Unit 2

Elements of C

5. Character set:

The set of characters that are used to form words, numbers and expression in C is called C character set. The characters in C are grouped as:

i) Letters / alphabets:

a) Upper case alphabets A, B, C, ... Z

b) Lower case alphabets a, b, c, ... z

ii) Digits:

0, 1, 2, ... 9

iii) Special character:

, → comma
. → period
; → semi-colon
" " → quotation mark
& → ampersand
^ → caret
* → asterisk
< → less than
> → greater than

iv) White space

- blank space
- horizontal tab
- vertical tab
- new line etc.

§. Key words

Keywords are pre defined words for a C-programming language. All keywords have fixed meaning and these meanings cannot be change.

- examples of keywords:

auto double int struct break float
else long switch case char for etc.

§. Identifiers

Words used in C program to identify the names of variables, functions, arrays, pointer and symbolic constants are known as identifiers.

Rules for naming identifiers are:

i) It must be a combination of letters and digits and must begin with a letter or underscore.
ii) Underscore is permitted between two words but space is not allowed.
iii) keywords can't be used as identifiers.
iv) It is case sensitive i.e X and x, Name and name are different.

§. Variables

Variables are memory location in the computer's memory that holds data. Variable hold different values during the execution of a program. Contents of a variable may vary hence the name is given as variable.

Example:

```
f = 1.8 * c + 32
```

Here f and c are variables.

5. Data type

In C, each variable must be declared before it is used. When a variable is declared, a type is assigned to it. The amount of memory reserved by the variable depends upon the type of variable.

There are three classes of data type:

i) Primary (fundamental) data type
ii) User defined data type
iii) Derived data type

6. Primary data type

Primary data types are categorized into five types:

a) Integer type (int)
b) floating point type (float)
c) double precision floating point type (double)
d) Character type (char)
e) Void type (void)

Data type	Category	Memory(byte)	Range	Format specifier
int	short signed	2	-32768 to 32767	%d or %i
	short unsigned	2	0 - 65535	%u
	signed	2	-32768 to 32767	%d
	unsigned	2	0 to 65535	%u
	long signed	4	-2147483648 to 2147483647	%ld
	long unsigned	4	0 to 4294967295	%lu
float	float	4	-	%f
double	double	8	-	%lf
	long double	10		%Lf
char	signed char	1	-128 to 127	%c
	unsigned char	1	0 to 255	%c

8. Constants

Constant is a quantity that doesnot change during the execution of a program.

Types of constants:

i) Numeric constant
- integer constant (E.g 485, 38, -45)
- real constant (24.45, -54.30, 4.1×10^8 etc.)

ii) Character constant
- single character (E.g 'a', '0', '4', 'F')
- String character (E.g "Hello")

(5)

Question:

Tick (✓) if the given identifier is valid and cross(x) with reason of invalid.

i) int
X - It is keyword

ii) x ✓
It is identifier

iii) 1xy X
Because digit is not allow to used. in first.

iv) sum ✓
It is identifier

v) avg no X
Because space is not allowed in identifier

vi) student1 ✓
It is identifier

vii) ?z X
Because symbols shouldn't be used in first.

viii) _name ✓
It is identifier

Variable declaration

A declaration associates a group of variables with a specific data types. All variables must be declared before they can appear in executable statements.

A declaration consists of a data type followed by one as more variable names, ending with a semi-colon.

E.g
```
int a, b, c;
float root1, root2;
int x;
int y;
char flag, text[80];
```

Syntax

Data_type variable_name 1, variable_name2...;

E.g
```
#include <stdio.h>
#include <conio.h>
main()
{
int num;
num=10;
printf("Number = %d", num);
getch();
}
```

Output

Number = 10

Escape sequence

An escape sequence is a non-printing character used in C. It is a character combination consisting of a back slash (\) followed by a letter or combination of digits.

Escape sequence	purpose
\a	Alert or keep sound
\b	backspace
\n	move cursor to new line
\v	vertical tab
\t	horizontal tab
\0	for null character

E.g:

```
#include <stdio.h>
#include <conio.h>
    main( )
{
printf ("Hello ! \n we are testing ");
printf ("Escape sequence \n Thanks! ");
getch ( );
}
```

Output

```
Hello!
we are testing Escape sequence
Thanks!
```

Symbolic constant

Symbolic constant is a name that is used in place of a sequence of characters. The character may represent numeric constant, a character constant or a string constant.

Syntax: #define name value

E.g

#define PI 3.14

#define SIZE 80

Rules for defining a symbolic constant

- Generally they are defined in capital letter.
- No blank space between # and word define.
- A blank space is required between #define and symbolic name and its value.
- After definition, other value can't be assigned to the symbolic name.

Advantages of using symbolic constant

- Modifyability
-

Tokens in C:

The basic elements recognized by the C compiler are called tokens. A token is source program text that the compiler doesn't break down into component elements. keywords, identifiers,

string literals, special symbol : [ ], { }, operators etc are examples of tokens.

Comments:

The comments are given in a program to make it easier to understand by the readers or the developer if the same program is seen after long period of time. It is non executable statement which is skipped by the compiler.

Types of comments:

i) Single line Comment:

E.g: // this is single line comment

ii) Multi-line comment.

E.g: /* comments written in this way are multiple line comments */

Delimiter:

A delimiter is a unique character or series of characters that indicates the beginning or end of a specific statement, string or function body set.

examples:

- round brackets ( )
- curly brackets { }
- escape sequence or comments : \n, //, /*
- double quotes for string literals: " "

5. Statements:

In C program, instructions are written in the form of statements. A statement is an executable part of the program and causes the computer to carry out some action.

Example:

i) `x = 5;` // simple statement

ii) `x = y - z;` // simple statement

iii)
```
{
int l=4, b=2;    //compound statement
int area;
area = l+b;
}
```

Chapter - 4
Operators and Expressions

§. Operators:

An operator is a symbol that operates on a certain data type or data item. Operators are used in programs to perform certain mathematical or logical manipulations.

For example: 8+9

Here symbol + is called an operator

8 and 9 are data items.

The data items that operators act upon are called operands. so, 8 and 9 are operands.

§ Expression:

An expression is a combination of variables, constant and operators written according to the syntax of the language.

For example:

(i) 8+9 (ii) a+b*c (iii) a>b (iv) a*b/3

Classification of operators according to the number of operands:-

i) Unary Operators
ii) Binary Operators
iii) Ternary Operators

i) Unary Operators: The operators which require only one operand are known as unary operators.

