bit implies the smallest memory location that can store either 0 or 1.

General Example :-

We choose the container based on the content type.

E.g. when we want to store 50ml water then we take a glass.

Similarly when we want to place fruits we take a bag.

** Hence we can say that container is selected based on the type of content.

Now when we want to store an integer choose a memory location that will be able to hold an integer. And name the location so that it can referenced further.

```
int  a ;
```

creates a memory location which can hold some integer value.

i.e.

a is the name for this location.
a is called identifier (id)

To store some value in a

```
a = 5;
```

this means store value 5 in the memory location represted by a.
i.e. assign 5 to a.

After this statement

```
a = 8 ;
```

it replaces value 5 in a to value 8

Since we are able to change the value this type of memory location is referred as variable.

Note: Technically a is referred as an integer variable. As the name implies it can hold some integer value and it is variable i.e. changable.

Q: When we assign 5.4 to a then what happens?

Ans:

Since a is integer variable it can store only integers. Though we say

```
a = 5.4;
```

only 5 is saved in a. And 0.4 is truncated.

Q: How to store 5.4 ?

Ans:

Take float or double variable.

e.g.

```
float b;
b = 5.4;
```

(or) `float b = 5.4;`

Q: What is the difference between float and double ?

Ans:

The word float comes from floating point variable and double from double precision floating point variable.
double is more accurate than float. When we store 5.4 in float variable the actual value stored is 5.399999 where as it is 5.4 in double.

i.e.

```
float x = 5.4;
  value stored in x is 5.3999

double y = 5.4;
  value stored in y is 5.4
```

Q: If variable is changable then is there any other block whose value can not be changed?

Ans:

Yes, memory location whose value can not be changed is referred as constant.

Example :

```
const double PI=3.1417;
```

See const keyword, it represents constant. PI is the name of the memory location.

Since it is declared as constant it can not be changed.

i.e. PI = 2.0; ✗

Standard: When a constant is named only capital letters are used.

Various format specifiers used in C language for datatypes :-

char	%c
int	%d
unsigned int	%u
long int	%ld
float	%f
double	%lf
long double	%Lf

These will be covered later

string	%s
pointer	%p

Difference between signed and unsigned :-

C language comes up with two different variants of integral types i.e.(char, int,short,long)

1) signed
2) unsigned

Let us illustrate the concept with an example.

```
int a ;
unsigned int b;
```

Here unsigned keyword is not specified for a and hence it is considered as signed int a.
Both a and b occupy two bytes in memory. Since a is signed the most significat bit (left most bit) is left as sign bit and only 15 bits are used for data. Where as in b all 16 bits are used for data.

sign bit
contains 1 if number is -ve and 0 if number is +ve

Note: unsigned is not applicable for floating types.

i.e. unsigned float a; ✗

A program to read two integer values and printing their sum,difference,product and division results.

```
#include <stdio.h>

void main()
{
  int a,b,c,d,e,f;

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

  c = a + b;
  d = a - b;
  e = a * b;
  f = a / b;

  printf("sum %d" , c );
  printf("dif %d" , d );
  printf("mul %d" , e );
  printf("div %d" , f );
}
```

Memory

Memory

Memory

Prg1.c

Output :

```
Enter a 20 ↵
Enter b 10 ↵
sum 30 dif 10 mul 200 div 2
```

next arithmetic operators

Operators :-

Consider the following example.

```
        operator
A + B
operands
```

Classification of Operators :-

We can classify operators based on the number of operands on which it operates. They are

(1) Uniary operators :-
These operators operate on single operand.

e.g. +A, -A,!A etc.

(2) Binary operators :-
These operators operate one two operands.

e.g. A+B,A-B,A*B etc.

(3) Terenary Operator :-
This operator operate on three operands. This operator is also known as conditional operator. It is ?:

condition ? expr1 : expr2

In this case when condition is true then expr1 is the result otherwise expr2 is the result.

e.g.

```
int a = 10;
int b = 20;
int c;
c = a > b ? a : b;
```

Here value is c is 20 since a>b is false.

Example :-

```
#include <stdio.h>
void main()
{
  int a,b,c;
  printf("Enter a");
  scanf("%d", &a );
  printf("Enter b");
  scanf("%d", &b );
  c = a < b ? a : b;
  printf("small is %d",c);
}
```

```
output :-  Enter a 10
           Enter b 20
           small is 10
```

Classification of operators based on the operation performed :-

Based on the operation performed by the operator they are classified into

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Bitwise operators
5) Shift operators
6) Assignment operators
7) Special operators.

Arithmetic Operators :-

Operators such as
+ , - , / , * etc.
come under this category.

Note: here * for multiplication.

E.g.

```
int a=10,b=3,c,d,e,f,g;
c = a + b;   /* c will be 13 */
d = a - b;   /* d will be 7 */
e = a / b;   /* e will be 3 */
f = a * b;   /* f will be 30 */
g = a % b;   /* g will be 1
                (reminder) */
```

Note:- Any text between /* and */ is considered as comment and it will not be processed by the compiler. It is normally written for documenting the statement or a program.

% (Modulus) operator :-

Produces the reminder.

E.g.

```
 12  %  5 => 2
-12  %  5  => -2
 12  % -5 => 2
-12  % -5 => -2

12.25 % 5 => error
```

Tip: sign of the numerator is the sign of the result.

**Note: % operator in C language is applicable only on integral types i.e. char,short,int,long.

Q: Can you explain why a/b is 3 and not 3.33 ?
Ans:

To answer this we need to know about the type conversions.

In childhood days what we have done for

```
12      12
+ 2  ==> 02     and     12.2      12.2
---     --              + 3   ==> 03.0
          14            ----      ----
                                  15.2
```

In these cases both values are translated to same type before proceeding with operation.

Similarly when integer is operated with long , integer must be converted into long and the end result will be a long value.

i.e.

```
   long                    long

(op) int    ==>         long
     ----               ----
                        long
```

When an integer is operated with float then integer is tranlated into a float and the end result will be a float value.

i.e.

```
   float                   float

(op)  int    ==>        float
      -----             -----
                        float
```

These are different combinations
* char and int => int
* int and short => int
* int and long => long
* long and float => float
* long and double => double
* float and double => double.

Let us illustrate the concept with some examples :-

Example 1 :-

```
  7     /     2
(int)       (int)
          (int)
            3
```

no need for type conversion.

Example 2 :-

```
  7.4     /     2
(double)      (int)
                 |
              (double)
                2.0
 (double)
   3.7
```

Note: 7.4 is double and not float.

Example Program :-

```
#include <stdio.h>
void main()
{
  int a=7,b=2;
  double c;
  c = a / b;
  printf("c is %lf",c);
}
```

Prg2.c

```
output :-
  c is 3.0
```

Explanation :-

```
  c     =     a     /     b
(double)      (int)   (int)
                 |
              (int)
                3
                 |
              (double)
                3.0
```

***Note: In all the above examples C language compiler will automatically perform these type conversions. Hence this type of conversions are referred as implicit conversions.

Before proceeding forward let us discuss the priority of operators :-

Note: DONT APPLY BODMAS RULE IN COMPUTERS

Priority sequence of important operators
(1) ()
(2) / * % => all have same priority.
(3) + - => " "

Numbers in the following examples indicate evaluation order :-

Ex1:-

```
  ①     ②
a + b - c
```

Since both + and - have same priority which ever occurs first from the left it will be evaluated.

Ex 2:-

```
  ①     ②
a * b / c
```

* and / both have same priority since
* came first , it is evaluated.

Ex 3:-

```
  ①     ③     ②
a / b + c % d
```

Ex 4:-
Write an equivalent expression for a^2 / b^2 ?

Wrong one :-

```
a * a / b * b
①     ②     ③
```

It becomes

```
  2
a / b  * b
```

Correct one :-

```
a * a / ( b * b )
②     ③     ①
```

Type conversions continued :-

What will be the output of this program?

```
#include <stdio.h>
void main()
{
  int a = 7 , b = 2;
  double c ;
  c = a / b * 3.5;
  printf(" %lf " , c );
}
```

Prg3.c

```
output :-
  10.5
```

Explanation :-

```
  c     =     a     /     b     *     3.5
(double)      (int)   (int)       (double)
                 |
              (int)
                3
                 |
              (double)
                3.0
                 |
              (double)
                10.5
```

*Note: Conversion is not made unless it is needed.

The same expression when written as

```
c = a * 3.5 / b ;
```

will result in 12.25

All these are implicit type conversions. When we manually ask the compiler to perform type conversion then it is referred as explicit type conversion.

Example :-

```
c = (double) a   / b * 3.5;
```

Example program to find the percentage of student after taking obtained marks and total marks as input :-

```
#include <stdio.h>

void main()
{
  int obt,tot;
  double per;

  printf("Enter obtained marks");
  scanf("%d",&obt);

  printf("Enter total marks");
  scanf("%d",&tot);

  per = (double)obt / tot * 100;

  printf("Percentage %lf", per);
}
```

Prg4.c

```
output :

  Enter obtained marks 190 ↵
  Enter total marks 200 ↵
  percentage 95.0
```

Tip:-
Try to write expression like this
(per = obt / tot * 100 ;)
and check output.

Post Increment and Decrement operator:

Increment and Decrement Operators :-

++ and -- operators are referred as increment and decrement operators. They can be used in two ways

var++

this type of usage is called post increment. Since ++ is used after the variable.

++var

this type of usage is called pre increment.

Similarly,

var--

referred as post decrement and

--var

is referred as pre decrement.

e.g.

```
int a = 10;
a++; /* a will be 11 after
        this statement */
```

e.g.

```
int a = 10;
--a; /* a will be 9 afte
        this statement */
```

Q: What is the difference between pre increment and post increment?

Ans:

Difference can be identified when this is used in combination with other expression.

e.g.

```
int a = 10;
int b;
b = a++;
```

In this case value of b is 10 and value of a is 11.

e.g.

```
int a = 10;
int b;
b = ++a;
```

In this case value of a is 11 and value of b is 11.

Reason :-

Post increment as the name implies increment a after substitution.

In b = a++; existing a value is substituted i.e. 10 and then a is incremented, then a becomes 11.

Where as in pre increment variable must be incremented before substitution.

In b = ++a; a is incremented before substitution and hence a is 11 and incremented value is substituted hence b is also 11.

Similarly pre and post decrement :-

e.g.

```
int a = 10;
int b ;
b = a--;
```

b is 10 and a is 9.

e.g.

```
int a = 10;
int b;
b = --a;
```

b is 9 and a is 9.

Example Program :-

Prg5.c

```
#include <stdio.h>

void main()
{
  int a = 10,b,c,d,e;

  b = a++; /* b is 10 a is 11 */
  c = ++a; /* a is 12 c is 12 */
  d = a--; /* d is 12 a is 11 */
  e = --a; /* a is 10 e is 10 */

  printf(" a is %d " , a );
  printf(" b is %d " , b );
  printf(" c is %d " , c );
  printf(" d is %d " , d );
  printf(" e is %d " , e );
}
```

output :-

```
a is 10
b is 10
c is 12
d is 12
e is 10
```

next relational operators.

Relational Operators :-

Verifies the given relation between two values and returns true (1) when the condition is satisfied and returns false (0) otherwise.

Various relational operators available are :-

```
== ( equal to )
!= ( not equal to )
>  ( greater than )
<  ( less than )
>= ( greater than or equal to )
<= ( less than or equal to )
```

Example Program :-

Prg6.c

```
#include <stdio.h>

void main()
{
  int a = 10,b=10,c,d,e,f,g,h;
  c = ( a == b ); /* 1 */
  d = ( a != b ); /* 0 */
  e = ( a < b );  /* 0 */
  f = ( a > b );  /* 0 */
  g = ( a <= b ); /* 1 */
  h = ( a >= b ); /* 1 */
  printf(" c is %d " , c );
  printf(" d is %d " , d );
  printf(" e is %d " , e );
  printf(" f is %d " , f );
  printf(" g is %d " , g );
  printf(" h is %d " , h );
}
```

Logical Operators :-

Three logical operators are available in C Language

```
&&  ( logical and )
||  ( logical or )
!   ( logical not )
```

&& and || are used to combine two relational expressions. Where as ! operator is an uniary operator.

Logical And (&&) :-

End result of the expression is true when both the expressions evaluate to true. If one is false then end result is false.

e.g.

```
( a < b ) && ( a < c )
```

****Note:
In C language we can substitute integral value for a boolean result. In such case non-zero value is considered as true and zero is considered as false.

e.g.

```
  10 && 20  => true  i.e. 1
 -10 && 20  => true  i.e. 1
   0 && 20  => false i.e. 0
```

In the first two cases both are non-zero values and hence treated as true.

Logical OR (||) :-

End result is true if any one of the expression is true , false if both are false.

e.g.

```
( a < b ) || ( a < c )
```

**Note:
As in previous case

```
10 || 0 => 1 ( true )
0  || 0 => 0 ( false )
```

Not operator (!) :-

If the given condition is true then ! makes it false , if it is false then ! makes it true.

e.g.

```
! ( a < b )
```

Example Program :-

Prg7.c

```
#include <stdio.h>

void main()
{
  int a=10,b=20,c=30,d,e,f;

  d = ( a < b ) && ( a < c ); /* 1 */
  e = ( a < b ) || ( a < c ); /* 1 */
  f = ! ( a < b );            /* 0 */

  printf(" d is %d ", d );
  printf(" e is %d ", e );
  printf(" f is %d ", f );
}
```

***Technical TIP :-

Logical And operator doesn't evaluate second expression if first evaluates to false.

E.g.

```
int a = 0 , b = 0 , c;
c = ( a++ > 0 ) && ( b++ > 0 );
```

What are the values of a,b,c ?

Ans:-

```
                 ---------------> a is 1
c = ( a++ > 0 ) && ( b++ > 0 )
      \_0_/
       \ 0 > 0 /
        \_____/
         false (i.e.0)
         \_____________________/
                false( i.e. 0)
```

a is 1
b is 0 (since it is not evaluated.)
c is 0 (since end result is false)

Try this :-

```
int a=10,b=20,c;
c = ( a++ > 0 ) || ( b++ > 0 );
```

What are the values of a,b,c?

***Technical Tip1 :-

What is the value of a in the following expression ?

```
a = 10 || 20 && 0 ;
```

Ans:- tip is to combine and operator. i.e.

```
a = 10 || [20 && 0];
```

It is 10 or (20 and zero). Since 10 is true end result is true i.e. a is 1.

Test:-
What are the values of a,b,c,d,e?

```
int a=0,b=0,c=0,d=0,e;
e = a++ && b++ || c++ && d++ ;
```

Ans:

```
a is 1
b is 0
c is 1
d is 0
e is 0.
```

next bitwise operators

Bitwise Operators :-

These operators operate on the individual bits of the given values. Available operators are

```
&   ( bitwise and )
|   ( or )
^   ( xor )
~   ( 1's compliment )
```

Example :-

```
#include <stdio.h>

void main()                                  Prg8.c
{
  int a=5,b=6,c,d,e,f;

  c = a & b ;
  d = a | b ;
  e = a ^ b ;
  f = ~a ;

  printf(" c is %d ", c );
  printf(" d is %d ", d );
  printf(" e is %d ", e );
  printf(" f is %d ", f );

}
```

Output :-

```
c is 4
d is 7
e is 3
f is -6
```

Explanation :-

5 in binary is 101 and 6 is 110

1) c = a & b

```
 101
 110
-----
 100   => 4
```

If both bits are 1 then result is 1 otherwise 0.

2) d = a | b ;

```
 101
 110
-----
 111   => 7
```

If any one of the bit is 1 then result is 1 otherwise 0

3) e = a ^ b ;

```
 101
 110
-----
 011   => 3
```

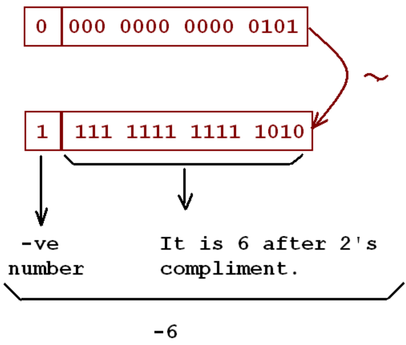
XOR:- Opposite bits implies 1 , equal bits implies 0.

4) f = ~a;

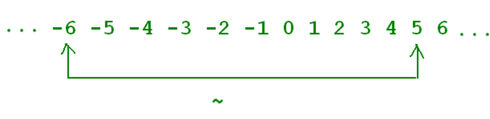
~ makes all 0s to 1s and all 1s to 0s.

```
 101
~
-----
 010
```

Value in a



Shortcut for compliment(~):-



next shift operators

Shift Operators :-

These operators will shift the bits in the memory location. There are two types of shift operators available

1) Left Shift(<<)
2) Right Shift (>>)

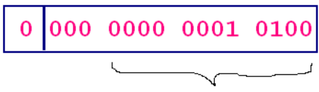
1) Left Shift (<<) :-

Shifts given n bits to the left. Left most n bits will be dropped.

E.g.

```
int a = 20;
```

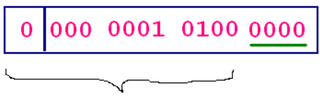
Memory image of a is



Now

```
int b = ( a << 4 );
```

Memory image of b is



Value of b is 320.

2) Right Shift (>>) :-

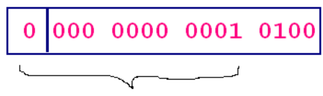
Operation of this operator is compiler dependent. It will shift given value n bits to the right. The newly formed left most bits are filled with zero in some compilers and with previous sign bit in some other compilers.

Standard is to fill them with previous sign bit value.

E.g.

```
int a = 20;
```

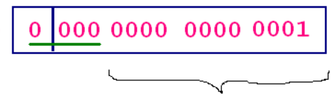
Memory image of a is



Now

```
int b = ( a >> 4 );
```

Memory image of b is



Example program :-

Prg9.c

```
#include <stdio.h>

void main()
{
  int a = 20 , b , c;

  b = a << 4 ;
  c = a >> 4 ;

  printf(" b is %d " , b );
  printf(" c is %d " , c );
}
```

```
Output:-
  b is 320
  c is 1
```

****Note:
Bitwise and shift operators are applicable only on integral types. i.e. char , int , short , long.

next assignment operators

Assignment Operators :-

Assignment operators are used to assign value to a variable. It is denoted by the symbol = .

Usage :-

```
Variable = Value ;
```

Example :-

```
int a;
a = 10;
```

***Note:
int a = 10;
Here this = operation is treated as initialization and not assignment.

Assignment operators have least priority. (last but one) Hence they are evaluated at the end of the expression.

```
a = b + c;
  ②   ①
```

Special Property of assignment operator :-

Like other operators they also produce result. i.e. the assigned value is returned as a result. Also when an expression has more than one assignment operator they are evaluated from right to left. This is technically referred as right to left associativity.

Example :-

```
int  a , b , c;

a  =  b  =  c  =  5;
            5
      5
      5
```

assigns 5 to c and returns 5 as the result. Then it is assigned to b and produces 5 as the result then it is assigned to a and produces 5 as the result. Last result is ignored by us.

Arithmetic Assignment :-

+= , -= , /= , *= etc are referred as arithmetic assignment operators. They perform respective operation on the given variable and stores the result.

Example :-

```
int a = 10;
a *= 2 ;
```

after which a will be 20.

Example 2:-

```
int a = 10;
a *= 2 + 2;
```

after which a is 40. Since + operator has more priority than *= operator.

Test:
int a = 10 ;
what is a /= 2 * 2 ?

Special Operators :-

There are many special operators such as [] , * , . , .* , -> etc. They will be discussed later.

Exercise :-

1) Write a program to find the average of the given three double values?

2) Write a program to find the biggest of three values using conditional operator ?

3) Write a program to find the simple interest for the given principal amount, rate of interest and number of years ?

4) Write a program to calculate the value of the expression

$x^3 + y^3 + xy$

for the given x and y values ?

next control structures

Control Structures :-

Control structures enable us to organize the execution path. Types of control structures available in C language are

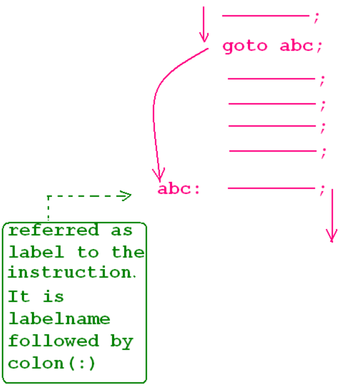
1) Branching
   a) Unconditional Branching.
   b) Conditional Branching.
2) Looping

1.a) Unconditional Branching:-

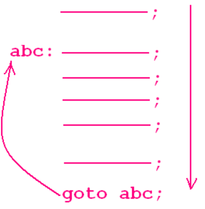
Allows us to take the control to the desired instruction with out verifying any condition.

goto statement in C language is used for this purpose.

Forward Branching :-



Backward Branching :-



***Note:
Backward branching must be carefully done with out which it may lead to infinite loop.

Example Program :-

Prg10.c

```
#include <stdio.h>

void main()
{
  printf(" hello ");

  goto sample;

  printf(" go to hell ");
sample:

  printf(" how r u ");
}
```

This statement will not be executed at all. Hence you may get unreachable code.

```
output :
   hello how r u
```

What is the output of this program ?

Prg11.c

```
#include <stdio.h>

void main()
{
  printf("line 1");
  goto a;
  printf("line 2");
a:printf("line 3");
  goto b;
  printf("line 4");
b:printf("line 5");
}
```

1.b) Conditional branching :-

In the above case with just goto we branched the control directly to the desired instruction. Now with conditional branching control path is selected based on some condition. There are two conditional branching statement available

1) if statement
2) switch statement

next if statement :

Before proceeding forward with if statement first we need to know what is a statement and types of statements.

In simple words a statement is an instruction that ends with a semicolon. There are three types of statements.

1) Simple Statement.
2) Compound Statement.
3) Null Statement.

Simple Statement :-

Consider it as an instruction that ends with a semicolon.

e.g

```
printf("hello");
```

Null Statement :-

A statement with a blank semi-colon is referred as null statement.

e.g.

```
;
```

Compound Statement or a block :-

It is a set of simple or compound statements which begins with { and ends with }.

e.g

```
{
    printf("hello");
    printf("learn c quickly");
}
```

1.b.1) If statement :-

If statement is used to execute a statement only when the given condition is true.

General Structure of if statement :-

```
if ( expression )
    {statement}
else
    statement2
```

If the expression evaluates to true i.e. a non-zero value value then Statement1 will be executed and if it evaluates to zero(false) then Statement2 will be executed.

Example Program :-

```
#include <stdio.h>
void main()
{
  int a,b;
  printf("Enter a");
  scanf("%d", &a );

  printf("Enter b");
  scanf("%d", &b );

  if( a > b )
  printf(" a is big ");
  else
  printf(" b is big ");
}
```

Pyg12.c

```
Output:
Enter a  10 ↵
Enter b  20 ↵
b is big

Enter a 20 ↵
Enter b 10 ↵
a is big
```

If can be further subdivided into three types

1) Simple if
2) if..else...
3) Nested if

Simple if:-

If there is only if part without else part then it is treated as simple if statement.

Usage:-

```
if( expression )
    Statement
```

In such case statement is executed only if the given expression is evaluated to true.

Example :-

```
if( b != 0 )
    printf("%d", a / b );
```

In this sample example division is performed only when denominator is not equal to zero.

if ... else ... :-

When if is combined with else part we can consider it as if..else statement.

```
  Refer to the previous
  example program.
```

Nested if :-

When if statement is combined with another if statement then it is referred as nested if statement.

Usage:-

```
if ( expression1 )
    Statement1
else
if ( expression2 )
    Statement2
else
    Statement3
```

If expression1 is true then Statement1 is executed, otherwise if expression2 is true then Statement2 is executed otherwise Statement3 is executed.

*Note: Only one of the three statements above can be executed.

Example Program :-

A program to find the biggest of three integers :-

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

void main()
{
  int a,b,c,big;

  clrscr(); /* clear screen */

  printf(" Enter a ");
  scanf("%d", &a);

  printf(" Enter b ");
  scanf("%d", &b );

  printf(" Enter c ");
  scanf("%d", &c );

  if( a > b && a > c )
    big = a;
  else
  if( b > c )
    big = b;
  else
    big = c;

  printf(" biggest is %d ", big);
}
```

Pyg13.c

```
output :-

Enter a 10 ↵
Enter b 20 ↵
Enter c 15 ↵
biggest is 20
```

Hint for the above program: If biggest is not a then biggest is either b or c. And hence comparing with a is not required.
i.e. b > a && b > c is not necessary.

Exercise :-

1) Write a program to find the biggest of four numbers ?
2) Write a program to find the bonus for the given salary.
If the salary is between 0 to 5000 then bonus is 10% of the salary. If it is between 5001 to 15000 then bonus is 20% of the salary. Otherwise bonus is 25% of the salary.

Technical Points :-

Q: What happens when b is 10 for this following code snippet.

```
if( b < 5 )
    printf(" hello ");
    printf(" sample ");
```

Ans:

Only one statement following if is considered under if. Hence only printf("sample") is executed since b < 5 is not true.
i.e.

```
if( b < 5 )
    printf(" hello ");
    printf(" sample ");
```

not under if hence executed irrespective of if condition.

Q: How to make more than one statement come under if statement ?

Ans:

Consider the previous example. incorder to make both statements come under if statement combine them into a compound statement.
i.e.

```
if( b < 5 )
{
  printf( hello ");
  printf( sample ");
}
```

Q: What is the output after the execution of this statement

```
if( 10 )
    printf("true");
else
    printf("false");
```

Ans:

Output is true. Since C language considers any non-zero value as true and zero is considered as false.
i.e. if(10) considered as true
if(0) considered as false.

Q: What is the output ?

```
int a = 5 , b = 6;
if( a > b )
    printf("equal");
else
    printf("not equal");
```

Ans:

Since the expression used in if statement is

```
a > b
```

i.e
assignment operator hence b values is assigned to a and the result is the assigned value. Since the assigned value is 6 and it is non-zero it is true. Hence equal.

```
if( a = b )
```

After the if statement value of a is 6. For comparison we need == .

Example Program :-

A program to find the gross and net salaries of an employee using the following table.

gross = basic + da + hra
net = gross - pf - it

Percentage of da,hra,pf,it based on basic

basic	0-1000	10001 or above
da	10%	12%
hra	12%	15%
pf	13%	18%
it	20%	25%

Ans:

Before writing any program answer the following questions.

What is the input ?
What is the process ?
What is the output ?

For the above program it sufficient to know basic salary and net can be calculated.

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

void main()
{
  double basic,da,hra,pf,it,gross,net;

  clrscr();

  printf(" enter basic ");
  scanf("%lf",&basic);

  if( basic >= 0 && basic <= 10000 )
  {
    da = basic * 10 / 100;
    hra = basic * 12 / 100;
    pf = basic * 13 / 100;
    it = basic * 20 / 100;
  }
  else
  {
    da = basic * 12 / 100;
    hra = basic * 15 / 100;
    pf = basic * 18 / 100;
    it = basic * 25 / 100;
  }

  gross = basic + da + hra;
  net = gross - pf - it ;

  printf("\n Basic %lf ", basic );
  printf("\n Da %lf ", da );
  printf("\n hra %lf ", hra );
  printf("\n pf %lf ", pf );
  printf("\n it %lf ", it );
  printf("\n gross %lf ", gross );
  printf("\n net salary %lf ", net );

  getch();
}
```

Pyg14.c

```
output :-

enter basic 12000 ↵
Basic 12000.00000
Da 1440.00000
hra 1800.00000
pf 2160.00000
it 3000.00000
gross 15240.00000
net salary 10080.00000
```

General Mistake beginners do :-

```
if( a > b )
    printf(" a is big ");
else
    printf(" b is big ");
```

note semi-colons here.

It will produce an error misplaced else.

Reason behind it :-

Let us divide the above if statement

```
if( a > b )
    printf(" a is big ");
else
    printf(" b is big ");
```

It becomes simple if. Note that even an empty semi-colon is also a statement Null statement). And if considers immediately following statement under if. Hence if ended with the given semi-colon the following statement is treated as where I follow it immediately.

Now else doesnot have the corresponding if.

1.b.2)Switch Statement :-

Allows us to define operations for each individual case.

Usage:

```
    switch( expression )
    {
    case value1:
            statements
            [break;]
    case value2:
            statements
            [break;]
    case value3:
            statements
            [break;]
    ...
    case valueN:
            statements
            [break;]
    default:
            statements
            [break;]
    }
```

If the expression evaluates to any one of the value then the corresponding statements with in that case will be executed. If no case is matched then the statements under default are executed.

Note:
- default is optional
- default need not be written at the end. i.e. it can be written in the middle also.

Example :-

Prg15.c

```
#include <stdio.h>

void main()
{
 int num;

 printf(" Enter any number ");
 scanf("%d",&num);

 switch( num )
 {
 case 1:
        printf("one");
        break;
 case 2:
        printf("two");
        break;
 case 3:
        printf("three");
        break;
 default:
        printf("not in 1 or 2 or 3");
        break;
 }
}
```

General Mistake beginners do :-
They write case1 without space here there is space between case and 1 i.e. case 1 and it is followed by colon(:) and not semi-colon.

```
output :

 Enter any number 2↵
 two

 Enter any number 4↵
 not in 1 or 2 or 3
```

Explanation :-
Execution Sequence when given number is 3.

```
(1) printf(" Enter any number ");
(2) scanf("%d",&num);←given 3

(3) switch( num )
    {
    case 1:
            printf("one");
            break;
    case 2:
            printf("two");
            break;
(4) case 3:
(5)         printf("three");
(6)         break;
    default:
            printf("not in 1 or 2 or 3");
            break;
    }
(7) ( break terminates switch )
```

Q: What happens when break is ignored for some cases ?
Ans:
With out break switch continues to execute following cases after the matching case until it finds break or end of switch.
Following example illustrates the execution sequence when given num is 2.

```
                                  num = 2
(1) switch( num )
    {
    case 1:
            printf("one");
(2) case 2:
(3)         printf("two");
                                  (no break)
    case 3:
(4)         printf("three");
(5)                               (break)
    default:
            printf("not in 1/2/3");
            break;
    }
(6)
```

Q: Are there any situations where break is ignored?
Ans:
Yes, if there is common logic for more than one case then we tend to ignore break since there is no need to repeat the code.
Consider the following example :-

Prg16.c

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

void main()
{
 int num;

 printf("Enter number");
 scanf("%d",&num);

 switch( num )
 {
 case 1:
 case 2:
 case 3:
        printf("one/two/three");
        break;
 case 4:
 case 5:
 case 6:
        printf("four/five/six");
        break;
 default:
        printf("none");
 }
}
```

Common Logic for more than one case

Note: no break for default since it is written at the end of the switch and switch will automatically end after it. Hence no need for break.

```
output:

 Enter number 2 ↵
 one/two/three

 Enter number 4 ↵
 four/five/six

 Enter number 9 ↵
 none
```

Example Program :-

A program to find the gross and net salaries of an employee based on the following data.
gross = basic + da + hra
net = gross - pf - it.

grade	1	2	3/4
da	30%	25%	22%
hra	25%	20%	18%
pf	15%	12%	10%
it	15%	12%	10%

These percentages are given in terms of basic.

Ans:

Here we need basic salary and grade as inputs and rest of the values can be calculated.

Prg17.c

```
#include <stdio.h>

void main()
{
 int grade;
 double basic,da,hra,pf,it;
 double gross,net;

 printf("Enter basic salary");
 scanf("%lf",&basic);

 printf("Enter grade ");
 scanf("%d",&grade);

 switch( grade )
 {
 case 1:
        da = basic * 0.3;
        hra = basic * 0.25;
        pf = basic * 0.15;
        it = basic * 0.15;
        break;
 case 2:
        da = basic * 0.25;
        hra = basic * 0.2;
        pf = basic * 0.12;
        it = basic * 0.12;
        break;
 case 3:
 case 4:
        da = basic * 0.22;
        hra = basic * 0.18;
        pf = basic * 0.10;
        it = basic * 0.10;
 }
 gross = basic + da + hra ;
 net = gross - pf - it;

 printf("grade: %d" , grade);
 printf("basic: %lf",basic);
 printf("da :%lf",da );
 printf("hra : %lf",hra );
 printf("pf : %lf",pf);
 printf("it : %lf",it);
 printf("gross: %lf",gross);
 printf("net :%lf",net );
}
```

Input

Process

Output

*There is a small programming bug(i.e.error) in this example but it is ignored here.

Test:
What is the output of this program ?

```
#include <stdio.h>

void main()
{
 int x = 2;
 switch( x )
 {
 case 1:  x+=;
 case 2:  x -=10;
 case 3:  x +=5;
 default: x +=9;
 case 4:  x +=10;
 }
 printf("%d",x);
}
```

Prg18.c

Exercise :-
1) Write a program to print the month name for the given month number using
   a) if statement
   b) switch statement.

next looping structures

2)Looping Structures or Iterative Structures:-

In graph theory an edge for which source and destination is same vertex such edge is referred as a loop.

Similarly in computer programming there are some statements that enable the programmer to repeat a statement multiple times. These are referred as looping or iterative statements. Three statements of such kind are available in C language. They are

1) while statement
2) do...while statement
3) for statement.

2.1) while statement :-

Repeats the statement as long as the expression evaluates to true i.e. non-zero value.

Syntax:-

```
while( expression )
  statement
```

Example :-

Example that prints "hello" message 10 times.

Prg19.c

```
#include <stdio.h>

void main()
{
   int n = 1;

   while( n <= 10 )
   {
     printf("hello \n");
     n = n + 1;
   }
}
```

```
output:
  hello
  hello
  ...etc for 10 times
```

Note: output in multiple lines due to \n

Execution Flow :-

2.2) do .. while statement :-

Same as while statement but the main difference is that the statement is executed atleast once since condition is verified at the end of the statement executed.

Syntax :-

```
do
  Statement
while( expression );
```

Note semi-colon here.

Example :-

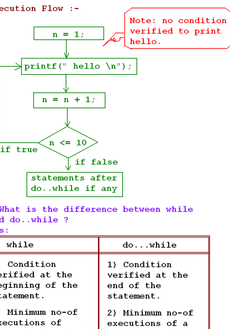
Prg20.c

```
#include <stdio.h>

void main()
{
   int n = 1;
   do
   {
     printf("hello \n");
     n = n + 1;
   }while( n <= 10 );
}
```

Tip:
What is the output when n is initialized to 11.
i.e.
int n=11;

```
output:
  hello
  hello
  ... 10 times
```

Execution Flow :-



Q:What is the difference between while and do..while ?

Ans:

while	do...while
1) Condition verified at the beginning of the statement. 2) Minimum no-of executions of while statement is zero or more. 3) Since condition verified initially it is called entry control statement.	1) Condition verified at the end of the statement. 2) Minimum no-of executions of a do..while statement is one or more. 3) Since condition is verified at the end it is called exit control statement.

2.3) for statement :-

for statement too is used to repeat the statement more than once. But in for initialization, conditional expression and increment/decrement all can be written with in for itself which makes it more readable than while statement.

Syntax :-

```
for( [exprl] ; [expr2] ; [expr3] )
      statement
```

expr1 : this expression is evaluated initially before expr2 is evaluated. This is normally used to initialize the loop counter.

expr2 : this represents conditional expression. As long as expr2 is true i.e. non-zero , execution of the statement is repeated.

expr3 : this expression is evaluated each time after the execution of the statement. This is normally used to increment or decrement the loop counter.

Syntax simplified :-

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

Example :-

Prg21.c

```
#include <stdio.h>

void main()
{
   int n;

   for( n = 1 ; n <= 10 ; n = n + 1 )
   {
     printf("hello \n");
   }
}
```

* Observe all three i.e. initialization, condition and increment all are inside for itself.

* Since there is only one statement under for i.e. print("hello \n"); there is no need for braces.
i.e.

```
for( n = 1 ; n <= 10 ; n = n + 1 )
   printf( "hello \n");
```

* In the above for loop
  ▶ n = 1 is executed once before proceeding with condition.
  ▶ n <= 10 is the condition which must be true(1) inorder to proceed with statement execution.
  ▶ n = n + 1 is executed after executing the statement.

* Flow is same as that of while loop.

Note: both for and while come under entry control statements and do..while under exit control statement.

Tip:-

All three expressions in for are optional. If expr2 is ignored it is treated as true
e.g.

no condition => true

```
for( ; ; )
  printf( " hello ");
```

prints hello infinite times. There are other statements such as break etc to terminate the loop.

Note:
Ignoring condition is valid only in for loop and not in while or do..while.
i.e.