For example: increment operator (++), decrement operator (--), Unary minus (-) and unary plus (+).

2> **Binary Operators:** The operators which require two operands are known as binary operators.
For example: plus (+), minus (−), multiply (*), division (/), less than (<) etc.

3> **Ternary Operators:** The operators that require three operands are known as ternary operators.
For example: The operator pair "? :" (conditional operator) is a ternary operator.

5. Types of Operators

1) Arithetic operators
2) Relational Operators
3) Logical or boolean Operators
4) Assignment operators
5) conditional or ternary operator
6) Bitwise Operators
7) Increment /decrement operator
8) comma operator

1> **Arithetic Operators:-** Major arithetic operations are addition, subtraction, multiplication & division. A list of arithetic operators and their meaning are given below:

operator	Meaning	Example (if a = 11, b = 3, c = 2)
+	addition	a+b = 14
−	subtraction	a−b = 8

*	Multiplication	a*c = 22
/	division	a/b = 11/3 = 3
%	modulo division	a%b = 11%3 = 2

Hint: int / int = int
int / float = float
float / int = float
float / float = float

2) Relational Operators: Relational operators are symbol used to test the relationship between two variables or between a variable and a constant

Operators	Meaning	Example (if a=3, b=5, c=3)		Result
==	equal to	a==b	3==5	0
>	greater than	b>a	5>3	1
<	less than	c<b	3<5	1
!=	not equal to	a!=b	3!=5	1
>=	greater than or equal to	c>=a	3>=3	1
<=	less than or equal to	b<=c	5<=3	0

Relational operator check for relation and returns true (1) if the relation is correct and returns false (0) if the relation is incorrect.

3) Logical or boolean Operators: An expression that combines two or more expression is called a logical expression. For combining these expression we use logical operators. These operators, return

0 for flase and 1 for true.

Operators	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

1) AND operator (&&): AND operator return true(1) if and only if all the given conditions are true otherwise false.

Truth table:

condition 1	condition 2	Result (&&)
0 (F)	0 (F)	0 (F)
0 (F)	1 (T)	0 (F)
1 (T)	0 (F)	0 (F)
1 (T)	1 (T)	1 (T)

E.g:

If a = 3, b = 10, c = 4

1) (a<b) && (b==c)

(3<10) && (10==4)

T && F

F (Result = 0)

2) (b>a) && (b>=c)

(10>3) && (10>=4)

T && T

T (Result = 1)

2) OR operator (||) : If all the given conditions are false(0) OR operator return 0 (false), otherwise it return true (1).

Truth table:

Condition 1	Condition 2	Result (11)
0 (F)	0 (F)	0 (F)
0 (F)	1 (T)	1 (T)
1 (T)	0 (F)	1 (T)
1 (T)	1 (T)	1 (T)

E.g:

If a = 6, b = 10, c = 4

i) a < b || b == c

6 < 10 || 10 == 4

T || F

T (1)

(ii) a > b || b < c

6 > 10 || 10 < 4

F || F

F (0)

iii) a = b || b > c

6 = 10 || 10 > 4

F || T

T (1)

NOT operator (!) : It inverts the given condition to get the result that is if input is true (1) output is false (0) and if input is false (0) then output is true (1).

Truth table

Condition	Result
T(1)	F(0)
F(0)	T(1)

E.g:

a = 3, b = 10, c = 4

i) !(a > b || b < c)
!(3 > 10 || 10 < 4)
!(F || F)
!(F)
T(1)

ii) !(a < b || b == c)
!(3 < 10 || 10 == 4)
!(T || F)
!(T)
F(0)

4) Assignment Operator

C treats = as an assignment operator. It is used for the assignment of value in a variable.

E.g:

i) a = 2;
ii) b = a * 2 + a;
iii) i = j - 5

Some other assigment operators.

Operators	Expression	Equivalent expression
+=	i+=5	i = i+5
-=	f-=9	f = f-9
*=	j*(i-3)	j = j*(i-3)
/=	f/=3	f = f/3
%=	i%=j-2	i = i%(j-2)

Note: = and == are different
= is an assignment operator
== is equality operator

If a=3, b=5, c=2 then calculate the result of the following assignment operator.

i) x += a
x = x + a
= x + 3

ii) b -= 5
b = b-5
= 5-5
= 0

iii) a *= (b*c)
a = a*(b*c)
= 3*(5*2)
= 3*10
= 30

iv) b /= 5
b = b/5
= 5/5
= 1

v) c% = b+2
c = c%(b+2)
= 2%(5+2)
= 2%7
= 2

5] Conditional or ternary Operator :-

Conditional expression is represented by ? and : symbols. It is called ternary operator because it operates upon three values rather one or two.

Form of conditional expression:

e1 ? e2 : e3

If the value of e1 is true then the value returned is that of- e2. If the value of e1 is false then value returned is that of e3.

Program

```
#include<stdio.h>
#include<conio.h>
  main( )
{
int i, j ;
printf ("Enter the value of i: ");
scanf ("%d", &i);
 j = (i>4) ? 200 : 100 ;
printf ("\n The value of j =%d ", j);
getch ( );
}
```

output 1

Enter the value of i : 6
The value of j = 200

output 2

Enter the value of i : 2
The value of j = 100

⑧ Bitwise Operators

Bitwise operators are used for manipulating data at bit level. These operators are applied to integer type operands only.

There are three types of bitwise operators

1) Bitwise logical operator
2) Bitwise shift Operator
3) Ones complement Operator

1) **Bitwise logical Operator:** There are three logical bitwise operator. They are:

a) Bitwise AND (&)
b) Bitwise OR (|)
c) Bitwise Exclusive OR (XOR) (^)

a) **Bitwise AND (&):** It performs logical bitwise ANDing between two operands.

n_1 = 60 = 0000 0000 0011 1100
n_2 = 15 = 0000 0000 0000 1111
$n_1 \& n_2$ = 0000 0000 0000 1100 (12)

b) **Bitwise OR (|):** It performs logical bitwise ORing between two operands.

n_1 = 60 = 0000 0000 0011 1100
n_2 = 15 = 0000 0000 0000 1111
$n_1 | n_2$ = 0000 0000 0011 1111 (63)

c) Bitwise exclusive OR (XOR) (^) : The result of exclusive ORing operation is only if both the bits of operands are different.

n_1 = 60 = 0000 0000 0011 1100
n_2 = 15 = 0000 0000 0000 1111
$n_1 \wedge n_2$ = 0000 0000 0011 0011 (51)

2) Bitwise shift Operator : Bitwise shift operator is used to more bit patterns either to the left or to the right. There are two bitwise shift operators.

a) Left shift (<<)
b) Right shift (>>)

a) Left shift (<<) :

syntax : Operand << n

The bits in the Operands are shifted to the left by n positions.

If n_1 = 60 ;

$n_2 = n_1 << 3$;

n_1 = 0000 0000 0011 1100
shift1 = 0000 0000 0111 1000
shift2 = 0000 0000 1111 0000
shift3 = 0000 0001 1110 0000

b) Right shift (>>) :

syntax : operand >> n

The bit in the operand are shifted to the right by n positions.

If $n_1 = 60$

$n_2 = n >> 3$;

n_1 = 0000 0000 0011 1100

shift1 = 0000 0000 0001 1110

shift2 = 0000 0000 0000 1111

shift3 = 0000 0000 0000 0111

3) Bitwise One's complement (~) : It is unary operator that inverts all the bits of operand.

If $n_1 = 60$

$n_2 = \sim n_1$

n_1 = 0000 0000 0011 1100

$n_2 = \sim n_1$ = 1111 1111 1100 0011

7) Imp Increment and decrement Operator

C has two useful operators increment (++) and decrement (--). These are unary operators. The increment operator (++) increments the value of the variable by 1 and decrement operator (--) decrement the variable value by 1.

```
++x     // x = x + 1
--x     // x = x - 1
```

These operators are of two types:

i) Prefix increment / decrement

ii) Postfix increment / decrement

i) Prefix Increment / decrement : Here first the value of variable is incremented / decremented then the new value is used in the operation.

```
#include <stdio.h>
#include <conio.h>
 void main ( )
{
  int x = 8;
printf ("x = %d \t ", x);
printf ("x = %d \t ", ++x);
printf ("x = %d \t ", x );
printf ("x = %d \t ", --x);
printf ("x = %d ", x );
getch ( );
}
```

output

x = 8 x = 9 x = 9 x = 8 x = 8

ii) Postfix increment / decrement : Here first the value of the variable is used or assigned in the operation and then increment or decrement is performed

if x = 3

y = x+1

Here, assignment is done at first and then the value is incremented or decremented,

i.e y = x;

x = 3;

and $x = x + 1$

$= 4$

Program:

```
#include<stdio.h>
#include <conio.h>
 void main ( )
{
 int x = 8;
printf("x = %d \t ", x );
printf(" x = %d \t ", x++);
printf (" x = %d \t ", x );
printf(" x = %d \t ", x--);
printf (" x = %d ", x);
getch ( );
}
```

output

x = 8 x = 8 x = 9 x = 9 x = 8

8) Comma Operator

The comma operator is used to premit different expression to appear in situations where only one expression would be use. A chain of comma in an expression is evaluated from left to right and the value of right most expression is the value of combine expression.

Example

```
v = (x=10, y=5, x+y);
```

first 10 is assigned to x
and 5 is assigned to y
finally (10+5) 15 is assigned to v.

Program

```
#include <stdio.h>
#include <conio.h>
main()
{
int a, b, c, sum;
sum = (a=8, b=7, c=9, a+b+c);
printf ("sum = %d ", sum);
getch();
}
```

Output

sum = 24

Input and Output

Conversion specification (type specification):

While data is being output or input, it must be notified with some identifier and their format specier. Format specifiers are the characters starting with % sign and followed with a character. It specifies the type of data that is being processed.

Data type	format specifier symbol
integer	%d
unsigned integer	%u
octal	%o
hexadecimal	%x
float (simple)	%f
float (exponential)	%e
character	%c
string	%s

Input/Output operation:

Reading input function scanf():

This is the function which is used to send the data into the memory, to provide the data into the memory we use keyboard. With the help of scanf() library function we can enter any type of data into the memory.

Syntax:

```
scanf("specifier", &var1, &var2, ...);
```

Example 1:

```
#include <stdio.h>
#include <conio.h>
void main( )
{
int salary;
scanf ("%d ", &salary);
getch( );
}
```

Example 2:

```
#include <stdio.h>
#include <conio.h>
void main( )
{
char ch;
scanf("%c", &ch);
getch( );
}
```

Writing output function printf( ):

This is the function which is used to send the data into the computer screen from memory. The C provides printf( ) library function for output of data from the memory.

Syntax:

i) printf("specifiers", var1, var2, ...);

ii) printf("printing statement");

Ex.1:

```
#include <stdio.h>
#include <conio.h>
void main()
{
printf("Welcome to Ambikeshwari Campus");
getch();
}
```

Ex.2:

```
#include<stdio.h>
#include<conio.h>
void main
{
int a=10, b=20;
printf("a=%d \t b=%d", a, b);
getch();
}
```

Output

Welcome to Ambikeshwari Campus

Output

a=10 b=20

Unit -1

Problem solving with computer

Problem analysis:

The way of analysing the problem to get better solution is problem analysis. It is important in our daily life activities to get proper solution for different problems. While we are writing computer programs we have to analyse the problem, choose the best option and solve the problems. If proper analysis is done, solution comes fast and is accurate but if analysis is deviated the solution may not come at all.

Algorithm

An algorithm is defined as a finite sequence of explicit instructions that produces an output with the set of input values. The steps of algorithm are never ambiguous. It terminates after finite number of steps. Algorithm are not computer programs. They can't be executed by the computer.

Program 1:

write an algorithm to make a telephone call.

step1: Read or remember the telephone number.
step2: Lift the receiver
step3: Is there a dial tone?
If yes, then dial telephone.
go to step 4.

If no, then put the receiver.
go to step 2.
step4: speak
step5: put down the receiver.

Program 2:
Write an algorithm to find the largest number among three input numbers.

step1: start
step2: Read 3 numbers say: A, B, C
step3: Find the largest number between A and B and store it in Max.
step4: Find the largest number between Max and C and store it in Max.
step5: Display Max
step6: Stop

Program 3:
Write an algorithm to find the smallest number among three input number.

step1: start
step2: Read 3 numbers say: A, B, C
step3: Find the smallest number between A and B and store it in Mini.
step4: Find the smallest number between Min and C and store it in Mini.
step5: Display Mini.
step6: stop

Program 4:
Write an algorithm to calculate the interest of principle, time and Rate are given by User.