```
while()
   printf("hello");
```

next examples for loops

A program to print numbers from 1 to n :-

Prg22.c

```
#include <stdio.h>

void main()
{
  int i,n;

  printf("Enter n");
  scanf("%d",&n );

  i = 1;
  while( i <= n )
  {
    printf(" %d " , i );
    i++;
  }
}
```

```
say n = 8
i = 1
1 <= 8 => true
  print i i.e 1
  i++ => i = 2
2 <= 8 => true
  print i i.e.2
  i++ => i = 3
3 <= 8 => true
  print i i.e.3
  i++ => i = 4
....
8 <= 8 => true
  print i i.e.8
  i++ => i = 9
9 <= 8 => false
  loop ends.
```

```
output  :
Enter n 8↵
 1 2 3 4 5 6 7 8
```

Exercise :-
1) Write the above program using for loop?
2) Write a program to print the numbers from n to 1 ?

A program to print the multiplication table for the given number :-

Prg23.c

```
#include <stdio.h>

void main()
{
  int i,n;

  printf("Enter n");
  scanf("%d",&n );

  i = 1;
  while( i <= 10 )
  {
   printf("\n%d x %d = %d",n,i,n*i);
   i++;
  }
}
```

```
output  :
  Enter n 8↵
  8 x 1 = 8
  8 x 2 = 16
  .....
  8 x 10 = 80
```

printf explained :-

```
                    value of i
                    ↓
printf("\n%d x %d = %d",n,i,n*i);
          ↑                 ↑
       value of n        value of n*i
```

Here n is fixed but i varies from 1 to 10.

A program to find the sum of n natural numbers :-

Prg24.c

```
#include <stdio.h>

void main()
{
  int i,n,sum;

  printf("Enter n");
  scanf("%d",&n );

  sum = 0;
  for( i = 1 ; i <= n ; i++ )
     sum = sum +  i;

  printf(" sum is %d " , sum );
}
```

```
output :
  Enter n 10↵
  sum is 55
```

i = 1	i = 2	
sum = sum + i i.e. 0 + 1 ------ 1	sum = sum + i i.e. 1 + 2 ------ 3	....

note this initial value.

Exercise :-
1) Simplify the above program with the formula that you know ?

A program to find the product of n natural numbers, also called factorial of n :-

Prg25.c

```
#include <stdio.h>

void main()
{
  int i,n,fact;

  printf("Enter n");
  scanf("%d",&n );

  fact = 1;
  for( i = 1 ; i <= n ; i++ )
      fact = fact * i;

  printf(" factorial is %d " , fact );
}
```

```
output :-
 Enter n 5 ↵
 factorial is 120
```

Explanation :-

Note the initial value what if it is 0?

i = 1	i = 2	
i <= 5 ✓ fact = fact * i i.e. 1 * 1 ------ 1	i <= 5 ✓ fact = fact * i i.e. 1 * 2 ------ 2	
i = 3 i <= 5 ✓ fact = fact * i i.e. 2 * 3 ------ 6	....	i = 6 i <= 5 ✗ false

---

A program to check whether the given number is prime or not :-

* Any prime number has only two factors i.e 1 and itself.
* A number is said to be a factor of n if it divides n. i.e. reminder is zero.
* In the following program count represents factors count. If it is equal to two then the given number is prime.

Prg26.c

```
#include <stdio.h>

void main()
{
  int n,i,count;

  printf("Enter n");
  scanf("%d",&n );

  /* now to check whether n is prime or not */

  count = 0;
  for( i=1; i <= n ; i++ )
  {
      if( n % i == 0 )
      {
          count++ ;
      }
  }

  if( count == 2 )
      printf(" it is prime ");
  else
      printf(" it is not prime ");
}
```

```
output :-
 Enter n 9 ↵
 it is not prime

 Enter n 7 ↵
 it is prime
```

Note: It can be written as
for(i=1; i<=n; i++)
if(n % i == 0)
count++;
entire if is just one statement.

Note: Other efficient logics also available they will be discussed later.

Explanation :-

Consider n = 4
count = 0 (initial value) ①

i = 1 ② i <= 4 ✓ 4 % i == 0 ✓ count ++ ; => count = 1	i = 2 ③ i <= 4 ✓ 4 % i == 0 ✓ count++ ; => count = 2
i = 3 ④ i <= 4 ✓ 4 % i == 0 ✗	i = 4 ⑤ i <= 4 ✓ 4 % i == 0 ✓ count++ ; => count = 3
i = 5 ⑥ i <= 4 ✗ loop terminates.	count != 2 ⑦ hence it is not prime

A program to find the sum of the digits of the given positive number :-

Prg27.c

```
#include <stdio.h>

void main()
{
  int n,digit,sum;

  printf("Enter n");
  scanf("%d",&n );

  sum = 0;
  while( n > 0 )
  {
    digit = n % 10;
    sum += digit;
    n = n / 10 ;  /* (or) n /= 10; */
  }
  printf("digits sum %d " , sum );
}
```

```
output :
Enter n 125 ↵
digits sum 8
```

Explanation :-

① n = 125 (given)
sum = 0 (initial value)

n > 0 => 125 > 0 ✓ digit = n % 10 => 125 % 10 => 5 sum = sum + digit => 0 + 5 => 5 ②	n > 0 => 12 > 0 ✓ digit = n % 10 => 12 % 10 => 2 sum = sum + digit => 5 + 2 => 7 ③
n = n / 10 => 125 / 10 => 12 (integer)	n = n / 10 => 12 / 10 => 1
n > 0 => 1 > 0 ✓ digit = n % 10 => 1 % 10 => 1 sum = sum + digit => 7 + 1 => 8 n = n / 10 => 1 / 10 => 0 ④	n > 0 => 0 > 0 ✗ loop terminates. Note: n becomes zero after the end of the loop. A point for next example. ⑤

A program to check whether the given number is armstrong number or not :-

An armstrong number is the one whose sum of the cubes of the individual digits of the given number is equal to the same number.
e.g.
153 => $1^3 + 5^3 + 3^3$
=> 153

Prg28.c

```
#include <stdio.h>

void main()
{
  int n,digit,sum,m;

  printf("Enter n");
  scanf("%d",&n );

  m = n;
  sum = 0;
  while( n > 0 )
  {
    digit = n % 10;
    sum += digit * digit * digit;
    n /= 10 ;
  }

  if( m == sum )
    printf(" it is armstrong ");
  else
    printf(" it is not armstrong");
}
```

n value is saved in m since n will be zero after the end of the loop.

sum is compared with m and not n.

```
output :
Enter n 153 ↵
it is armstrong

Enter n 12 ↵
it is not armstrong
```

A program to check whether the given number is perfect number or not :-

A number is said to be a perfect number if sum of the factors is equal to double the number.
e.g.
6 => 1 + 2 + 3 + 6
=> 12
=> i.e. 2 * 6

Prg29.c

```
#include <stdio.h>
void main()
{
  int n,sum,i;

  printf("Enter n");
  scanf("%d",&n );

  i = 1;
  sum = 0;

  while( i <= n )
  {
    /*if i is a factor of n then add to sum */
      if( n % i == 0 )
          sum += i;

      i++;
  }

  if( sum == 2 * n )
    printf(" it is perfect no ");
  else
    printf(" it is not perfect no");
}
```

```
output :
Enter n 6 ↵
it is perfect no

Enter n 7 ↵
it is not perfect no
```

Explanation :-

given n = 6 , sum = 0 (initial value)

i = 1	n % i == 0 ,	sum += i => sum = 1
i = 2	n % i == 0 ,	sum += i => sum = 3
i = 3	n % i == 0 ,	sum += i => sum = 6
i = 4	n % i != 0	
i = 5	n % i != 0	
i = 6	n % i == 0 ,	sum += i => sum = 12

Exercise :-
1) Write a program to find the sum of the digits of the given number upto single digit.
i.e. n = 987 => 9 + 8 + 7
=> 24
=> 2 + 4
=> 6

2) Write a program to print the following series
1 , 2 , 2 , 4 , 8 , 32 , 256 ...

A program to print the fibonacci series :-

The following series is referred as fibonacci series
0 , 1 , 1 , 2 , 3 , 5 , 8 , ...
i.e. next number is the sum of the previous two numbers.

Prg30.c

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

void main()
{
  int max , n1 , n2 , num ;

  clrscr();

  printf(" Enter maximum value ");
  scanf("%d" , &max );

  n1 = 0;
  n2 = 1;

  if( max >= 1 )
  {
    printf(" %d" , n1 );
    printf(" %d" , n2 );
  }

  while( n1 + n2 <= max )
  {
    num = n1 + n2 ;
    printf(" %d" , num );
    n1 = n2;
    n2 = num;
  }
}
```

```
output:
Enter maximmum value 5
0 1 1 2 3 5
```

Explanation :-

```
max =  5
n1 = 0
n2 = 1

if( max >= 1 ) ✓
   print n1 , n2 i.e. 0  1

n1 + n2 <= max  i.e. 0 + 1 <= 5 ✓

num = n1 + n2
    => 0 + 1
    => 1

print num i.e. 1

n1 = n2
   => 1

n2 = num
   => 1

n1 + n2 <= max i.e. 1 + 1 <= 5  ✓

num = n1 + n2
    => 1 + 1
    => 2

print num  i.e. 2

n1 = n2
   => 1

n2 = num
   => 2

n1 + n2 <= max i.e. 1 + 2 <= 5  ✓

num = n1 + n2
    => 1 + 2
    => 3

print num i.e. 3

n1 = n2
   => 2

n2 = num
   => 3

n1 + n2 <= max i.e. 2 + 3 <= 5  ✓

num = n1 + n2
    => 2 + 3
    => 5

print num  i.e. 5

n1 = n2
   => 3

n2 = num
   => 5

n1 + n2 <= max is false i.e. 3 + 5 <= 5
loop terminates.
```

In Simple Flow :-

```
0     1     1     2     3     5
n1    n2    num
            (n1+n2)

      n1    n2    num

            n1    n2    num

                  n1    n2    num
```

next nesting of loops

Nested Loops :-

When a looping statement is written inside another one then it is referred as nesting of loops.

Example program that prints multiplication tables for 1 to 10.

Prg31.c

```
#include <stdio.h>

void main()
{
  int i , j ;
  i = 1 ;
  while ( i <= 10 )
  {
    j = 1;
    while( j <= 10 )
    {
      printf("\n %d x %d = %d",i,j,i*j);
      j++;
    }
    printf("\n");
    i++;
  }
}
```

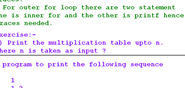
Q:How many times this printf will be executed?

```
output:-

1 x 1 = 1
1 x 2 = 2
......

2 x 1 = 2
2 x 2 = 4
.....
upto 10.
```

Explanation :-

i = 1 j = 1 to 10 print i*j	i = 3 j = 1 to 10 print i*j
i = 2 j = 1 to 10 print i*j	.....

Diagrammatically you can view the code as



Same code using for :-

```
for( i=1; i<=10; i++ )
{
  for( j=1; j <= 10 ; j++ )
    printf("\n %d x %d = %d",i,j,i*j);
  printf("\n");
}
```

* For the inner for loop there is only one statement i.e printf(....) hence no need for braces.

* For outer for loop there are two statement one is inner for and the other is printf hence braces needed.

Exercise:-
1) Print the multiplication table upto n. Where n is taken as input ?

A program to print the following sequence

```
    1
    1 2
    1 2 3
    1 2 3 4
... up to n.
```

Prg32.c

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

void main()
{
  int n,i,j;

  printf( " Enter n " );
  scanf( "%d" , &n );

  for( i = 1 ; i <= n ; i++ )
  {
    printf( "\n" );

    for( j = 1 ; j <= i ; j++ )
      printf( " %d" , j );
  }
}
```

output: as given

Explanation :-

i = 1 j = 1 to i => 1 to 1 print j
i = 2 j = 1 to i => 1 to 2 print j
i = 3 j = 1 to i => 1 to 3 print j
etc. upto i <= n

Exercise :-
1) Write a program to print the following sequence

```
  a) 1
     2 1
     3 2 1
     etc. upto n.

  b) 1
     2 2
     3 3 3
     4 4 4 4
     etc. upto n.
```

A program to print the following sequence

```
      *
     ***
    *****
   *******
```

When we study the above pattern we have to identify a mathematical formula for generating spaces and asterisks.

Say n = 4 => i.e. 4 lines then

	n-i spaces, 2*i-1 asterisks
i=1 bbb*	4-1 spaces, 2*1-1 asterisks
i=2 bb***	4-2 spaces, 2*2-1 asterisks
i=3 b*****	4-3 spaces, 2*3-1 asterisks
i=4 *******	4-4 spaces, 2*4-1 asterisks

b represents blank space. For simplification we have written just b for blank space.

Prg33.c

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

void main()
{
  int n,i,j,k;

  printf( " Enter n " );
  scanf( "%d" , &n );

  /* for n lines */

  for( i = 1 ; i <= n ; i++ )
  {
    /* a for loop for blank spaces
       and it is followed by a for
       loop for asterisks */

    for( j = 1 ; j <= n-i ; j++ )
      printf( " " ); /* space in printf */

    for( k = 1 ; k <= 2*i-1 ; k++ )
        printf("*");

    printf("\n"); /* new line */
  }
}
```

Exercise :-
1) Write a program to print the following sequence

```
      a)    *******
             *****
              ***
               *

      b)       *
              ***
             *****
            *******
             *****
              ***
               *
```

Hint:
for i = 1 to n

for i = n-1 to 1

If we write everything then what you will do, write them?

A program to find the result of the following series after reading the value of x.

$$1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots\ldots$$

Prg34.c

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

void main()
{
  int i,n,sign;
  double num,deno,x,result;

  printf("Enter no of terms ");
  scanf ("%d" , &n );

  printf("Enter value of x ");
  scanf ("%lf" , &x );

  result = 1;
  num = x;
  deno = 1;
  sign = -1;
  for( i = 2 ; i <= n ; i++ )
  {
    result += sign*(num / deno);
    num = num * x;
    deno = deno * i;
    sign = sign * -1;
  }

  printf(" result is %lf " , result );
}
```

```
output:
Enter no of terms 5 ↵
Enter value of x 1.25 ↵
result is 0.307454
```

Explanation :-

```
n = 5
x = 1.25
    ( read from keyboard )

num = x (consider it as x instead of 1.25)
deno = 1
sign = -1

result = 1
```

```
i = 2
i <= n ( true )

  result += sign * (num/deno)
            i.e. 1 + [-1 * x / 1]
            i.e. 1 - x
  result = 1-x

  num = num * x
    i.e. x * x => x^2

  deno = deno * i
    i.e. 1 * 2 => 2!

  sign = sign * -1
    i.e. -1 * -1 => +1
```

```
i = 3
i <= n ( true )

  result += sign * ( num / deno )
  i.e. (1-x) + [ 1 * x^2 / 2! ]
  i.e. 1 - x + x^2/2!

  result = 1 - x + x^2/2!

  num = num * x
    i.e. x^2 * x => x^3

  deno = deno * i
    i.e. 2! * 3 => 3!

  sign = sign * -1
    i.e. +1 * -1 => -1
```

```
 ....
```

Write a program to find the result of the following series

a) $1 + x + x^2 + x^3 + x^4 + \ldots$ n terms

b) $1 - 2x + 3x - 4x + 5x \ldots$ n terms

c) $1 + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \ldots\ldots$ n terms

next break and continue statement

break statement :-

Can be used only with in switch or while or do..while or for.

It is used to terminate the switch or the loop.

e.g.

```
.............
while( .... )
{
  ........
  if( someCondition )
    break;
  ........
}
..........
```

if executed it will terminate the loop.

Q: Where it is used ?

Ans:

Some times we have to stop the loop when the goal has been reached in the middle, even before the end of the loop.

Example :-

A program to check whether the given number is prime or not :-

Though we had it covered , this is a different logic.

Points for the following program:-

* For any number there won't be any factors after n/2, except itself.
For example, if n = 100 there is no need to compare from 51 to 99 since they are not factors.
* if any number from 2 to n/2 divides n then it can not be prime. Hence no need to proceed forward.
* Note: 1 is not a prime number.

Prg35.c

```
#include <stdio.h>
void main()
{
  int n,i,prime;

  printf(" Enter n ");
  scanf( "%d" , &n );

  prime = 1; /* say 1 implies yes */
  for( i = 2 ; i <= n/2 ; i++ )
  {
     if( n % i == 0 )
     {
        prime = 0; /* 0 implies no */
        break;
     }
  }

  if( n != 1 && prime == 1 )
     printf(" it is prime ");
  else
     printf(" it is not prime ");
}
```

Since 1 is not prime, n should not be 1 and variable prime should contain 1, for n to be prime.

****
TIP :- It is not even n/2, we can check upto /n and if one divides n then it is not prime.

continue statement :-

unlike break, continue statement is used to proceed forward with the next iteration without executing further part of the loop.

e.g.

```
.............
while( expression )
{
  .........

  if( someCondition )
    continue;

  .........
  .........
}
```

these statements will not be executed if continue is executed.

Example :-

Prg36.c

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

  i = 1;
  while( i <= 10 )
  {
     if( i == 3 || i == 5 )
     {
        i++;
        continue;
     }

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

if the condition is true.

output:

```
1 2 4 6 7 8 9 10
```

See there is no 3 and 5 since in that case we have executed continue statement.

Q: Where do we use continue ?

Ans:

In a case where we will find a statement like this is not what I want " then we issue continue statement.

Check this example :-

```
.........
while( end of the list )
{
  if( this is not the element )
  {
     next step;
     continue;
  }

  this is what we want.
  do some task.
}
.........
```

If not now you will understand this after some time.

A program to print all prime numbers between n and m, where n and m are read from the keyboard.

Core Logic :-

```
read n
read m
for i = n to m
    if ( i is prime )
        print i.
```

Program :-

Prg37.c

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

void main()
{
  int n, m,prime,i,j;

  printf("Enter n");
  scanf( "%d" , &n );

  printf("Enter m");
  scanf( "%d" , &m );

  for( i = n ; i <= m ; i++ )
  {
     prime = 1; /*say i is prime*/
     for( j = 2; j <= sqrt(i); j++ )
     {
        if( i % j == 0 )
        {
           prime = 0;
           break;
        }
     }
     if( prime == 1 )
        printf(" %d " , i );
  }
}
```

break ; breaks the loop in which it is written and here it is inner for loop which is terminated.

output:

```
Enter n 3 ↵
Enter m 10 ↵
3 5 7
```

note n and m should not be 1. For simplification we have ignored this case.