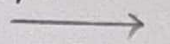
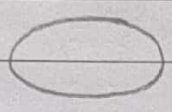
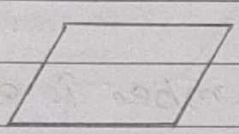
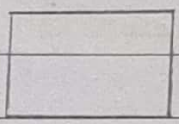
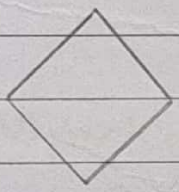
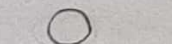
step1 : start
step2 : Read Principle, Time, Rate
step3 : Multiply principle, Time, Rate
step4 : Divide by 100
step5 : Write the answer
step6 : Any more Calculation ?
If yes, then go to step2.
step7 : stop

Program 5:
Write an algorithm to find the area and circumference of the circle.

step1 : start
step2 : Read π, r
step3 : Calculate Area = $\pi * r * r$
step4 : Display Area
step5 : Calculate Circumference = $2 * \pi * r$
step6 : Display Circumference
step7 : stop

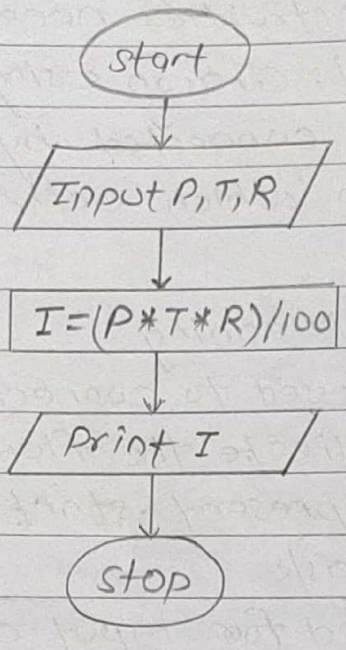
Flowchart

Flowchart is a diagram showing the sequence of events that describes the activities necessary in the solution to a problem. It is drawn using the set of symbols. The symbols are connected by arrow heads that indicates the order in which activities will occur.

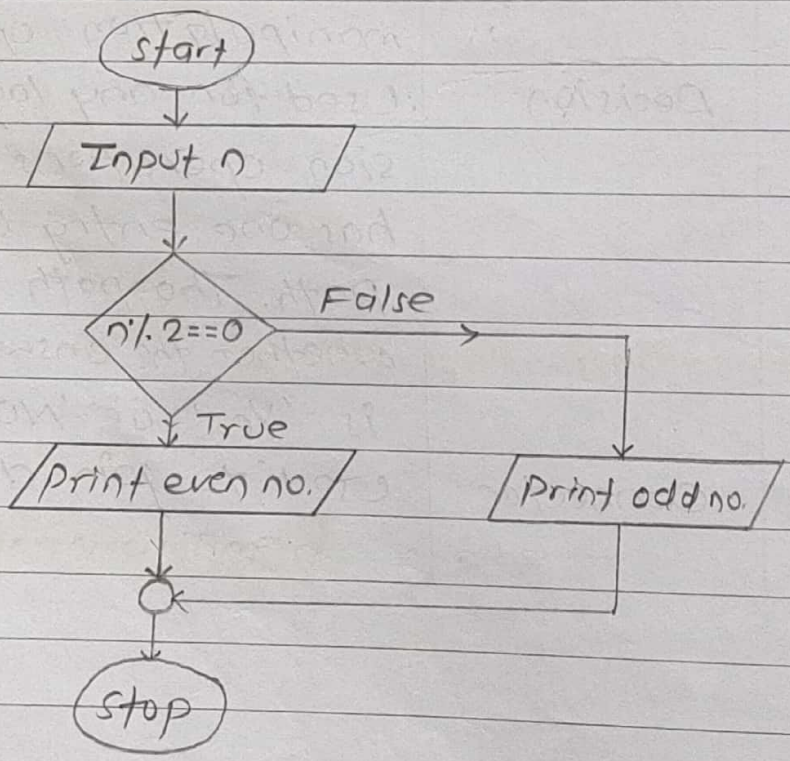
Symbols	Name	Meaning
	Flowline	used to connect symbols and indicate the flow of logic
	Terminal	Represent start or end of task
	Input/Output	used for input and output operations
	Processing	Used for arithmetic and data manipulation operations
	Decision	Used for any logic and comparision operations. Decision symbol has one entry and two exit path. The path chosen depends on whether the answer to the question is 'Yes' or 'NO'.
	Connector	Used to join different flowlines.

Program:

1) Flowchart to calculate simple interest

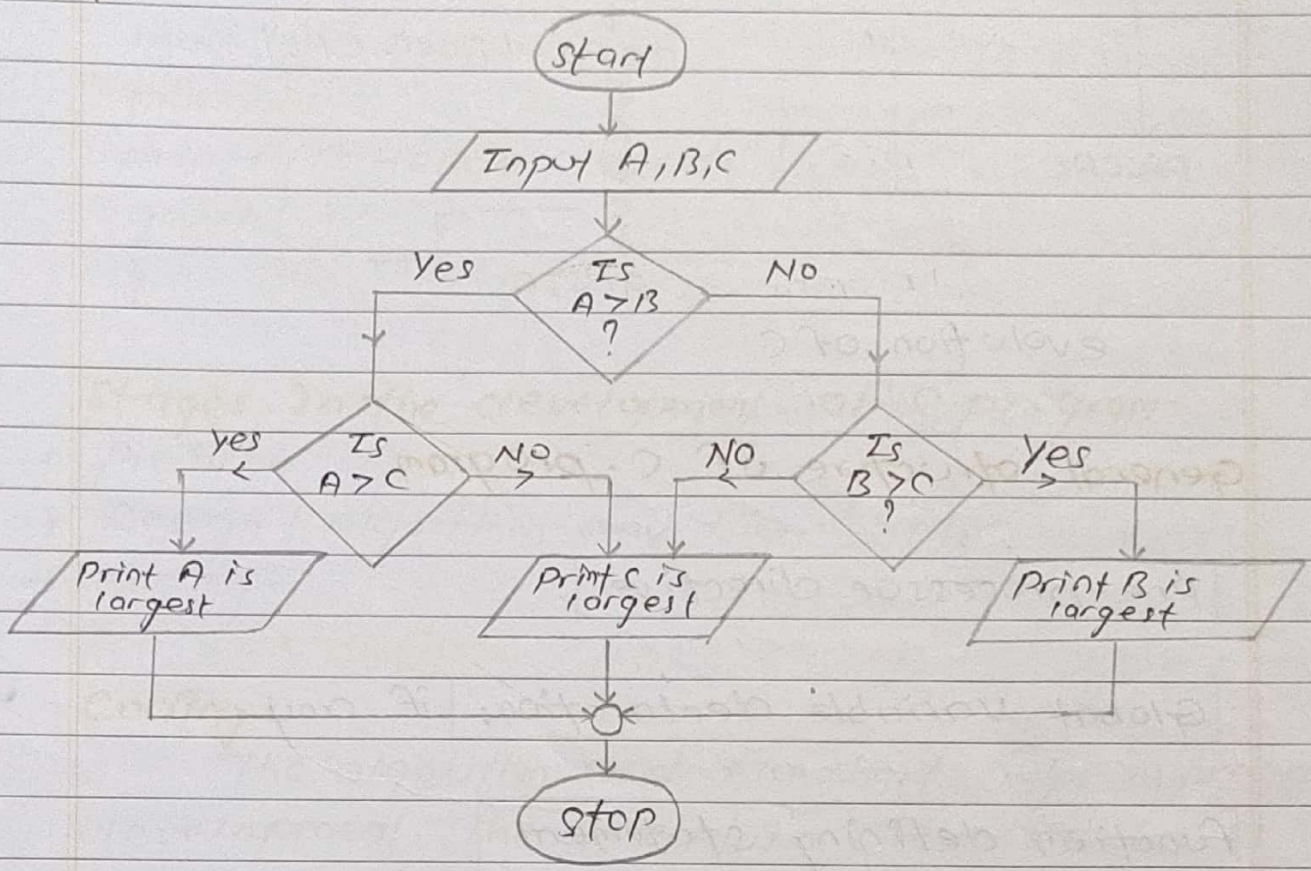


2) Flowchart to determine if the input number is even or odd.



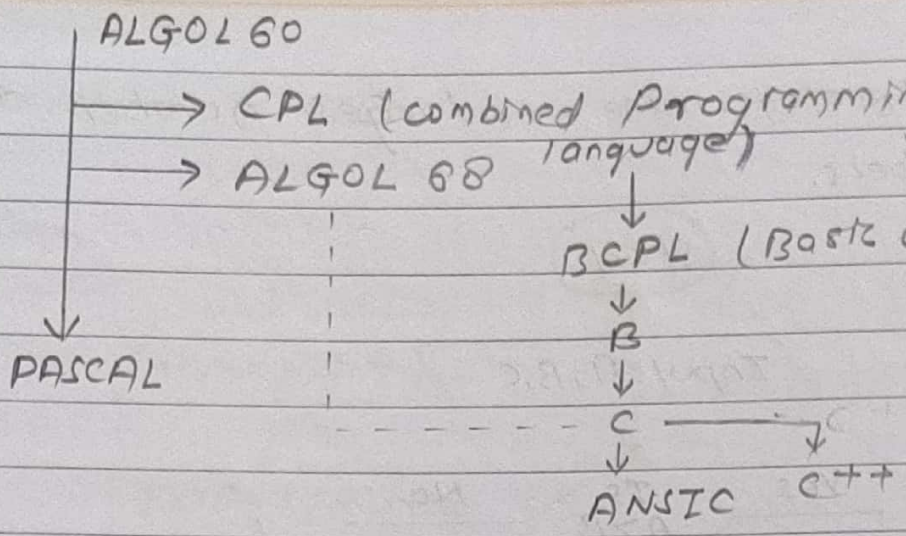
Program:

Flowchart to determine the largest number among the three numbers.



Historical development of C:

C language was developed At AT and T Bell lab of USA in early 1972 by a system programmer Dennis Ritchie. It was written originally for programming under an operating system called UNIX, which itself was later written almost entirely in C.



evolution of C

General structure of C-program

Pre-processor directive

Global variable declaration;	if any

function defining statement
{
local variables;
executable statements
}

Each program must have a function named main() from where execution begins.

Example:

```
#include <stdio.h>
#include <conio.h>
main()
{
printf("Hello students");
getch();
}
```

Stages in the development of C program:

→ Problem Analysis
→ Design (Algorithm and flowchart)
→ Coding

Coding:

The algorithm and flowchart helps in program development. The concept is converted by a programmer into a program which can be executed to give certain result.

Compilation and Execution:

After a program is developed the next step is to translated the program into machine language. This is done using a 'C' compiler. The preprocessor processed the source code before it is passed to the compiler for compilation. The preprocessor give the expanded source code. If there is no error in our source code it is compiled by

compiler and produces the object code.

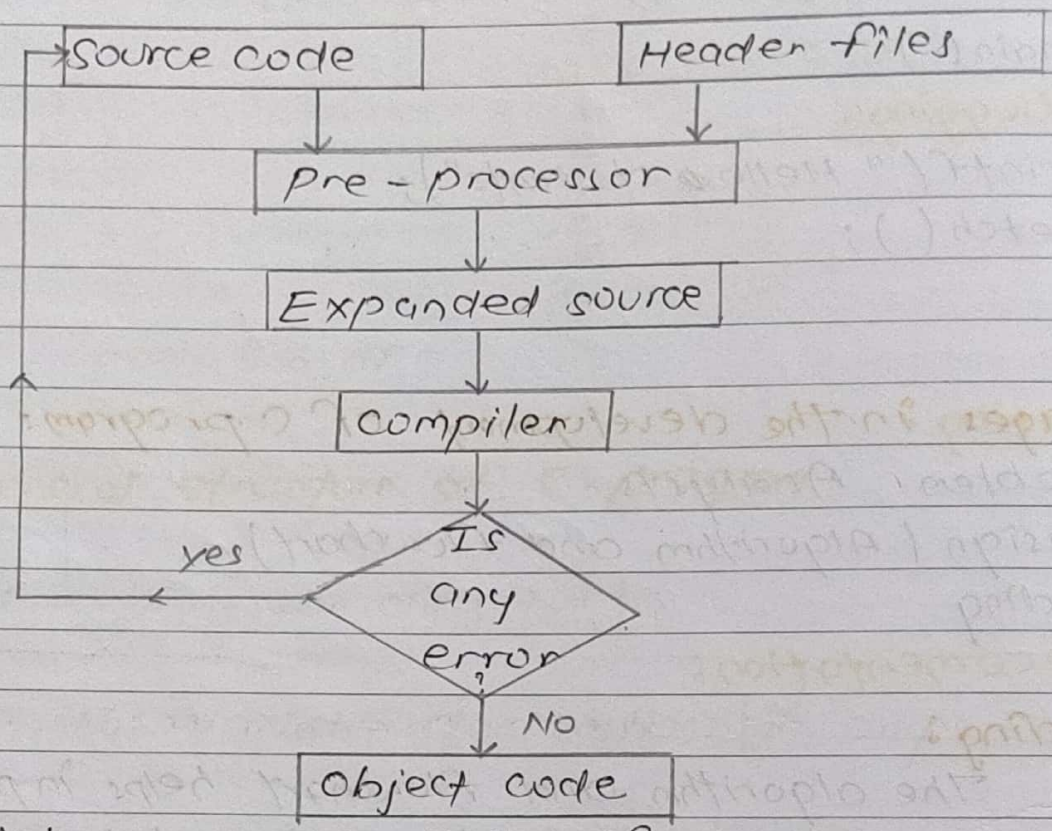


Fig: stages in Compilation of a C program

Linking the program

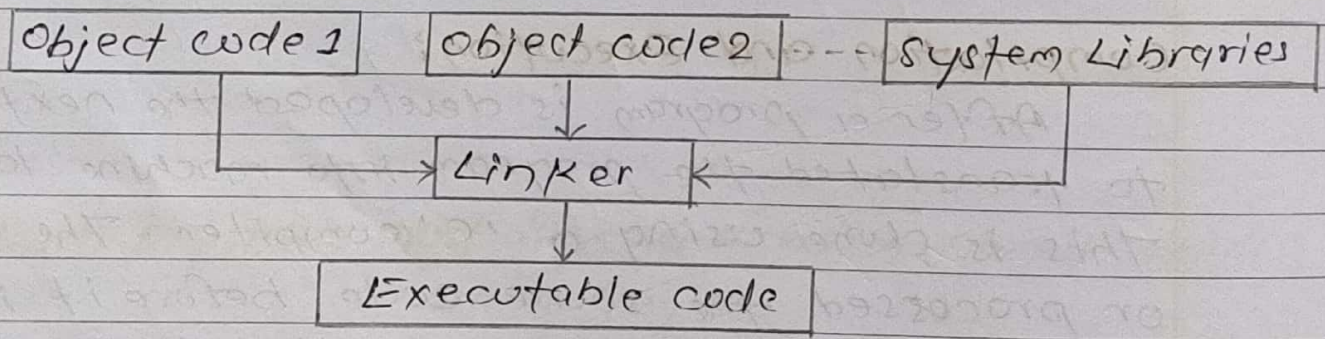


Fig: Linking of object codes with system libraries.

In order to produce an executable code also known as run files, the object codes are to be linked together and also with the system library.

Debugging:

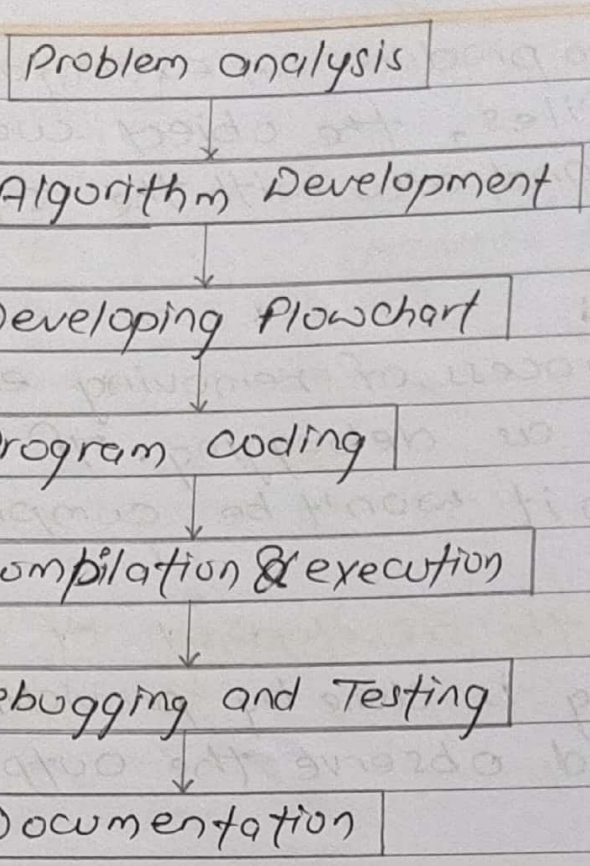
The process of removing error in our program is known as debugging. If our program has any error then it won't be compile.

Testing:

Testing is done by providing expected input values and observe the output.

Documentation:

Here, we prepare the document of the overall program which includes algorithm, flowchart, source code, operation mannual etc.



Imp
Fig: Steps in problem solving by a computer

Control statement:

The program written by us upto now are executed in a sequence as they appeared. But sometimes we may transfer the control to the different part of the program which is possible with the help of control structure i.e decision and iteration. (looping)

Control structure are used to specify the order of sequence into the different parts of the program which determines the flow of control.

There are three types of control statement. They are:

i) Sequential statement
ii) Decision or selection statement
iii) Looping statement.

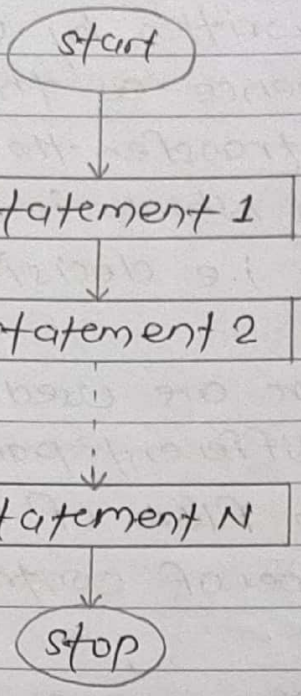
i) Sequential statement:

In sequential structure, the program codes or statement are executed one after another serially. There is no condition and looping.