Explanation :-

given n=3 and m = 10 then for loop is for i = 3 to 10

say i = 9 then

```
prime = 1
                    i.e. sqrt(9)
for j = 2 to 3
   => j = 2 and 9 % j != 0
   => j = 3 and 9 % j == 0
        hence prime = 0

if( prime == 1 ) is false
   hence i is not printed.
```

Exercise :-

1) Write a program to print all armstrong numbers between n and m

2) Write a program to print all strong numbers between n and m

3) Write a program to print all perfect numbers between n and m.

A program to print all the binomial co-efficients for the given n :-

Prg38.c

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

void main()
{
  int n,r,f1,f2,f3,ncr,i;

  clrscr();
  printf("Enter n");
  scanf("%d",&n );

  for( r = 0 ; r <= n ; r++ )
  {
     /* find n factorial in f1 */
     f1 = 1;
     for( i = 1; i <= n ; i++ )
        f1 *= i;

     /* find r factorial in f2 */
     f2 = 1;
     for( i = 1; i <= r ; i++ )
        f2 *= i;

     /* find n-r fact in f3 */
     f3 = 1;
     for( i = 1; i <= (n-r); i++ )
        f3 *= i;

     ncr = f1 / ( f2 * f3 );
     printf(" %d ", ncr );
  }
}
```

output :-

```
Enter n 5 ↵
1 5 10 10 5 1
```

Note:-
Because of the limit of an integer we can find factorial of small numbers only. Change it to double to extend it to some extent.

Exercise :-

1) Write a program to print the pascal triangle

```
      1
    1 2 1
  1 3 3 1
1 4 6 4 1
  ..... upto n.
```

next single dimensional arrays

Single Dimensional Arrays :-

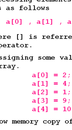
An array is a sequential collection of memory locations of similar datatype. All elements are identified using a single name i.e. identifier. Each element of an array is identified using its index. In C language index of an array begins with 0.

Example :-

Creating an array of 5 integers.

```
int a[5];
```

Memory representation of a is



Accessing elements of the above array is as follows

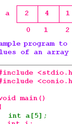
```
a[0] , a[1] , a[2] , a[3] , a[4]
```

Here [] is referred as array subscript operator.

Assigning some values to elements of an array.

```
a[0] = 2;
a[1] = 4;
a[2] = 1;
a[3] = 9;
a[4] = 10;
```

Now memory copy of a is



Example program to read and print the values of an array :-

Prg39.c

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

void main()
{
  int a[5];
  int i;

  printf("Enter values into an array");
  for( i = 0 ; i < 5 ; i++ )
    scanf( "%d" , &a[i] );

  printf("Values in the array ");
  for( i = 0 ; i < 5 ; i++ )
    printf( " %d", a[i] );
}
```

output :-

```
Enter values into an array 10 ↵
20 ↵
30 ↵
40 ↵
50 ↵
Values in the array 10 20 30 40 50
```

Exercise :-

1) Write a program to print the array of elements in reverse.

Main advantages of array :-

- All elements are identified using a single name.
- We can design the logic for a[i] and by just changing i value we can apply the logic to the required elements of an array.

```
i.e.  for i = 2 to 5
                logic on a[i]
i.e.  we applied the logic for
                a[2],a[3],a[4],a[5]
```

A program to find the biggest in an array of 5 double values.

```
#include <stdio.h>

void main()
{
  double arr[5],big;
  int i;

  printf("Enter values into an array");
  for( i = 0 ; i < 5 ; i++ )
    scanf( "%lf" , &arr[i] );

  big = arr[0];
  for( i = 1 ; i < 5 ; i++ )
    if( arr[i] > big )
      big = arr[i];

  printf(" biggest is %lf " , big );
}
```

```
output:
  Enter values into an array 10.2 ↵
  20.4 ↵
  12 ↵
  13 ↵
  40.5 ↵
  biggest is 40.5
```

Explanation :-

1) After declaration :-

2) After read operation :-

3) big = arr[0];

big 10.2 (double)

4) i = 1;
if(arr[i] > big) => arr[1] > big =>true
big = arr[i]; => big = 20.4;

big 20.4 (double)

5) i = 2;
if(arr[i] > big) => false

6) i = 3;
if(arr[i] > big) => false

7) i = 4;
if(arr[i] > big) => arr[4] > big =>true
big = arr[i]; => big = 40.5;

big 40.5 (double)

8) i = 5 and 5 < 5 is false, for loop ends.

9) Print big
=> 40.5

A program to check whether an element exists in an array or not :-

In the following program we are declaring an array of 10 integers.Instead of reading all 10 integer values we read n integer values, where n is given by the user. i.e. if n = 5 we read 5 integers i.e. 0 to 4

Prg41.c

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

void main()
{
  int a[10],n,i,ele;
  int found;

  printf("Enter n");
  scanf("%d" , &n );

  printf("Enter elements");
  for( i=0; i < n ; i++ )
    scanf("%d", &a[i] );

  printf("Enter element to search");
  scanf( "%d" , &ele );

  found = 0;
  for( i=0; i < n ; i++ )
  {
    if( a[i] == ele )
    {
      found = 1;
      break;
    }
  }

  if( found == 0 )
    printf(" not found ");
  else
    printf(" found ");
}
```

```
output :-
  Enter n 4 ↵
  Enter elements 10 ↵
  20 ↵
  30 ↵
  40 ↵
  Enter element to search 30
  found
```

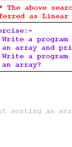
Explanation :-

1) After declaration.

2) read n <= given 4

n 4

3) read 0 to n-1 values into the array.
i.e. for i = 0 to n-1 read a[i]
i.e. for i = 0 to 3



4) read element to search <= given 30.

ele 30

5) found = 0;

found 0

6) for i = 0 to n-1
{ if(a[i] == ele)
{
found = 1;
break;
}
}

since a[2] == ele ,
found = 1
break the loop.

found 1

7) if(found == 0) is false
...
else
printf(" found ");

*** The above search process is technically referred as Linear search(sequential search).

Exercise:-

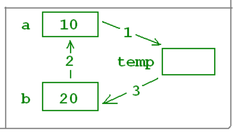
1) Write a program to find the smallest number in an array and print its position?
2) Write a program to find the second smallest in an array?

next sorting an array

Logic Swapping two elements:-

Consider that we need to swap a and b.
e.g.

```
int a=10,b=20,temp;

temp = a;  -->(1)
a = b;     -->(2)
b = temp;  -->(3)
```



Try these on your own :-

Tip:-
With out using temporary variables ?

```
Procedure1 :
      a = a + b;
      b = a - b;
      a = a - b;
```

```
Procedure 2:
  a=(a+b)-(b=a);
```

```
Procedure 3 :
      a = a ^ b;
      b = a ^ b;
      a = a ^ b;
```

Reminder:
^ is xor operation.

A program to sort an array of elements using bubble sort technique :-

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

void main()
{
  int arr[10],n,i,j;
  int temp;

  printf(" Enter n ");
  scanf("%d",&n );

  printf(" Enter %d elements " , n );
  for( i=0 ; i < n ; i++ )
     scanf("%d" , &arr[i] );

  /* bubble sort logic */
  for( i = 1; i < n ; i++ )
  {
    for( j = 0 ; j < n-i ; j++ )
    {
       if( arr[j] > arr[j+1] )
       {
         temp = arr[j];
         arr[j] = arr[j+1];
         arr[j+1] = temp;
       }
    }
  }
  printf(" Sorted list \n ");
  for( i = 0; i < n ; i++ )
    printf(" %d",arr[i] );
}
```

```
output :
  Enter n 5 ↵
  Enter 5 elements 10↵
  2 ↵
  3 ↵
  15 ↵
  4 ↵
  sorted list
  2 3 4 10 15
```

Explanation :-

Given array

10	2	3	15	4

Basic Logic :-
In each iteration compare all adjacent elements and take the biggest element to the right.

10	2	3	15	4

i.e.

10	2	3	15	4

since 10 is big swap them

2	10	3	15	4

since 10 is big swap them

2	3	10	15	4

since 15 is big swap not needed.

2	3	10	15	4

since 15 is big swap them

2	3	10	4	15

This is the result of the 1st iteration. After 1st iteration biggest element comes to the last position.

After second iteration array will be:-

2	3	4	10	15

It is sorted by now , but we will proceed with sort logic and after 3rd,4th iterations array will be in sorted order.

Exercise :-
1) Sort an array in descending order.
2) Sort all alternate elements of an array.
   i.e. 0,2,4,6 etc.

next characters

Characters :-

A character is a symbol identified by computer.
e.g. a-z,A-Z,0-9,/,* etc.
Each character is uniquely identified using ASCII code provided by American Standards Association.
ASCII American Standard Code for Information Interchange.

Displaying Character Internals :-

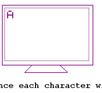
Character that is displayed will have a binary matrix in memory which represents its design.
For illustration consider a 4x4 matrix with binary values(i.e. 0 or 1).

0	1	1	0
1	0	0	1
1	1	1	1
1	0	0	1

Consider that 1 represents a dot and 0 represents an empty space.
Now divide the region that displays a character into a 4x4 matrix and place dots according to the above matrix then you will find

This is how you will see a character in the monitor.

Note: This is for character mode display only.



Since each character will have a binary matrix and it is impractical to store matrix each time when we want to store a character. Hence we are not going to store entire matrix each time in reality it is the ASCII code that represents the character is stored in the memory block.
i.e.

```
                    ---1---
    char ch ;       ch
                    ---1---
    ch = 'A';       ch  65
                            ASCII code for 'A'

    printf(" %d " , ch );
                 you will find 65 printed.
                 since we used %d.

    printf(" %c " , ch );
                 Here we are asking the character(%c) to
                 be printed corresponding to ASCII code 65.
```

Example program that calculate the percentage and grade based on the following chart

```
percentage 60 - 100 => grade = 'A'
percentage 50 - 59  => grade = 'B'
percentage 35 - 49  => grade = 'C'
percentage 0  - 34  => grade = 'D'
```

Prg43.c

```
#include <stdio.h>

void main()
{
  double obt,tot,per;
  char grade;

  printf("Enter obtained marks");
  scanf( "%lf" , &obt );

  printf("Enter total marks ");
  scanf( "%lf" , &tot );

  per = obt/tot * 100;

  if( per >= 60 )
    grade = 'A' ;
  else
  if( per >= 50 )
    grade = 'B';
  else
  if( per >= 35 )
    grade = 'C';
  else
    grade = 'D';

  printf(" Your percentage %lf ", per );
  printf(" Your grade %c ", grade );
}
```

```
Output :
 Enter obtained marks 60↵
 Enter total marks 100↵
 Your percentage 60.0
 Your grade A
```

Tip :-
Though we can directly assign 65 for a character
i.e. char ch = 65 ;
i.e. ASCII code of 'A'
But it is always good to assign character, since it is more readable.
i.e. char ch = 'A';

Character manipulation functions :-

1) Reading a single character

```
i) char ch;
   printf("Enter some character ");
   ch = getchar();
```

Enter some character A↵ → ch 'A'
press enter key after typing the character.

```
ii) char  ch;
    printf("Enter some character ");
    ch = getche();
```

Enter some character A → ch 'A'
Notice that there is no need to press enter key since getche() returns immediately after pressing some character.

```
iii) char ch;
     printf("Enter some character ");
     ch = getch();
```

Enter some character
You might have typed A but see here it is not displayed. getch doesn't echo the typed character.But ch will get the value.
ch 'A'

For getchar() we have to include stdio.h and for getch() and getche() include conio.h

2) Displaying character :-

```
char ch = 'A';

printf("%c",ch);
   (or)
putchar( ch );
   (or)
putch( ch );
```

In both the cases there is no need for %c since putchar and putch are defined for character only.
Where as printf is defined for all primitive types hence we need to mention what type of value we want to print.

3) Other character related operations :-

i) isalpha(char ch) :-
returns true if ch is in a-z or A-Z otherwise false.

ii) isdigit(char ch) :-
return true if ch is in 0-9 otherwise false.

iii) toupper(char ch) :-
returns upper case equivalent of ch.
i.e. char ch = toupper('a');
=> char ch = 'A';

iv) tolower(char ch) :-
returns lower case equivalent of ch.
i.e. char ch = tolower('A');
=> char ch = 'a';

Note: for all these methods we need to include ctype.h.

Example program that demonstrates character operations :-

Prg44.c

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

void main()
{
  char ch;

  printf("Enter some character");
  ch = getche();

  printf("Entered character ");
  putch( ch );
  if( isalpha( ch ) )
    printf("\n it is in alphabet ");
  else
    printf("\n it is not in alphabet");

  if( isdigit( ch ) )
    printf("\n it is a digit ");
  else
    printf("\n it is not a digit ");

  printf("\n in upper case %c ",toupper(ch));
  printf("\n in lower case %c ",tolower(ch));
}
```

try getch() in place of getche() and check the output.

```
output :-
 Enter some character a
 Entered character a
 it is in alphabet
 it is not a digit
 in upper case A
 in lower case a
```

next strings

Strings :-

A string is an array of characters followed by '\0' which is referred as null terminating character.

e.g.
```
char s[5]="abcd";
```

ASCII code of 'a' (small a).

Q: What is the need for '\0' i.e. 0 at the end of the string ?
Ans
You may consider an array which is larger than the string actual size. How to indicate that string ends here we need '\0' i.e.0.

e.g.
```
char s[10]="abc";
```

In reality memory copy of s is

memory copy of s is

These blocks will contain garbage values. i.e. unpredictable values.

represents end of the string.

Q: How to read and print the string ?
Ans
Use the code to represent string type. Note: In C language there is no separate string datatype, it is an array of characters only.

Prg45.c
```
#include <stdio.h>

void main()
{
  char name[40];

  printf("Enter your name:");
  scanf( "%s", name );

  printf("You have entered %s ", name );
}
```

Note there is no & i.e. &name

output:
```
Enter your name : xyz ↵
You have entered xyz

Enter your name : I am an Indian ↵
You have entered I
```

by default scanf considers enter key and space as separator keys

```
i.e. int a,b;
     scanf("%d", &a);
     scanf("%d",&b);
```

For the above two scanfs we can supply input in two ways

```
1) 10 ↵
   20 ↵

2) 10 20 ↵

case (2) illustrates that space too separates the inputs.
```

Q: Then what about "am an Indian" ?
Ans:
If there is any other scanf that follows the first one then it will read it.

Q: How to read names with spaces ?
Ans
We can use gets function or there is another option in scanf to consider spaces...

Prg46.c
```
#include <stdio.h>

void main()
{
  char name[40];

  printf("Enter your name : ");
  gets( name );

  printf("You have entered %s ", name );
}
```

output :
```
Enter your name : I am an Indian↵
You have entered I am an Indian
```

Tip:-
Reading spaces using scanf :-
```
  scanf("%[^\n]s",name);
```
^ :- all characters
\n :- separator i.e. new line character separates input.

refer to Prg47.c

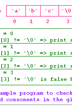
Example program to print the string by directly manipulating the character array :-

Prg48.c
```
#include <stdio.h>

void main()
{
  char s[100];
  int i;

  printf("Enter some text: ");
  gets( s );

  printf("You have entered ");

  i = 0;
  while( s[i] != '\0' )
  {
    putch( s[i] );
    i++;
  }
}
```

Output:
```
Enter some text : Just for test↵
You have entered Just for test
```

No can directly use printf("%s", s) but many times we have to process individual characters of the array

Explanation :-
Consider that we have entered abc then



i = 0
s[0] != '\0' => print s[0] i.e. a
i = 1
s[1] != '\0' => print s[1] i.e. b
i = 2
s[2] != '\0' => print s[2] i.e. c
i = 3
s[3] != '\0' is false hence loop terminates

Example program to count the number of vowels and consonants in the given string :-

Prg49.c
```
#include stdio.h>

void main()
{
  char s[100];
  int i;
  int vcount=0,ccount=0;

  printf("Enter some text: ");
  gets( s );

  for( i = 0 ; s[i] != '\0' ; i++ )
  {
    if (isalpha( s[i] ) )
    {
      switch( s[i] )
      {
        case 'a':
        case 'A':
        case 'e':
        case 'E':
        case 'i':
        case 'I':
        case 'o':
        case 'O':
        case 'u':
        case 'U':
                  vcount++;
                  break;
        default:
                  ccount++;
      }
    }
  }

  printf("No of vowels %d ", vcount );
  printf("No of consonants %d", ccount );
}
```

output:
```
Enter some text : this is sample
No of vowels 4 No of consonants 8
```

A program to check whether the given string is palindrome or not :-

A string is said to be palindrome if the reverse of the string is equal to the same string.
e.g. liril
malayalam
madam etc.

Logic :-

length of the string is 9 characters, Note: string length excludes '\0'.

len=1

=> 0 and len-1 must match
=> 1 and len-2 must match
=> 2 and len-3 must match

=> i and len-i-1 must match.

Prg50.c
```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
  char s[100];
  int i,len;

  printf("Enter any string ");
  gets( s);

  len = strlen( s);

  /* consider that it is palindrome */

  palin = 1;

  for( i = 0 ; i < len/2 ; i++ )
  {
    if( s[i] != s[len-i-1] )
    {
      /* one doesn't match implies
         it is not palindrome  */

      palin = 0;
      break;
    }
  }

  if palin == 1 )
    printf(" it is palindrome ");
  else
    printf(" it is not palindrome ");
}
```

returns the length of the string. see next topic.

output :-
```
Enter any string malayalam↵
it is palindrome
Enter any string malayam ↵
it is not palindrome
```

String manipulation function :-

strlen(str) :-
returns the length of the given string.
e.g.
```
int n = strlen("abc");
```
=> n will be assigned value 3.

strcpy(dest , src) :-
copies the content of source string into the destination string.
e.g.
```
  char s1[50]="abcd";
  char s2[5];
  strcpy( s2 , s1 );
```
=> copies the content of s1 to s2.

strcat(dest , src) :-
concatenates the content of src to dest.
e.g.
```
  char s1[10]="abc";
  char s2[10]="xyz";
  strcat( s1 , s2 );
```
=> After the above function s1 is "abcxyz".

strrev(str) :-
reverses the given str.
e.g.
```
  char s[10]="abcd";
  strrev( s );
```
=> after the function value of s is "dcba".

strupr(str) :-
converts the entire str to uppercase.
e.g.
```
  char s[10]="abc";
  strupr( s );
```
=> after the function s is "ABC"

strlwr(str) :-
converts the entire str to lowercase.
e.g.
```
  char s[10]="ABC";
  strlwr( s );
```
=> after the function s is "abc"

strcmp(str1 , str2) :-
compares the given two strings str1 and str2 and returns an integer value representing the comparison result.
if the return value is
```
  -ve then str1 < str2
  +ve then str1 > str2
  0   then str1 = str2
```
e.g.
```
  char str1[8]="abc";
  char str2[8]="abd";
  int n = strcmp( str1 , str2 );
  if( n > 0 )
    printf(" str1 > str2 ");
  else
  if( n < 0 )
    printf(" str1 < str2 ");// res.
  else
    printf(" str1 = str2 ");
```

'a' < 'b'
since
97 < 100
(ASCII codes)

strcmpi(str1 , str2) :-
compares the given two strings str1 and str2 and returns an integer value representing the comparison result.
if the return value is
```
  -ve then str1 < str2
  +ve then str1 > str2
  0   then str1 = str2
```
e.g.
```
  char str1[8]="abc";
  char str2[8]="ABC";
  int n = strcmpi( str1 , str2 );
  if( n > 0 )
    printf(" str1 > str2 ");
  else
  if( n < 0 )
    printf(" str1 < str2 ");// res.
  else
    printf(" str1 = str2 ");
```

'a' < 'A'
since
'a' < 'A'
i.e. 65 < 97

'a' < 'A' in this case there is no corresponding character to compare with 'a' so such case consider it is '\0'
i.e. 0. 0 < 'a'

strcmp(str1 , str2) :-
compares the given two strings str1 and str2 and returns an integer value representing the comparison result.
if the return value is
```
  -ve then str1 < str2
  +ve then str1 > str2
  0   then str1 = str2
```
e.g.
```
  char str1[8]="abc";
  char str2[8]="abc";
  int n = strcmp( str1 , str2 );
  if( n > 0 )
    printf(" str1 > str2 ");
  else
  if( n < 0 )
    printf(" str1 < str2 ");// res.
  else
    printf(" str1 = str2 ");
```

strcmpi(str1 , str2) :-
compares the given two strings str1 and str2 and returns an integer value representing the comparison result.
if the return value is
```
  -ve then str1 < str2
  +ve then str1 > str2
  0   then str1 = str2
```
e.g.
```
  char str1[8]="abc";
  char str2[8]="ABC";
  int n = strcmpi( str1 , str2 );
  if( n > 0 )
    printf(" str1 > str2 ");
  else
  if( n < 0 )
    printf(" str1 < str2 ");// res.
  else
    printf(" str1 = str2 ");
```

strcpy(s2 , s1) /* s1 : abc */
strrev(s2) /* s2 : cba */

if(strcmp(s1 , s2) == 0)
  printf(" it is palindrome ");
else
  printf(" it is not palindrome ");

Exercise :-
1) Write a program to find no of words in the given string.
2) Write a program to replace all vowels with character '*'
3) Write a program to reverse words in the given string.
e.g. "This is sample"
output -->"sihT si ylmaes"

2-D or double dimensional arrays.

Double Dimensional Arrays :-

A double dimensional array can be divided into rows and columns.

e.g.

```
int a[3][4];
```

General view of a in memory.

		0	1	2	3
	0	(0,0)	(0,1)	(0,2)	(0,3)
a	1	(1,0)	(1,1)	(1,2)	(1,3)
	2	(2,0)	(2,1)	(2,2)	(2,3)

To access each element in a matrix we need to specify row index as well as its column index.

Example :-

Prg52.c

```
#include <stdio.h>

void main()
{
   int a[3][2] = {
                    { 10,20 },
                    { 30,40 },
                    { 50,60 }
                  };

   int i,j;

   /*
      i to change row from 0 to 2
      j to change col from 0 to 1.
   */

   for( i = 0 ; i < 3 ; i++ )
   {
      for( j = 0 ; j < 2 ; j++ )
         printf(" %d ", a[i][j] );

      /*
        send new line character after each row
        to print next row in a separate line
      */

      printf("\n");

   }
}
```

```
      j ---->
      0  1
  i 0
    1
    2
```

```
output :
  10 20  (note after every row we have sent \n)
  30 40
  50 60
```

A program to add two nxm matrices :-

Logic:-

```
      0 1 2 ... m-1
    0 [10 ....      ]
    1 [             ]
    2 [             ]
A =   :
      :
  n-1 [             ]

      0 1 2 ... m-1
    0 [20 ....      ]
    1 [             ]
    2 [             ]
B =   :
      :
  n-1 [             ]

      0 1 2 ... m-1
    0 [30 ....      ]
    1 [             ]
    2 [             ]
C =   :
      :
  n-1 [             ]
```

i.e.
c[0][0] = a[0][0] + b[0][0];
c[0][1] = a[0][1] + b[0][1];
....
c[i][j] = a[i][j] + b[i][j];

Prg53.c

```
#include <stdio.h>
#include <cconio.h>
void main()
{
  int a[10][10],b[10][10],c[10][10];
  int n,m;
  int i,j;

  printf("Enter n");
  scanf("%d",&n);

  printf("Enter m");
  scanf("%d",&m);

  printf("Enter first matrix ");
  for( i = 0; i < n ; i++ )
    for( j = 0; j < m ; j++ )
      scanf("%d", &a[i][j] );

  printf("Enter second matrix ");
  for( i = 0; i < n ; i++ )
    for( j = 0; j < m ; j++ )
      scanf("%d", &b[i][j] );

  /* addition logic */
  for( i = 0; i < n ; i++ )
    for( j = 0; j < m ; j++ )
      c[i][j]=a[i][j]+b[i][j];

  printf(" added matrix \n ");
  for( i = 0; i < n ; i++ )
  {
     printf("\n");
     for( j = 0; j < m ; j++ )
       printf(" %d" , c[i][j] );
  }
}
```

```
output :-
 Enter n 3↵
 Enter m 2↵
 Enter first matrix 10↵
 20↵
 30↵
 40↵
 50↵
 60↵
 Enter second matrix 10↵
 20↵
 30↵
 40↵
 50↵
 60↵
 added matrix
  20  40
  60  80
 100  120
```

A program to check whether the given nxn matrix is an identity matrix or not.

e.g.

```
     0  1  2
  0 [1  0  0] left diagonal elements are
  1 [0  1  0] (0,0),(1,1),(2,2) all these should
  2 [0  0  1] be 1 and rest should be 0.
```

Prg54.c

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

void main()
{
  int a[10][10];
  int n,identity;
  int i,j;

  printf("Enter n");
  scanf("%d",&n);

  printf("Enter values ");
  for( i = 0; i < n ; i++ )
    for( j = 0; j < n ; j++ )
      scanf("%d", &a[i][j] );

  /* say given matrix is identity matrix */

  identity = 1;

  for( i = 0; i < n ; i++ )
  {
    for( j = 0; j < n; j++ )
    {
       /*
         if it is left diagonal
         element and not 1
       */

       if( ( i == j ) && ( a[i][j] != 1 ) )
       {
          identity  = 0;
          break;
       }

       /*
         other than left diagonal element
         and not 0.
       */

       if( ( i != j ) && ( a[i][j] != 0 ) )
       {
          identity = 0;
          break;
       }

    }

    /*
      if it is not identity matrix then no
      need to proceed with further rows.
    */

    if( identity == 0 )
       break;
  }

  if( identity == 1 )
     printf(" it is identity matrix ");
  else
     printf(" it is not identity matrix ");
}
```

```
output :-
  Enter n 3↵
  Enter values 1↵
  0↵
  0↵
  0↵
  1↵
  0↵
  0↵
  0↵
  1↵
  it is identity matrix
```

Exercise :-

1) Write a program to print the transpose of the given matrix.

2) Write a program to calculate the inverse of the given 2x2 matrix.

3) Write a program to sort each row of the given matrix.

next matrix multiplication

Matrix Multiplication :-

$$\underset{n \times q}{C} = \underset{n \times m}{A} * \underset{p \times q}{B}$$

Note: multiplication is possible only when m = p

Consider the following matrices

A	*	B
2x3		3x2

A =

	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)

B =

	0	1
0	(0,0)	(0,1)
1	(1,0)	(1,1)
2	(2,0)	(2,1)

Resultant matrix is a 2x2 matrix. Identify the relation while calculating the value.

Note: We are using (*,*) for simplicity.

```
c(0,0) = a(0,0) * b(0,0) +
         a(0,1) * b(1,0) +
         a(0,2) * b(2,0)

c(0,1) = a(0,0) * b(0,1) +
  i,j    a(0,1) * b(1,1) +
         a(0,2) * b(2,1)
           i        j

..
c(i,j) = a(i,0) * b(0,j) +
         a(i,1) * b(1,j) +
         a(i,2) * b(2,j)
```

It is the no of cols of first matrix (or) no of rows in the second.

$$= \sum_{k=0}^{2} a(i,k) * b(k,j)$$

Hence to multiply an nxm and pxq matrices each term in c[i][j] can be calculated as

```
for i = 0 to n-1
  for j = 0 to q-1
      c[i][j] = Σ(k=0 to m(or)p) a[i][k] * b[k][j]
```

since c will be an nxq matrix

Prg55.c

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

void main()
{
  int a[10][10],b[10][10],c[10][10];
  int n,m,p,q;
  int i,j,k;
  int product;

  printf("Enter n");
  scanf("%d",&n);

  printf("Enter m");
  scanf("%d",&m);

  printf("Enter p");
  scanf("%d",&p );

  printf("Enter q");
  scanf("%d",&q );

  if( m != p )
  {
     printf("multiplication not possible");

     /* exit terminates the
        program execution */

     exit(1);
  }

  printf("Enter first matrix ");
  for( i = 0; i < n ; i++ )
    for( j = 0; j < m ; j++ )
       scanf("%d", &a[i][j] );

  printf("Enter second matrix ");
  for( i = 0; i < p ; i++ )
    for( j = 0; j < q ; j++ )
       scanf("%d", &b[i][j] );

   /* multiplication logic */

   for( i = 0; i < n ; i++ )
   {
     for( j = 0; j < q ; j++ )
     {
       product = 0;

       for( k = 0; k < m ; k++ )
         product += a[i][k] * b[k][j] ;

       c[i][j] = product;
     }
   }

   printf(" multiplication result \n ");
   for( i = 0; i < n ; i++ )
   {
     printf("\n");

     for( j = 0; j < q ; j++ )
        printf(" %d" , c[i][j] );
   }
}
```

```
output :-
Enter n 3 ↵
Enter m 3 ↵
Enter p 3 ↵
Enter q 3 ↵
Enter first matrix 10 ↵
10 ↵
10 ↵
10 ↵
10 ↵
10 ↵
10 ↵
10 ↵
10 ↵
Enter second matrix 1 ↵
0 ↵
0 ↵
0 ↵
1 ↵
0 ↵
0 ↵
0 ↵
1 ↵
multiplication result
  10 10 10
  10 10 10
  10 10 10
```

Exercise :-

1) Write a program to find the inverse of a 3x3 matrix.
2) Write a program to print only the upper triangle of the matrix of an nxn matrix
3) Write a program to print only the lower triangle of the matrix of an nxn matrix.
4) Write a program to find the sum of the left and right diagonal elements of an nxn matrix.

Initializing arrays :-

We can directly assign values to an array.

e.g. int a[5]={10,20,30,40,50};

While assigning value to a single dimensional array we can also ignore the size of the array. In such case C compiler will automatically determine the size of the array using the number of values supplied to the array.

e.g. int a[]={10,20,30,40,50};

When supplied values are less than the size of an array, rest of the values are considered as zero.

e.g. int a[5] = {10,20};

Now a[0] is 10 and a[1] is 20 , a[2],a[3],a[4] are zeros.

Initializing Double Dimensional Arrays :-

e.g.

```
int a[3][2] = {
                ( 10 , 20 ),
                ( 30 , 40 ),
                ( 50 , 60 )
              };
```

Braces are just for readablility purpose.

(or)

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

(or)

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

*** we can ingore only the row size and not column size(i.e.2 here). If we ignore column size then compiler cannot determine no of elements per row. i.e. int a[][]={10,20,30,40,50,60};

How many rows can you guess ???

Multidimensional Arrays :-

We can also declare multi-dimensional arrays in C language. Upper limit on the number of dimensions is dependent on the compiler.

e.g.

```
int a[3][4][5];
```

Consider it as a 4x5 matrix upto this point.

Three 4x5 matrices.

Total number of integers : 3 x 4 x 5 = 60
Size of the array : 60 x 2 => 120 bytes.

These are seldom used and hence it is just for awareness.

next array of strings

Array of Strings :-

Consider the following example

```
char s[3][5] = { "abc","xyz","pqrs" };
```

Memory view of s is

s					
0	'a'	'b'	'c'	'\0'	
1	'x'	'y'	'z'	'\0'	
2	'p'	'q'	'r'	's'	'\0'

each row represents a separate string.

Here s[0] represents the first string, s[1] represents the second string and s[2] represents the third string.

A program to sort an array of strings using bubble sort :-

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
   char s[10][20];
   char temp[20];
   int i,j,n;

   clrscr();

   printf("no of strings you want to enter");
   scanf("%d",&n );

   for( i = 0; i < n ; i++ )
   {
      printf(" enter string %d :" , i );
      scanf("%s" , s[i] );
   }

   for( i = 1 ; i < n ; i++ )
     for( j = 0 ; j < n - i ; j++ )
        if( strcmp( s[j] , s[j+1] ) > 0 )
        {
            strcpy( temp , s[j] );
            strcpy( s[j] , s[j+1] );
            strcpy( s[j+1] , temp );
        }

    printf("\n strings after sort ");
    for( i = 0 ; i < n ; i++ )
      printf(" \n  %s ", s[i] );

    getch();
}
```

maximum 10 strings and each with capacity 20.

temporary space to hold one string with capacity 20.

note: no need or &

```
output :-
 no of string you want to enter 4 ↵
 enter string 0 : abc ↵
 enter string 1 : ab ↵
 enter string 2 : abd↵
 enter string 3 : aba ↵
 strings after sort
 ab
 aba
 abc
 abd
```

For logic see bubble sort in single dimensional arrays.

Problem :-
Calculate how much memory is wasted in the following declaration

```
char s[3][50]={"abc","xyz","pqrs"};
```

Initializing the character array :-

We can also initialize the character array by supplying the values directly.

e.g.

```
char s[4]="abc";
(or)
char s[]="abc";
(or)
char s[4]={'a','b','c','\0'};
(or)
char s[]={'a', 'b','c','\0' };
(or)
char s[4]={97,98,99,0};
(or)
char s[]={97,98,99,0};
(or)
char s[4]={97,98,99};
```

ASCII codes.

in this case last value is automatically assigned zero since size is 4 and number of values supplied are three.

Problem :-
Pick the odd one in the following declarations:-

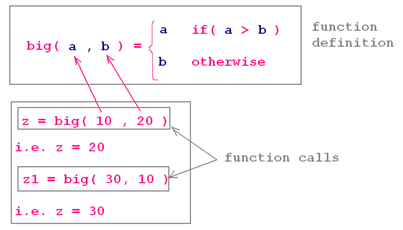
a) char str[5]="ABCD";
b) char str[]={'A','B','C','D','\0'};
c) char str[]={65,66,67,68};
d) char str[5]={65,'B',66,'D'};

scroll .....

Ans: It is (c) since str is not ended with null terminating character('\0').
i.e. char str[] = { 65 , 66 , 67 , 68 };
since size is not mentioned it is taken as 4 because of the number of values supplied. There is no place for '\0' i.e. 0. It is not a problem but in this case str should be used as a character array instead of string.

next Introduction to functions

Sample function example from maths :-



C languge program equivalent to the above :-

Note: written for integers only.

***Note: In the following example numbers with in the circle indicate execution sequence.

Prg57.c

```
#include <stdio.h>

int big( int a , int b )
{
  int result;

  if( a > b )
    result = a;
  else
    result = b;

  return result;
}
void main()
{
   int z,z1;

   z = big( 10 , 20 );

   z1 = big( 30 , 10 );

   printf(" z is %d  , z1 is %d ", z , z1 );
}
```

(3) a:10,b:20	(10) a:30,b:10
(4)	(11)
(5) a<b	(12) a>b
	(13) result=30
(6) result=20	
(7) return 20	(14) return 30
(1)	
(2) (8) z=20	
(9) (15) z1=30	
(16)	

```
output :
z is 20 , z1 is 30
```

Introduction to functions :-

A function is a code block which is defined to perform some specific task. When once a function is defined it can be used multiple times, which reduces the number of instructions we need to write in the program i.e. we can avoid code redundancy.

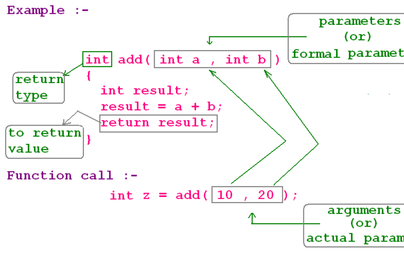
A function can receive values in the form of arguments passed to the function and it can return a single value.

General Structure :-

```
[return type] functionId( [ parameters ] )
{
   [ function body ]
}
```

*** if return type is ignored it is int by default.

Example :-



Example program to find ncr value for the given n and r values :-

In finding ncr we have to find n!,r! and (n-r)! i.e.the same factorial logic is repeated thrice.

Instead of repeating it, we can define a function for calculating factorial and supply different values to find factorial for different numbers such that factorial logic is written only once.