Syntax:

```
instruction 1;
--------2;
--------
--------
instruction N;
```

flowchart:



ii) Decision or selection or branching statement:

In this type of structure, different sections of program are executed on the basis of given condition in term of "True" or "False" the decision is made how the program flow or control will jump. Various control statements for branching operations are:

a) if statement
b) if - else statement
c) nested if statement
d) if - else if ladder
e) switch - case statement
f) break statement

Syntax:

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

flowchart:

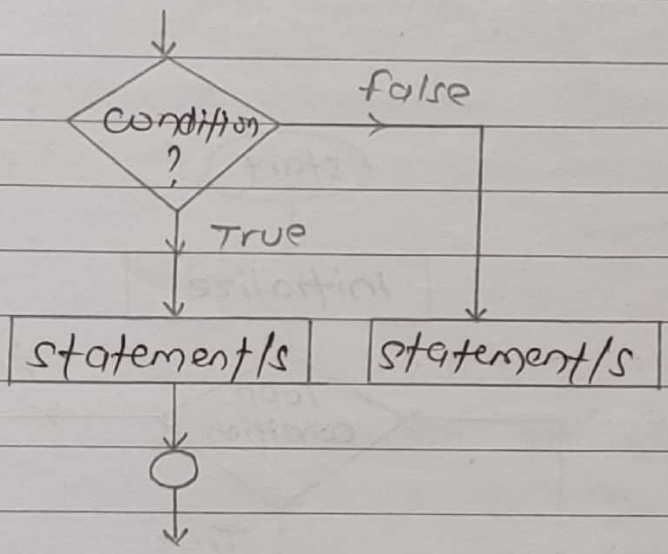


Fig: Flowchart of conditional statement

iii) Looping or iteration structure:

In this control structure block of program statement are executed upto a fixed number of times or until a condition is satisfied. C provides three types of looping statements

1) while statement

ii) do-while statement
iii) for statement

Syntax:

```
initialization
loop condition
{
statement/s;
increment/decrement
.....
.....
}
```

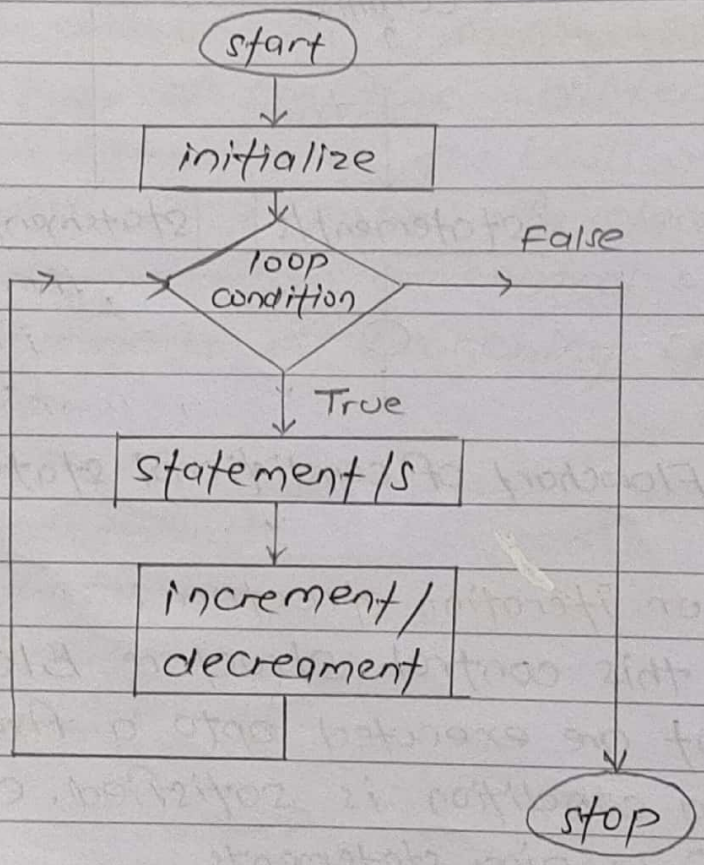


Fig: Flowchart of loop statement

a) if-statement

The if-statement is used to evaluate the conditional expression.

syntax1:

```
// single statement
if (condition)
statement;
```

syntax2:

```
// multiple statement
if (condition)
{
statement 1;
statement 2;
- - - - - - -
- - - - - - -
statement n;
}
```

Program:

WAP to find out commission if sales >= 8000. commission rate is 8%.

```
#include <stdio.h>
#include <conio.h>
void main()
{
float sales, com;
```

```
printf("Enter amount of sales");
scanf("%f", &sales);
if (sales >= 8000)
  com = 0.08 * sales;
printf("commission = 0.2%f", com);
getch( );
}
```

b) if ... else statement:

This statement is called bidirectional statement because when the condition is true then the control is transfer to one part of the program and when the condition is false then the control is transfer to another part of the program.

Syntax:

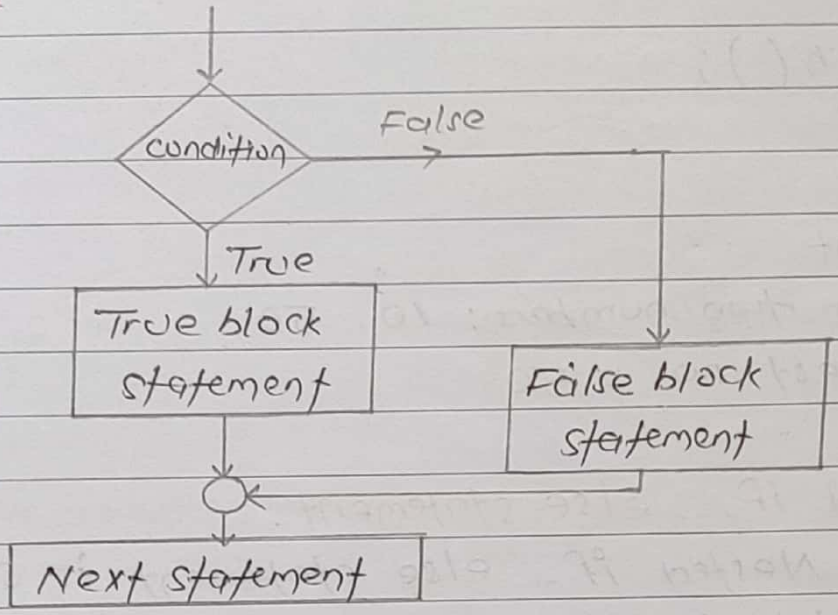
```
//for single line
if (condition)
statement 1;
else
statement 2;
```

Syntax:

```
//for multiline
if (condition)
{
statement;
- - - - -
- - - - - -
}
```

```
else
{
 statement ;
- - - - - -
- - - - - -
}
```

flowchart



Program:

WAP to find largest among two numbers.

```
 /* To find largest among two numbers */
#include <stdio.h>
#include <conio.h>
void main( )
{
int a, b;
printf("Enter two numbers:");
scanf("%d %d", &a, &b);
```

```
if (a>b)
{
printf("largest no. = %d", a);
}
else
{
printf("largest no. = %d", b);
}
getch();
}
```

Output

```
Enter two numbers: 10  50
largest no. = 50
```

c) nested if... else statement

Nested if... else statement is such type construct in which there is an if... else statement within another if... else statement.