Prg58.c

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

int factorial( int n )
{
  int fact = 1,i;

  for( i = 1 ; i <= n ; i++ )
     fact = fact * i;

  return fact;
}
void main()
{
  int n,r,ncr;

  clrscr();

  printf(" enter n ");
  scanf("%d",&n );

  printf(" enter r ");
  scanf("%d",&r);

  ncr=factorial(n)/(factorial(r)*factorial(n-r));

  printf(" ncr value is %d ", ncr );

  getch();
}
```

value (supply)

value! (in return you get)

```
output :-
  enter n 5 ↵
  enter r 2 ↵
  ncr value is 10
```

return statement :-

It can be used to return a value to the calling function and when executed it will also terminate the execution of the function and return to the calling function.

Note: for a function whose return type is void , we can not return any value and hence in such case when we want to stop execution of the function and return from the function we have to use

```
return;
```

Note it is just return statement followed by semi-colon with out any value.

next examples using functions

A program to search whether the given number is prime or not using functions :-

$$\text{prime}(n) = \begin{cases} 1 & \text{if } n \text{ is prime} \\ 0 & \text{otherwise} \end{cases}$$

Prg59.c

```
#include <stdio.h>

int prime( int n )
{
   int i;

   if( n <= 1 )
     return 0; /* not prime */


   for( i = 2 ; i <= n / 2 ; i++ )
      if( n % i == 0 )
         return 0; /* not prime */

   return 1;  /* it is prime */
}

void main()
{
  int m;

  printf("Enter some value ");
  scanf("%d",&m );

  if( prime(m)  == 1 )
    printf(" it is prime ");
  else
    printf(" it is not prime ");
}
```

It is not three return values, it is only one return statement which can be executed.

Exercise :-
1) Write a program to check whether the given number is armstrong or not , use function ?
2) Write a program to check whether the given number is strong or not ?
3) Write a program to check whether the given number is perfect or not ?

Functions defined in terms of another :-

In simple words it is function invoking another function. Say

$$g(x) = x^2$$

$$f(x) = x.g(x-1)$$

what is f(5) ?

Here f(x) is defined in terms of g(x). Similarly one function can be defined using another.

A program to print all the prime number between n and m , where n and m are given as input :-

Logic :-

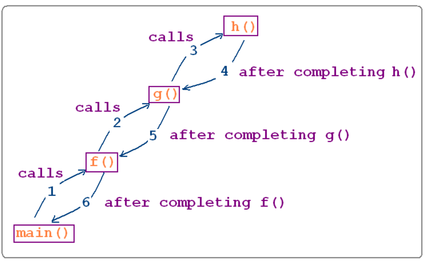
```
    for i = n to m
       if( i is prime )
           print i
```

Prg60.c

```
#include <stdio.h>

int prime( int n )
{
   int i;

   if( n <= 1 )
     return 0; /* not prime */


   for( i = 2 ; i <= n / 2 ; i++ )
      if( n % i == 0 )
         return 0; /* not prime */

   return 1;  /* it is prime */
}

/* here printprimes will not return any
value hence return type is void. */

void printprimes( int n , int m )
{
    int i;

    for( i = n ; i <= m ; i++ )
        if( prime( i )  == 1 )
            printf("  %d ", i );
}

void main()
{
  int start,end;

  printf("Enter start ");
  scanf("%d" , &start );

  printf("Enter end ");
  scanf("%d" , &end );

  printprimes( start , end );

}
```

Consider the following function call sequence

main() invokes f() , f() invokes g() and g() invokes h() then



Recurssive Functions :-

Normally people tend to say it as "a function that calls itself". But technically it is "a function that is defined in terms of itself".

Consider the following example :-

$$f(n) = \begin{cases} n.f(n-1) & \forall n > 1 \\ 1 & \text{otherwise.} \end{cases}$$

Now what is z = f(4)?

24
z = f(4)

6
f(4) = 4.f(3)

2
f(3) = 3.f(2)

1
f(2) = 2.f(1)

f(1)=1

referred as base condition where recurssion stops.
i.e. n <= 1

Program for the above example :-

Prg61.c

```
#include <stdio.h>

int f ( int n )
{
   if( n > 1 )
     return n * f( n - 1 );
   else
     return 1;
}

void main()
{
  int z = f(4);
  printf(" z is %d ", z );
}
```

Recurssive logic to generate fibonacci series :-

$$f(n) = \begin{cases} f(n-1) + f(n-2) & \forall n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0 \end{cases}$$

i.e.

		[f(1)+f(0)]	[f(2)+f(1)]	
f(0)	f(1)	f(2)	f(3)	....
0	1	1	2	

Program for the above example :-

Prg62.c

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

int f( int n )
{
   if( n == 0 )
      return 0;

   if( n == 1 )
      return 1;

   return f(n-1) + f(n-2) ;
}

void main()
{
   int i,n;

   clrscr();

   printf("Enter n ");
   scanf("%d",&n);

   for( i = 0; i <= n ; i++ )
      printf(" %d", f(i) );

   getch();
}
```

no need for else because if control enters if then it will return

```
output :
 Enter n 3
 0 1 1 2
```

Exercise :-

1) Write a recurssive function to print 1 to n numbers.
2) Write a recurssive function to compute the sum of n natural numbers.

Problem :-
Consider the following function, and calculate the value of f(25) ?

```
int f( int n )
{
  if( n == 0 )
    return 0;

  return ( n % 10 ) + f( n / 10 );
}
```

next function prototypes

Function Prototype Declaration :-

Consider the following code

Compiler will validate and translates the code in a sequential manner.

```
#include <stdio.h>

void main()
{
  double a=10.2,b=20.4;
  double c;

  c = add( a , b );

  printf("%lf",c);
}
..............
..............
..............
```

?

By the time compiler gets to the add function call it doesn't know what is add and it needs the following questions to be answered

1) Is add function available ?
2) Are arguments to add are two double values?
3) Will it return a double value?

If these questions are not answered then compiler can not determine whether the function call is right or wrong.

Q: How to let the compiler know what is add and how it behaves ?

Ans:

By declaring the prototype we can let the compiler know the details about the add method and later we can provide definition.

Compiler

knows what is add

validates the call based on prototype.

```
#include <stdio.h>

double add(double x, double y);

void main()
{
  double a=10.2,b=20.4;
  double c ;

  c = add( a , b );

  printf("%lf",c);
}
......
......
```

referred as function prototype.

Q: What happens when I don't provide definition now after declaring the prototype?

Ans:

Wonder is that the program will compile because compiler needs only type information to validate the statements, but when it comes to the linker it needs function definition with out which you will find linker error.

refer to " stages of translation " later

Example Program :-

Prg63.c

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

double add( double x , double y );

void main()
{
  double a = 10.2 , b = 20.4;
  double c;

  clrscr();

  c = add( a , b );

  printf(" %lf ", c );

  getch();
}

double add( double x , double y )
{
  return x+y;
}
```

valid according to

actual definition

Exercise :-

1) Similar to the above program write the functions for substraction, multiplication, division using function prototypes?

Pointers :-

A pointer is a memory block that holds the address of some other memory block.

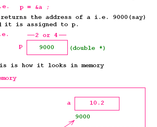
Observe the relation between the two :-

A small piece of paper with address

Similarly for every memory block there exists an address and we can store the address in another memory variable referred as pointer.

```
double a = 10.2 ;
```

Here 'a' is a double variable and it occupies 8 bytes in memory with value 10.2 in it.



How to store the address of a we have to declare the memory variable as

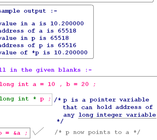
```
double * p ;
```

p is a memory block whose content will be double * i.e. address of some double variable.

How we can assign the address of a to p

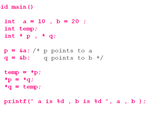
i.e. p = &a ;

&a returns the address of a i.e. 9000(say) and it is assigned to p.



This is how it looks in memory

Memory



what is the value of a ? 10.2
what is the value of &a ? 9000 (assumed)
what is the value of p ? 9000
what is the value of &p ? (address of p)

Q: How to access a through p ?

Ans: it is (*p). *p * is referred as dereferencing operator. Note one asterisk takes you one level up.

i.e.

```
printf(" %lf " , *p ); /* 10.2 */

(or)

*p = 20.2 ; /* modifies the value of a */
```

Example program :- Prg64.c

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

void main()
{
  double a = 10.2;
  double *p;      (or)  double *p = &a;
  p = &a ;
  printf("\n value in a is %lf ", a );
  printf("\n address of a is %u ",&a );

  printf("\n value in p is %u ", p );
  printf("\n address of p is %u ",&p );

  printf("\n value of *p is %lf ", *p );
}
```

```
sample output :-

value in a is 10.200000
address of a is 65518
value in p is 65518
address of p is 65516
value of *p is 10.200000
```

Fill in the given blanks :-

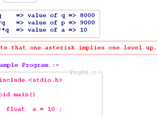
```
long int a = 10 , b = 20 ;

long int * p ;          /* p is a pointer variable
                           that can hold address of
                           any long integer variable.
                           */
p = &a ;                /* p now points to a */

printf( %ld ", *p );

p = &b ;                /* p now points to b */

printf( %ld ", *p );
```

Q:What is the output of the following code?
Ans:
Note: don't guess do it manually though you guess may be right.

Prg65.c

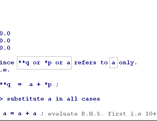
```
#include <stdio.h>

void main()
{
  int  a = 10 , b = 20 ;
  int temp;
  int * p , * q;

  p = &a; /* p points to a
  q = &b;    q points to b */

  temp = *p;
  *p = *q;
  *q = temp;

  printf(" a is %d , b is %d ", a , b );
}
```

Multiple levels of indirection :-



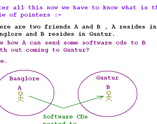
"long integer variable" i.e. a memory block that can hold some long integer value.

"Pointer variable to some long integer var." i.e. it can hold address of some long integer variable.



Double pointer i.e. pointer to a pointer here it is pointer to a "pointer to long"

How after the assignments




```
q    => value of q => 8000
*q   => value of p => 9000
**q  => value of a => 10
```

Note that one asterisk implies one level up.

Example Program :- Prg66.c

```
#include <stdio.h>

void main()
{
  float  a = 10 ;
  float *p = &a;
  float **q = &q;

  **q = a + *p ;

  printf("\n %f ",   a );
  printf("\n %f ",  *p );
  printf("\n %f ", **q );
}
```

What is the output ?

...scroll ....

```
20.0
20.0
20.0
```

since **q or *p or a refers to a only. i.e.

```
**q  =  a + *p ;
```

=> substitute a in all cases

```
a = a + a ; evaluate R.H.S. first i.e 10+10
```

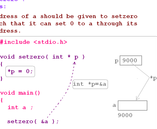
Here you might wonder that why don't we simply write int a=5; Instead of all this non-sense. Don't worry this is just the beginning.

The above approach i.e. sending address to function as argument , instead of a value is referred as call by reference.

It will be explained later.

Exercise :-
1) Write a function similar to the above setzero function to increment the value in the given address by 10.

Pointers as function arguments :-



Just like the above scenario, one function can access the data that belongs to another function through its address.

```
void setzero()
{
  a = 0;    <- error: a is undefined in setzero.
}                                          Banglore

void main()                                Guntur
{
  int a ;
  setzero();
  ......
}
```

Q: Then how to set a in main to zero from setzero ?
Ans:
Address of a should be given to setzero such that it can set 0 to a through its address.

```
#include <stdio.h>

void setzero( int * p )
{
  *p = 0;
}                          int *p=&a

void main()
{
  int a ;

  setzero( &a );

  printf("%d",a);
}
```

```
output :-
  0
```

Here you might wonder that why don't we simply write int a=0; Instead of all this non-sense. Don't worry this is just the beginning.

The above approach i.e. sending address to function as argument , instead of a value is referred as call by reference.

It will be explained later.

Exercise :-
1) Write a function similar to the above setzero function to increment the value in the given address by 10.

next call by value and by reference

Argument passing :-

There are two approaches in which we can send arguments(values) to the function
1) call by value
2) call by reference

1) call by value :-

Call the function by sending the value copy of the data block that you have. i.e. in simple words it is just like giving a photo copy of the content that you have. In such case your content is unaffected even if they change.

Example :-

Prg68.c

```
#include <stdio.h>

void change( int m )   (3) m  20
{
   m  += 10;           (4) m  30   No affect to n
}                      (5) m is undefined
void main()
{
   int n = 20;         (1) n  20

   change( n );        (2)

   printf(" %d ", n );  (6) n is unchanged.
}                          i.e. 20
```

2) call by reference :-

Call the function by sending the reference i.e. address i of the memory block that we have. Just like giving the original copy to the other person. In such case if other person makes any changes then original copy will be affected.

Prg69.c

```
#include <stdio.h>

void change( int * m )   (3) m  int *m = &n
                                m  6000
{
   *m  += 10;            (4)  *m += 10
}                        (5) m is undefined
void main()
{

   int n = 20;           (1) n  20 30
                                6000(say)

   change( &n );         (2)

   printf(" %d ", n );   (6)  output: 30
}
```

Mixing both :-

Prg70.c

```
#include <stdio.h>

void copy( int * dest  , int value )
{
   * dest = value;
}

void main()
{
   int m , n = 20;

   copy( &m , n );

   printf("\Nn n is %d ", n );
   printf("\n m is %d ", m );
}
```

dest 9000 value 20
&m (by ref.) n (value copy)
m 9000 n 20 8000

```
output :-
  n is 20
  m is 20
```

Example to swap two double values using pointers :-

Prg71.c

```
#include <stdio.h>

void swap( double * p , double * q )
{
  double temp;        p        q
  temp = *p;
  *p = *q;
  *q = temp;
}

void main()                 a  10.2    b  20.4
{
  double a = 10.2 , b = 20.4;

  swap( &a , &b );

  printf( "\n a is %lf ", a );
  printf( "\n b is %lf ", b );
}
```

```
output:-
 a is 20.4
 b is 10.2
```

Passing arrays to functions :-

Arrays can be passed to functions only by reference. i.e. we pass the address of the array to the function instead of sending duplicate copy of the array. Now function uses the sent address to access the content of the array.

To illustrate :-
Consider an array of 1000 integers.

```
     0    1          ...    999
   | 10 | 20 | ..... |  9  |  5 |
 a: 8888
   (say that this is the starting
    address of the array)
```

Which is better ?
creating a duplicate copy of a i.e. creating another array of 1000 integers and filling it with the values of a
(or)
✓ Just passing the address 8888 to the function and asking the function to read values using it.

Q: How can you obtain the address of the array , is it &a ?
Ans:

No, no need for &a since array id itself contains the starting address of the array. i.e. a itself is set to the base address of the array.
i.e. a is 8888 from the above example.

Prg72.c

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

void display( int b[] , int n )
{
  int i;

  for( i = 0; i < n ; i++ )
     printf(" %d ", b[i] );
}

void main()                 a:900 (say)
{                                  b 900  n 4
  int a[]={10,20,30,40};

  clrscr();

  display( a , 4 );

  getch();
}
```

by ref by value

**Note: Since we have sent the address of the array to the function, any modification done to the array with in the function will affect the original one.

Prg73.c

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

void read( int b[] , int m )
{
   int i;
   printf("\n Enter %d values \n",m);
   for( i = 0; i < m ; i++ )
       scanf( "%d" , &b[i] );
}

void sort( int b[] , int m )
{
   int i,j,temp;

   for( i = 1 ; i < m ; i++ )
      for( j = 0 ; j < m - 1 ; j++ )
         if( a[j] > a[j+1] )
         {
            temp = a[j] ;
            a[j] = a[j+1];
            a[j+1] = temp;
         }
}

void display( int b[] , int m )
{
  int i;
  printf("\n values are \n");
  for( i = 0; i < m ; i++ )
      printf( " %d ", b[i] );
}
void main()
{
  int n;
  int a[20];

  clrscr();

  printf(" Enter n ");
  scanf( "%d" , &n );

  read( a , n );
  sort( a , n );
  display( a , n );

  getch();
}
```

```
output :
  Enter n 5 ↵
  Enter 5 values
  10 ↵
  2  ↵
  1  ↵
  15 ↵
  12 ↵
  values are
  1 2 10 12 15
```

In all these calls we have sent the address of a since a contains address

Exercise :-

1) Write a program to replace all prime numbers in the array to zero using functions.i.e. replace(a , n); should replace prime numbers to zero.
Basic logic:-

```
for i = 0 to n-1
   if( prime(a[i]) == 1 )
       a[i] = 0;
```

2) Write a function to swap all alternative elements of an integer array.
i.e. swap(a , n);
should swap (0,2),(4,6),(8,10) etc.

3) Write a function to search for the given value in an array of double values and should return its index position of the first occurance when element is found otherwise return -1.
i.e.

```
   index = search( a , n , value );
   if( index != -1 )
          value found at index
   else
          value not found.
```

next global data and storage classes

Global data :-

Data that is defined outside the functions and it is accessible to all the functions in the program.

Prg74.c

```
#include <stdio.h>
int a;          global variable

void read()
{
   printf(" Enter some value ");
   scanf("%d", &a );
}

void main()
{
   read();
   printf(" a is %d " , a );
}
```

output:-
Enter some value 10 ↵
a is 10

*** Default value for global variable is zero.

It looks like it is simple to write programs using global data , but it should be avoided since they make the program hard to maintain. Hence we are not enhancing this topic further.

Storage classes :-

Every variable not only has a data type but also have a storage class. If you don't specify any storage class, the compiler will assume a storage class depending on the context in which the variable is used. C language supports four storage classes

1) auto
2) static
3) register
4) extern

1) auto (Automatic variable):-

These variables are also referred as automatic variables because they are created and destroyed automatically. They are also referred as local variables.

```
void f()
{
  int a; (or) auto int a;
  ......
  ......
  ......
  ......
}
```

created automatically when f() is invoked.

a is available for this block.

a is destoryed(undefined) after the end of this function.

C language supports blocks inside the function and automatic variables may be declared inside the block too.

Prg75.c

```
#include <stdio.h>
void main()
{
   int b;
   b = 10;

   {
     int a ;/* a created here*/
     a = 10;
     printf(" a is %d " , a );
   }/* a destroyed here */

   printf(" b is %d " , b );

   printf(" a is %d " , a );
}
```

block

a is local to this block.

a is undefined after the block.

Note: In C language variables which are needed inside the block are declared immediately after starting the block.

Example program :-

Prg76.c

```
#include <stdio.h>
void f()
{
  int a = 0;                 a [ 0 ]
  a++;                       a [ 1 ]
  printf("\n a is %d " , a ); /* a is 1*/
}              /* a destroyed */
void main()
{
  f();
  f();
  f();
}
```

output :
a is 1
a is 1
a is 1

As you can observe in the output a is created and destroyed in each call to f(). Hence a is referred as automatic variable which is created and destroyed automatically.

2) static storage class :-

When a local variable is declared as static it will be created once the control enters into the block and will not be destroyed even after the end of the function. Next time when the same function is invoked, previously allocated memory block for the variable is used instead of creating new one. In technical terms

"A static local variable's memory is shared among the function calls to the same function".

Example :- Prg77.c

```
#include <stdio.h>
void f()
{
  static int a=0;
  a++;
  printf("\n a is %d " , a );
} /* a is not destroyed */
void main()
{
  f();
  f();
  f();
}
```

this is initialization and is executed only when a is created. i.e. first time

all three calls use the same memory block for a.

output :-
a is 1
a is 2
a is 3

**Note: static variables are allocated in global memory and hence their default value is zero. Here there is no need for initialization.

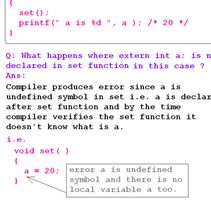
**Note: static is also used in another context i.e. to make the global identifier accessible with in the module.
Don't worry about this point now.

3) register storage class :-

A variable declared under register store class is given a CPU register if available otherwise it is treated as automatic variable.

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

Normally i will be here.



i can be accessed much faster when it is given a cpu register instead of placing it in memory. Normally a loop variable is accessed more frequently and inorder to enhance performance we declare it under register storage class. But there is no guarantee that a CPU register is allocated because they are limited in number.

4) extern storage class :-

extern is used to declare an identifier and its definition can be found elsewhere in the program. Let us illustrate it with an example

Prg78.c

```
#include <stdio.h>
void set()
{
  extern int a;
  a = 20;
}

int a;

void main()
{
  set();
  printf(" a is %d ", a ); /* 20 */
}
```

tells the compiler that a is an integer, but it is defined else where note that it is not local variable.

Q: What happens where extern int a; is not declared in set function in this case ?
Ans:
Compiler produces error since a is undefined symbol in set i.e. a is declared after set function and by the time compiler verifies the set function it doesn't know what is a.
i.e.

```
void set( )
{
  a = 20;
}
```

error a is undefined symbol and there is no local variable a too.

Main purpose of extern is to declare an identifier before its definition so that we can use it.

Technical Tips :-

Declaration :
Specifying characteristics of the identifier.
Definition :
Allocating memory for the identifier.

next structures and unions

Structures :-

Structures are used to create composite datatypes i.e. a datatype that is formed using multiple datatypes.

e.g.

```
struct Point
{
   int x;
   int y;
};
```

structure name or tag name

Creates a type called Point with two integer fields x and y.

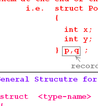
Now we can create variables of type Point to get memory block with two fields x and y.

e.g.

```
struct Point  p,q ;
```

Type Variables

In memory we can view p and q as



To access x and y

```
p.x = 10;
p.y = 20;
```

```
q.x = 30;
q.y = 40;
```

i.e. structureVariableName.memberid

Note: instead of creating record variables p and q separately we can directly declared them at the end of the structure

```
i.e.  struct Point
      {
         int x;
         int y;
      } p,q ;
```

record variables.

General Strucutre for defining structures :-

```
struct  <type-name>
{
   [ field-type  variable[,variable...;]]
   [ field-type  variable[,variable...;]]
   ...
} [ variable-list] ;
```

Example :-

Structure used in the following example

```
struct Student
{
 int sno;
 char name[20];
 double per;
} s1 ;
```

typename, if there is no need to create Student records anymore then we can ignore it too.

```
i.e. struct
     {
        int sno;
        // etc
     } s1;
```

This is referred as anonymous structure.
i.e. a structure with no name.

s1 is a record variable of type student, we can view s1 as

Prg79.c

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

struct Student
{
  int sno;
  char name[20];
  double per;
};

void main()
{
   struct Student s1;

   printf("Enter student number ");
   scanf("%d" , &s1.sno );

   printf("Enter student name ");
   scanf("%s" , s1.name );

   printf("Enter percentage ");
   scanf("%lf" , &s1.per );

   clrscr();

   printf("\n Number : %d ", s1.sno);
   printf("\n Name   : %s ", s1.name);
   printf("\n Percentage : %lf ", s1.per );

   getch();
}
```

Example program to read two points and calculate the distance between the two.

$$\text{distance} = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Prg80.c

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

struct Point
{
   int x,y;
};

void main()
{
  struct Point p,q;
  double distance;
  int diffx,diffy;

  printf("Enter first point ");
  printf("Enter x : " );
  scanf("%d" , &p.x );
  printf("Enter y : ");
  scanf("%d" , &p.y );

  printf("Enter second point ");
  printf("Enter x : " );
  scanf("%d" , &q.x );
  printf("Enter y : ");
  scanf("%d" , &q.y );

  diffx = p.x - q.x ;
  diffy = p.y - q.y ;

  distance = sqrt(diffx*diffx + diffy*diffy);

  printf(" distance is %lf " , distance );

  getch();
}
```

```
output:
   Enter first point Enter x : 6
   Enter y : 10
   Enter second point Enter x : 3
   Enter y : 6
   distance is 5.0
```

Accessing records through pointers :-

Consider the following structure

```
struct Point
{
   int x;
   int y;
};
```

example declarations

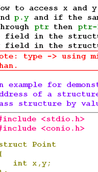
```
struct Point    p ;
struct Point * ptr ;
```

here p is a record variable of type Point and ptr is a pointer. i.e.

It can contain address of any record of type Point.

after assigning address of p to ptr

```
i.e. ptr = &p;
```



Now to access x and y through p we use p.x and p.y and if the same must be done through ptr then ptr->x and ptr->y i.e. the x field in the structure pointed by ptr and y field in the structure pointer by ptr.

Note: type -> using minus followed by greater than.

An example for demonstrating how to pass address of a structure to function and how to pass structure by value :-

Prg81.c

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

struct Point
{
  int x,y;
};

/* ex for call by reference */

void read( struct Point  *  ptr )  (3)
{
   printf("\n Enter x : ");
   scanf("%d" , &ptr->x );
   printf("\n Enter y : ");
   scanf("%d" , &ptr->y );
}
/* ex for call by value  */
void display( struct Point  q )  (6)
{
   printf("\n x : %d ", q.x );
   printf("\n y : %d ", q.y );
}

void main()
{
   struct Point p ;  (1)

   read( &p );       (2)

   display( p );     (5)
}                    (8)
```

p.t.: &p

(4)

q: 2 3 x y

(7)

p: 2 3 x y

Observe in (3) ptr takes address of p where as in (6) q takes a separate copy of p. (3) demonstrates call by reference and (6) demonstrates call by value.

In (4) we are reading values into x and y using ptr. Since ptr points to p , ptr->x and ptr->y refer to x,y in p.

Example for array of structures :-

Following example uses Product structure with three fields(pcode,name,price,qty). Reads a product records and displays the product descriptions with their total amount with 124 tax added.

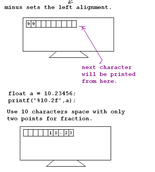
Structure definition :-

```
struct Product
{
  int pcode;
  char name[20];
  int qty;
  double rpi;
  double amount;
};
```

Array of structures :-

```
struct Product  p[3];
```

defines an array of three product records.



Brief description about formatted output :-

```
int a=9;
printf("%10d",a);
```

Uses 10 characters space to print a i.e.

```
printf("%-10d",a);
```

minus sets the left alignment.

next character will be printed from here.

```
float a = 10.23456;
printf("%10.2f",a);
```

Use 10 characters space with only two points for fraction.

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

struct Product
{
  int pcode;
  char name[20];
  int qty;
  double rpi;
  double amount;
};

void read( struct Product arr[] , int n )
{
  int i;
  for( i = 0 ; i < n ; i++ )
  {
    printf("\n Enter product details ");

    printf("\n Enter pcode : ");
    scanf("%d" , &arr[i].pcode );
    printf("\n Enter name : " );
    scanf("%s" , arr[i].name );

    printf("\n Enter quantity : " );
    scanf("%d" , &arr[i].qty );

    printf("\n Enter rate per item : " );
    scanf("%lf" , &arr[i].rpi );

    arr[i].amount = arr[i].qty * arr[i].rpi;
  }
}

void printline( int n )
{
  int i;
  printf("\n");
  for( i = 1; i <= n ; i++ )
    putch('-');
}

void display( struct Product arr[], int n )
{
  double total,tax;
  int i;
  total = 0;
  clrscr();
  printline(59);
  printf("\n");

  printf("%-10s %20s %5s %10s %10s",
    "code","name","qty","rpi","amount");

  printline(59);
  for( i = 0 ; i < n ; i++ )
  {
    total += arr[i].amount;
    printf("\n%-10d",arr[i].pcode );
    printf("%20s ", arr[i].name);
    printf("%5d ", arr[i].qty );
    printf("%10.2lf ", arr[i].rpi );
    printf("%10.2lf ", arr[i].amount );
  }
  printline(59);

  printf("\n");
  printf("%48s","total");
  printf("%11.2lf",total);

  printf("\n");
  printf("%48s","tax");
  printf("%11.2lf",tax);

  printline(59);

  printf("\n");
  printf("%48s","amount");
  printf("%11.2lf",total+tax);

  getch();
}

void main()
{
  struct Product products[20];
  int n;
  clrscr();
  printf(" Enter number of products ");
  scanf("%d",&n);

  read( products , n );
  display( products , n );
}
```

```
/*
  add this function due to a simple bug in Turbo C.
*/
void link()
{
  double a,*b=&a;
  a=*b;
}
```

output :-

```
Enter number of products : 1
Enter product details
Enter product code : 1
Enter name : rice
Enter quantity : 10
Enter rate per item : 45

-----------------------------------------------------------
code      name      qty        rpi       amount
-----------------------------------------------------------
1         rice      10       45.00       450.00
-----------------------------------------------------------
                                total     450.00
                                  tax      54.00
-----------------------------------------------------------
                              amount     504.00
```

Note: Use typedef to remove struct from each declaration.

e.g.

```
typedef  char   byte;
```

existing typename → new typename

All references to byte internally imply that they are characters.

e.g.

```
byte b;
```

internally imply it is char b;

Similarly we have a Product structure as follows.

```
struct Product { ..... };
```

now to create variable

```
struct  Product  p;
```

create an alias for "struct Product" and use it directly

```
typedef  struct  Product  ProductRecType;
```

From now on ProductRecType imply to struct Product.

```
ProductRecType   p;
```

Example :-

```
#include <stdio.h>

typedef  int  Phone;

void main()
{
  Phone  p;

  printf("Enter your phone number ");
  scanf("%d",&p );

  printf("Your phone number %d ", p);
}
```

Unions :-

A union is similar to a struct, except it allows you to define variables that share storage space.

Syntax:

```
union [conion type name>] {
    <type> <variable names> ;

  } [conion variables>] ;
```

For example,

```
union  int_or_long {
    int     i;
    long    l;
  } somenumber;
```

Here somenumber is given 4 bytes space since the largest member in the union long. And both i and l share the same space. You have to use only one of them at a time , but not both. Writing to one will overwrite the other.

Elements of union are accessed in the same manner as structure.

i.e.

```
somenumber.i
somenumber.l

somenumber
```

** Where the name is declared as structure then it will be given six bytes 2 for integer and 4 for long. But in case of unions all variables in union share the largest required space.

Example :-

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

union int_or_long
{
  int  intpart ;
  long longpart ;
} somenumber;

void main()
{
  int choice;

  while( 1 )
  {
    printf("\n\n");
    printf("Which value you want to enter");
    printf("\n 1: integer ");
    printf("\n 2: long ");
    printf("\n 3: quit ");
    printf("\n Enter your choice:");
    scanf("%d",&choice);
    switch( choice )
    {
     case 1:
      printf("Enter integer value");
      scanf("%d", &somenumber.intpart );
      printf("value : %d ",somenumber.intpart);
      break;
     case 2:
      printf("Enter long value ");
      scanf("%ld",&somenumber.longpart);
      printf("value: %ld",somenumber.longpart)
      break;
     case 3:
      exit(0);
    }
  }
}
```

output:

```
Which value you want to enter
1: integer
2: long
3: exit
Enter your choice: 1
Enter integer value 40
value : 40

Which value you want to enter
1: integer
2: long
3: exit
Enter your choice: 2
Enter long value 40000
value : 40000

Which value you want to enter
1: integer
2: long
3: exit
Enter your choice: 3
```

Note that we are using either one of integer or long but not both at a time.

next dynamic memory allocation

Dynamic Memory Allocation :-

Memory allocation done at runtime (while program is in execution) is referred as dynamic memory allocation.

Advantages of dynamic memory allocation :-

1) We can allocate the required number of blocks.
2) There is a provision to release those blocks when they are not needed anymore.
3) There is no wastage or shortage of memory

How to allocate and release ?

Use malloc and free functions available in alloc.h to allocate and release blocks.

e.g.

```
      int * arr;
      arr = (int *) malloc( 10 );
```

no of bytes

The above malloc function allocates 10 bytes of memory and returns the starting address of the allocated bytes which is held in arr for further use. If malloc fails to allocate then it returns NULL(0).

Access those allocated blocks just like an array i.e. arr[0],arr[1] etc. if the allocated blocks represent an array.

Use free function to release those blocks when they are not needed.

```
     i.e.
       free( arr );
```

Example :-

Prg85.c

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

void main()
{
   int   n,i;
   int * arr;

   clrscr();

   printf("Enter n");
   scanf("%d",&n);

   arr = (int *) malloc( n * sizeof(int) );

   if( arr == NULL )
   {
      printf("Unable to allocate memory");
      return;
   }

   printf("Enter values ");
   for( i = 0; i < n ; i++ )
      scanf("%d", &arr[i] );

   printf("Values are ");
   for( i = 0; i < n; i++ )
      printf(" %d ", arr[i] );

   free( arr );
}
```

(1) (2)

simply return with out value since it is void main

(1) allocates required number of integers i.e. n .
In the past programs just recall what we have done

```
      int arr[20]
      int n;
      .... read n....
      .... use n integers of arr .....
```

But what happens when n is 5 then we have wasted 15 integers. And also what happens when n is 25 there is 5 integers shortage.

Also there is no provision to release those blocks when they are not needed. But now we can release using free. (2)

Declaration and allocation explained :-

```
int * arr;                arr [(int *)]
                      .... say n = 5 ....
arr = (int *) malloc( n * sizeof(int) );
[9000] = [ | | | | ]
arr ---> 9000
```

sizeof operator :-

Returns the size of the resultant datatype of the given expression.

```
e.g. sizeof( char )=>1
     sizeof( int  )=>2[machine dependent]
     sizeof( 10.2 )=>8[since double]
     sizeof( 10L  )=>4
```

Note: sizeof is operator and not function.
i.e. sizeof int
also evaluates to 2.

Q:What is (int *) before malloc function call?

Ans:

malloc is used to allocate bytes in memory and they may belong to any datatype. So, malloc is more general function that is applicable to all datatypes. Hence its return type is (void *) referred as generic pointer i.e. address that may belong to any datatype.

Now consider the following statement

```
int  *  arr ;    /* can hold the address of
                    any integer */

arr = ___________ malloc( 2 );
```

expected (int *)

returns void * i.e. simply address

i.e. what is expected is integer address and what is given is just address.
Is types matched here ? No. Hence we need to typecast it to (int *) so that L.H.S and R.H.S will match and compiler can accept the assignment.

```
arr = (int *)  malloc( 2 );
```

consider that address is int *

Suggestion : As a beginner it may be beyond your reach, even if you understand it now it is partial don't worry. Pointers is a topic with a lot of depth you need time for it. Have you understood what is dynamic memory allocation? if yes it is enough.

calloc :-

Just like malloc it too allocates memory dynamically but the only difference is that it will fill the allocated blocks to zero.

Above malloc call using calloc :-

```
arr = ( int * ) calloc( n , sizeof(int) );
```

see here it is no-of blocks, sizeof each block. Where as it is no of bytes in malloc. i.e. n*sizeof(int)

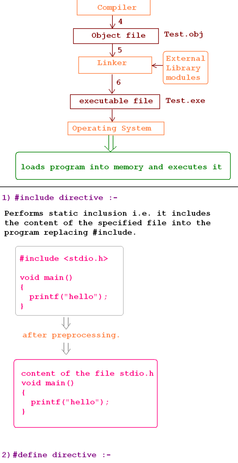
next pre-processor directives

Pre-processor directives :-

Program lines other than comments which start with # as the first non-white space character are referred as pre-processor directives.
i.e. they instruct pre-processor to do some activity.

What is pre-processor?
Initial part of the translation unit(such as compiler , linker etc.) which is responsible for processing pre-processor directives
i.e. #include , #define etc.

Translation Phases :-



1) #include directive :-

Performs static inclusion i.e. it includes the content of the specified file into the program replacing #include.

```
#include <stdio.h>

void main()
{
   printf("hello");
}
```

after preprocessing.

```
content of the file stdio.h
void main()
{
   printf("hello");
}
```

2) #define directive :-

It is used to define symbolic constants and macros. Content defined using #define is expanded in place of the call.

```
#include <stdio.h>

#define MAX 5

void main()
{
   int arr[MAX],i;

   for( i = 0; i < MAX ; i++ )
   {
     scanf("%d",&arr[i] );
     printf("%d",&arr[i] );
   }
}
```

after preprocessing.

```
content of the file stdio.h

void main()
{
   int arr[5],i;

   for( i = 0; i < 5 ; i++ )
   {
     scanf("%d",&arr[i] );
     printf("%d",&arr[i] );
   }
}
```

Observe that all occurances of MAX are replaced with 5.

2.1) defining macros using #define :-

Macro Definition : It is a set of statements identified using a unique name and those statements are expanded in place of every call to the macro.

Prg86.c

```
#include <stdio.h>

#define sqr(x)   x*x

void main()
{
   int a,b,c;

   a = sqr(4) ;
   b = 64 / sqr(4) ;
   c =  sqr(2+5) ;

   printf(" a is %d ", a );
   printf(" b is %d ", b );
   printf(" c is %d ", c );
}
```

Not a function call, it is macro call and it is expanded in place of the call.

after preprocessing.

```
content of the file stdio.h

void main()
{
   int a,b,c;

   a = 4 * 4;
   b = 64 / 4 * 4 ;
   c = 2 + 5 * 2 + 5;
   printf(" a is %d ", a );
   printf(" b is %d ", b );
   printf(" c is %d ", c );
}
```

Hence the more appropriate way to define sqr is

```
#define  sqr(x)   ((x) * (x))
```

evaluate it.

2.2) Nesting of macro calls

Prg87.c

```
#include <stdio.h>

#define sqr(x)  ((x)*(x))
#define cube(x) ((x)*sqr(x))

void main()
{
  int a = cube(2+5);
  printf("%d",a);
}
```

expanded as
int a=((2+5)*((2+5)*(2+5));

Can you identify the drawback?
Same expression 2+5 is evaluated thrice.

Q: What is the advantage of writing a macro instead of a function ?
Ans:
If sqr is written as a function i.e.

```
int sqr(int x)      (2)
{
   return x*x;      (3)
}
.......
.......
.......
int n=sqr(4);     (1) (4) n=16.
```

As you can see in the above code when a function is invoked control must switch to the function, process the code and then it has to return. Hence it needs additional processing time.
Where as in macro since definition is expanded in place of the call there is no need to switch and hence it reduces processing time.
i.e. int n=sqr(4);
is expanded as int n = 4*4; (if sqr is a macro)

Hence for those functions which can be expanded (i.e. a function with few lines of code) can be defined as a macro to enhance processing speed.
Caution: Macros doesn't have type checking i.e. we can supply any type of value for arguments. Where as in functions arguments must match with parameter type declaration.

3) #undef :-

Used to undefine the symbolic constant defined using #define.

```
#define MAX  5

  .....
  ..... here whereever MAX is referred
  ..... it is replaced with 5.

#undef MAX

  .... MAX is not available from now.
```

Conditional Compilation directives :-

4) #if, #ifdef, #else, #elif ,#endif

```
#include <stdio.h>

#define DEBUG 0

void main()
{
   #if DEBUG
     printf("it is printed when DEBUG is 1");
   #endif

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

after preprocessing.

```
content of the file stdio.h
void main()
{
   printf("\n hello");
}
```

Note that the printf in #if is not expanded since DEBUG is 0 and it is considered as false. If DEBUG is set to non-zero value e.g. 1 then it is considered as true and printf will be expanded. This is referred as conditional compilation. In simple words it can be considered as conditional expansion of code.

next enumerated datatypes.

Enumerated Datatypes :-

Used to define symbolic constants so that they can be used in the program to make it more readable. Indirectly it is an integer only.

e.g.

```
enum Day {SUN,MON,TUE,WED,THU,FRI,SAT};
```

Note: Since first constant is not given any value it is taken as zero. Next one will be given previous constant value + 1.

we have declared 7 symbolic constants under enumerated type Day. SUN is a constant with value 0. MON is given 1 i.e. (Prev+1) etc.
Consider the following declaration

```
enum  Day   day_of_the_week;
```

holds one of the value from those seven values i.e. (0 to 6). It sounds better than

```
int  day_of_the_week;
```

since it declares that variable day_of_week holds one value in the range of an integer i.e. -32768 to 32767.

Other examples :-

```
enum  { A=10 , B , C=20 , D , E=-1 , F};
```

Here A is 10 , B is 11 , C is 20 , D is 21, E is -1 , F is 0.

Prg89.c

```
#include <stdio.h>

enum Day {SUN,MON,TUE,WED,THU,FRI,SAT};

void main()
{
   enum  Day  day_of_the_week;

   printf("Enter dayno ");
   scanf("%d", &day_of_the_week );

   switch( day_of_the_week )
   {
   case SUN:
         printf("Sunday"); break;
   case MON:
         printf("Monday"); break;
   case TUE:
         printf("Tuesday"); break;
   case WED:
         printf("Wednesday"); break;
   case THU:
         printf("Thursday"); break;
   case FRI:
         printf("Friday"); break;
   case SAT:
         printf("Saturday"); break;

   }
}
```

```
output :-
Enter dayno 2 ↵
Tuesday
```

Command Line Arguments :-

Arguments which are supplied to the program while submitting the program for execution. These arguments are received by main function.

Example:-

Prg90.c

```
#include <stdio.h>

void main( int argc , char * argv[] )
{
  int i;
  clrscr();
  printf("no of arguments are %d",argc );
  printf("\n they are ");
  for( i = 0; i < argc ; i++ )
    printf("\n arg %d is %s",i,argv[i]);
  getch();
}
```

Steps for execution:-

1) compile the program (Alt+F9) => Prg90.obj
2) run the make option (F9) => Prg90.exe
3) open file menu => select dos shell
   You will be in the directory where
   Prg90.exe is available. e.g. c:\tc\bin

4) c:\tc\bin> Prg90 abc xyz 100 ↵

dont type this

referred as command line with arguments.
These are supplied to main

argc : 4	
argv[0]:	path to the file Prg90.exe itself
argv[1]:	"abc"
argv[2]:	"xyz"
argv[3]:	"100"

5) Type exit to return back to TC.
   i.e.
   c:\tc\bin> exit ↵

Example program to find the sum of the values supplied in the command line arguments :-

Note:
atoi is a macro defined in stdlib.h converts the given string into integer. i.e.
atoi("92") => 92

string "92"

'9'	'2'	'\0'

value 92

similarly you can use atof function to translate string to double.

***Note:
Though you may supply some integer value as command line argument , it is given in the form of string to main.

Prg91.c

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

void main( int argc , char * argv[] )
{
  int sum = 0;
  int i;

  /* ignore zero argument since it is
     file path itself */

  for( i = 1; i < argc ; i++ )
     sum += atoi( argv[i] );

  printf(" sum is %d " , sum );
}
```

Steps:-
1) compile (Alt+F9) and make(F9)
2) go to dos shell
3) c:\tc\bin> Prg91 10 20 30 40 ↵
   sum is 100
4) c:\tc\bin> exit ↵

Exercise :-
1) Write a program to find the biggest in the given command line arguments
2) Write a program to concatenate all the command line arguments
3) Write a program to count the number of duplicates in the given command line arguments.

next file processing

File Processing :-

File :-

A file is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind.
Types of files can be broadly divided into two types

1) Text files
2) Binary files

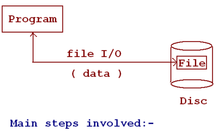
1) Text files :-

Text files contain a sequence of characters or multiple lines of text. Examples are C program files written till now e.g. Prg1.c, Prg1.cpp, AutoBiography.txt(caution: harddisk space is not enough for this file) etc.

2) Binary files :-

Content of these files is binary data such as audio,video, executable programs(e.g. Prg1.exe) and those files which contain records etc.

Processing files from program :-



Main steps involved:-

1) open file
2) read/write data
3) close file

1) opening file :-

Before opening a file we have to identify the mode in which we want to open the file i.e.

File modes :-

```
read mode(r) :-
  to read data from file
write mode(w):-
  to replace content of the file.
append mode(a):-
 to add content to the file.
```

```
Additional modes :-
   r+ : to read and write
   w+ : create new file for update
        (read and write)
   a+ : open for append at the end of the
        file.
```

```
Note: To open the file in text mode
attach t. i.e. rt , r+t etc.
To open the file in binary mode attach b.
i.e. rb , r+b.
```

Function to open file is fopen , it opens the given file and returns a pointer to the FILE structure which is used to perform subsequent operations on the file.

e.g.

```
FILE * fp = fopen("demo.txt","wt");

if( fp == NULL )
{
  printf("Unable to open the file ");
  exit(1);
}
```

3) closing file :-

Use fclose function to close the file. It is important to close the file since it will release the resources associated with the file.

```
e.g.
 fclose( fp );
```

2) read or write operations :-

There are numerous functions to perform these operations on any stream. For us most important functions are fgetc,fputc,fread and fwrite for now.

fgetc :-
Reads a character from the given stream. On success it returns the character and in case of failure it returns EOF.

fputc :-
Outputs a character to the stream.

Example :-

Prg92.c

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

void main(int argc , char * argv[] )
{
 FILE * src , * dest ;
 char ch;

 if( argc != 3 )
 {
  printf("usage!prg92 <srcfile> <destfile>");
  exit(1);
 }

 /* Note:
    argv[1] contains source file name and
    argv[2] contains destination file name.
    source file is opened in read mode and
    destination in write mode.
 */

 src = fopen( argv[1] , "r" );

 if( src == NULL )
 {
   printf("unable to open source file");
   exit(1);
 }

 dest = fopen( argv[2] , "w" );

 if( dest == NULL )
 {
   printf("unable to open destination file");
   exit(1);
 }

 /*
    read one character from source and
    write it to destination,
    repeat it until source file reaches
    end of file (EOF).
 */

 while( ( ch = fgetc( src ) ) != EOF )
   fputc( ch , dest );


 fclose( src );
 fclose( dest );
}
```

Steps for execution :-

1) compile (Alt+f9)
2) make (F9)
3) open dos shell (file menu => dosshell)

4) c:\tc\bin> `Prg92  prg80.c  temp.c` ↵

executable file Prg92.exe
source file argv[1]
destination file argv[2]

5)To see whether it is copied or not
c:\tc\bin> type temp.c ↵

6)type exit to return to tc.
c:\tc\bin> exit ↵

Note: Have to identified, here we have written a command partially equivalent to copy command in dos.

next file processing part 2

Processing with binary files :-

Let us illustrate binary file processing through product record we will read and write product records from/to file.

e.g.

```
struct Product
{
    int pcode;
    char name[20];
    double price;
};
```

When a record is created for the above Product structure it will occupy 30 bytes in memory(2 + 20 + 8).

e.g.

```
struct Product p=(1,"rice",90);
```

	— 2 —	— 20 —	— 8 —
p	1	rice	90

Now we can write this product record to file with the help of fwrite function.

```
fwrite( &p , sizeof(p) , 1 , fp );
```

Address from which data must be obtained.
length of each data item.
no of items
specifies output stream

Similarly to read a record from the file use fread and note that syntax is similar to the above except the process is reversed.

```
fread( &p , sizeof(p) , 1 , fp );
```

Address in which data should be stored.
length of each data item.
no of items
specifies input stream

Consider a new file product.dat

```
FILE * fp = fopen("product.dat","r+");
```

opens the file in read and write mode.

product.dat

```
0 1 2 ....
^
file
pointer
```

After writing the first record.

```
struct Product p1 = (1,"rice",90);
fwrite( &p1 , sizeof(p1) , 1 , fp );
```

product.dat

```
1 rice 90
0        29 30
            ^
          file
          pointer
```

After writing the second record.

```
struct Product p2 = (2,"wheat",50);
fwrite( &p2 , sizeof(p2) , 1 , fp );
```

product.dat

```
1 rice 90 2 wheat 50
0        29 30      59 60
                       ^
                     file
                     pointer
```

As you can identify filepointer points to the next write position and this is the same for read operation too.

We can reposition this pointer to the required index with the help of fseek function.

e.g.

```
fseek( fp , 0 , SEEK_SET );
```

Moves filepointer to 0. Explained later.

Before moving to fseek lets us take a look at how to read.

product.dat

```
1 rice 90 2 wheat 50
0        29 30      60
^
file
pointer
```

Now reading the first 30 bytes will give you first record and file pointer will point to 30. Reading next 30 bytes will get the second record. etc.

product.dat

```
1 rice 90 2 wheat 50
0        29 30      59 60
            ^
          file
          pointer
```

(after 1st read)

fseek explained :-

Consider the above file with 4 records

product.dat

```
1 rice 90 2 wheat 50 3 sugar 20 4  soap 20 EOF
0        30        60        90        120
            ^
          file
          pointer
```

Note: Consider 60 as the current position for all the following example.

There are three base positions from which we can calculate seek operations

```
SEEK_SET  : beginning of the file (0)
SEEK_CUR  : current position ( e.g. 60)
SEEK_END  : end position ( e.g. 120)
```

fseek(fp , n , position): moves the pointer n bytes from the given position.

	next pos
1) fseek(fp , -30 , SEEK_CUR) :-	30
2) fseek(fp , 30 , SEEK_CUR) :-	90
3) fseek(fp , 0 , SEEK_SET) :-	0
4) fseek(fp , 30 , SEEK_SET) :-	30
5) fseek(fp , 0 , SEEK_END) :-	120
6) fseek(fp , -30 , SEEK_END) :-	90

To move to n^{th} record:-

```
fseek( fp , (n-1)*sizeofrec , SEEK_SET )
```

i.e. To move to 3rd record.

```
fseek( fp , 2 * 30 , SEEK_SET );
```

Since we can move directly to any record in the binary file this property makes it Random Access File.
A file is said to be Random Access File if it allows you to access records from any given position with out processing the previous records.

Example :-

Prg93.c

```
#include <stdio.h>

struct Product
{
    int pcode;
    char name[20];
    double price;
};

void print( struct Product q )
{
  printf("\n product details ");
  printf("\n pcode : %d ", q.pcode );
  printf("\n name : %s ", q.name );
  printf("\n price : %lf ", q.price );
}

void main()
{
  FILE * fp;

  struct Product p1 = (1,"rice",90);
  struct Product p2 = (2,"wheat",50);
  struct Product p3 = (3,"sugar",20);

  struct Product temp;

  fp = fopen("product.dat","w+b");

  if( fp == NULL )
  {
    printf("Unable to open file");
    exit(1);
  }

  printf("\nCurrent position:%ld",ftell(fp));

  fwrite( &p1 , sizeof(p1) , 1 , fp );
  fwrite( &p2 , sizeof(p2) , 1 , fp );
  fwrite( &p3 , sizeof(p3) , 1 , fp );

  printf("\nCurrent position:%ld",ftell(fp));

  fseek( fp , 0 , SEEK_SET );

  printf("\n After fseek : %ld ",ftell(fp));

  /* read each record upto end of file */
  while ( ! feof( fp ) )
  {
    fread( &temp , sizeof(temp) , 1 , fp );
    if( ! feof( fp ) )
        print( temp );
  }

  fclose( fp );
}
```

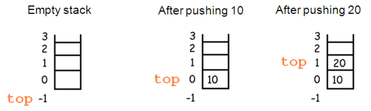
```
output:-
 Current position : 0
 Current position : 90
 After fseek : 0
 product details
 pcode : 1
 name : rice
 price : 90.000000
 product details
 pcode : 2
 name : wheat
 price : 50.000000
 product details
 pcode : 3
 name : sugar
 price : 20.000000
```

Stack :-

A data structure that follows last in first out(LIFO) i.e last inserted element must be the first to come out. E.g. A glass full of lemons. i.e last inserted lemon must be the first to come out of the glass.

The basic operations of stack are push and pop. Push to push the element on to the stack i.e insertion and pop to remove the element from the stack.

Following diagrams illustrate stack operations :-



Observe that top points to the topmost element of the stack.

Program :-

Stack.c

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

#define MAX 5

/* global variables */

int stack[MAX] , top = -1;

void push( int value )
{
    if( top == MAX - 1 )
    {
      printf("\n stack overflow error");
      exit(1);
    }

    top++;
    stack[top] = value ;
}
void pop()
{
    if( top == -1 )
       printf("\n stack empty");
    else
    {
       int value = stack[top];
       top--;
       printf("popped value %d",value);
    }
}
int menu()
{
    int choice;

    printf("\n");
    printf("\n 1: push ");
    printf("\n 2: pop ");
    printf("\n 3: exit ");
    printf("\n Enter your choice ");
    scanf("%d",&choice);

    return choice;
}
void main()
{
    int value;
    int choice;

    while( 1 )
    {
       choice = menu();

       switch( choice )
       {
       case 1 :
          printf("Enter a value to push ");
          scanf("%d",&value);
          push( value );
          break;
       case 2 :
          pop();
          break;
       case 3:
          exit(0);
       }
    }
}
```

Output :-

```
1:push
2:pop
3:exit
Enter your choice 1 ↵
Enter value to push 10
1:push
2:pop
3:exit
Enter your choice 1 ↵
Enter value to push 20
1:push
2:pop
3:exit
Enter your choice 2 ↵
popped value 20
1:push
2:pop
3:exit
Enter your choice 2 ↵
popped value 10
1:push
2:pop
3:exit
Enter your choice 2 ↵
stack empty
1:push
2:pop
3:exit
Enter your choice 3 ↵
```

next stack and queue part 2

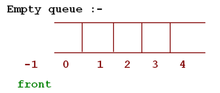
Queue :-

Queue is a linear datastructure that follows first in first out(FIFO) technique. i.e. first inserted element will be the first to come out. Technical name to insert an element into the queue is referred as enqueue and to delete the element from the queue is referred as dequeue operation.

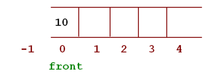
Following diagrams illustrate operations on an integer queue of size 5.

Note: front points to the first element and rear to the last element in the queue.

1) Empty queue :-

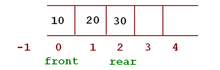


2) insert 10 :-

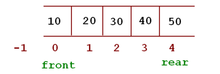


3) insert 20 :-

4) insert 30 :-

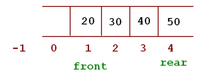


5) insert 40,50 :-

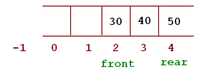


Now queue is considered to be full since `rear == MAX - 1` i.e. `rear == 4`

6) dequeue or delete removes 10 :-



7) next delete operation removes 20:-



**Note: Though there is space in the queue , it is considered to be full since rear == MAX - 1 and still is.

Example program :-

Queue.c

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

#define MAX 5

/* global variables */

int queue[MAX] , front=-1,rear=-1;

void enqueue( int value )
{
   if( rear == MAX - 1 )
   {
     printf("\n queue full ");
     return;
   }

   rear++;
   queue[rear] = value ;
   if( front == -1 )
     front = 0;
}
void dequeue()
{
  if( front == -1 )
     printf("\n queue empty");
  else
  {
     int value = queue[front];

     if( front == rear )
       front = rear = -1;
     else
       front++;

     printf("\n removed value %d", value );
  }
}
int menu()
{
  int choice;

  printf("\n");
  printf("\n 1: enqueue ");
  printf("\n 2: dequeue ");
  printf("\n 3: exit ");
  printf("\n Enter your choice ");
  scanf("%d",&choice);

  return choice;
}
void main()
{
  int value;
  int choice;

  while( 1 )
  {
     choice = menu();
     switch( choice )
     {
     case 1 :
       printf("Enter a value to insert");
       scanf("%d",&value);
       enqueue( value );
       break;
     case 2 :
       dequeue();
       break;
     case 3:
       exit(0);
     }
  }
}
```

output :-

```
  1:enqueue
  2:dequeue
  3:exit
  Enter your choice 1 ↵
  Enter a value to insert 10
  1:enqueue
  2:dequeue
  3:exit
  Enter your choice 1 ↵
  Enter a value to insert 20
  1:enqueue
  2:dequeue
  3:exit
  Enter your choice 2 ↵
  popped value 10
  1:enqueue
  2:dequeue
  3:exit
  Enter your choice 2 ↵
  popped value 20
  1:enqueue
  2:dequeue
  3:exit
  Enter your choice 2 ↵
  queue empty
  1:enqueue
  2:dequeue
  3:exit
  Enter your choice 3 ↵
```