Syntax:

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

```
}
else
{
    if (condition 3)
    {
    statement/s;
    }
    else
    {
    statement/s;
    }
}
```

Program

WAP to find the largest no. from 3 given numbers

```
#include <stdio.h>
#include <conio.h>
void main( )
{
int a, b, c, large;
printf("Enter 3 no.s : ");
scanf("%d %d %d", &a, &b, &c);
if (a>b)   (20>30) false
{
    if (a>c)
    {
    large = a;
    }
   else
    {
    large = c;
    }
```

```
}
else
{
    if (b>c)    (30>40) false
    {
     large=b;
    }
    else
    {
     large=c;
    }
}
 printf("\n Largest number = %d ", large );
 getch( );
}
```

Output1

```
Enter 3 no.s: 15  17  12
Largest number = 17
```

Output2

```
Enter 3 no.s: 20  30  40
Largest number = 40
```

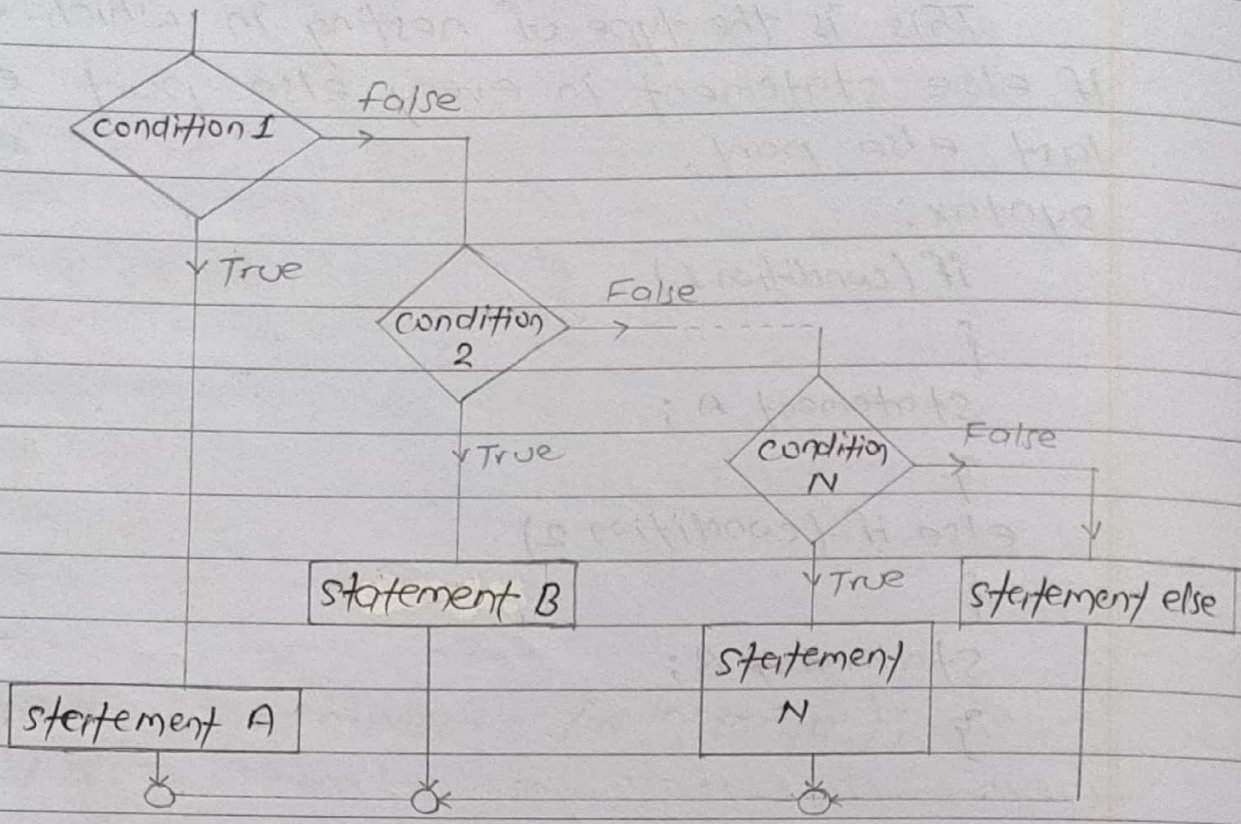
d) else...if ladder:

This is the type of nesting in which there is an if else statement in every else part except the last else part.

syntax:

```
if (condition 1)
{
statement A;
}
else if (condition 2)
{
statement B;
}
------
------
else if(condition 3)
{
statement N;
}
else
{
statement else;
}
```

flowchart



Program

Write a program to find out the grade of a student when the marks of five subject are given. The range of grade are

per >= 85	grade = a
per < 85 and per >= 70	grade = b
per < 70 and per >= 55	grade = c
per < 50 and per >= 40	grade = d

```
#include<stdio.h>
#include<conio.h>
 main( )
{
float m1, m2, m3, m4, m5, total, per;
Char grade;
printf("Enter marks of 5subjects");
scanf("%f %f %f %f %f", &m1, &m2, &m3, &m4, &m5);
total = m1+m2+m3+m4+m5;
if (m1>=35 && m2>=35 && m3>=35 && m4>=35
  && m5>=35)
{
printf("student is pass");
per = total/5;
if (per>=85)
 grade = 'a';
else if (per>=70)
  grade = 'b';
else if (per>=55)
 grade = 'c'
else if (per>=40)
 grade = 'd';
else
 printf("No grade");
}
else
{
printf("student is fail");
}
```

```
printf("\n Grade is : %c ", grade.);
getch( );
}
```

output

```
Enter marks of 5 subjects 90 50 60 70 80
student is pass
Grade is C
```

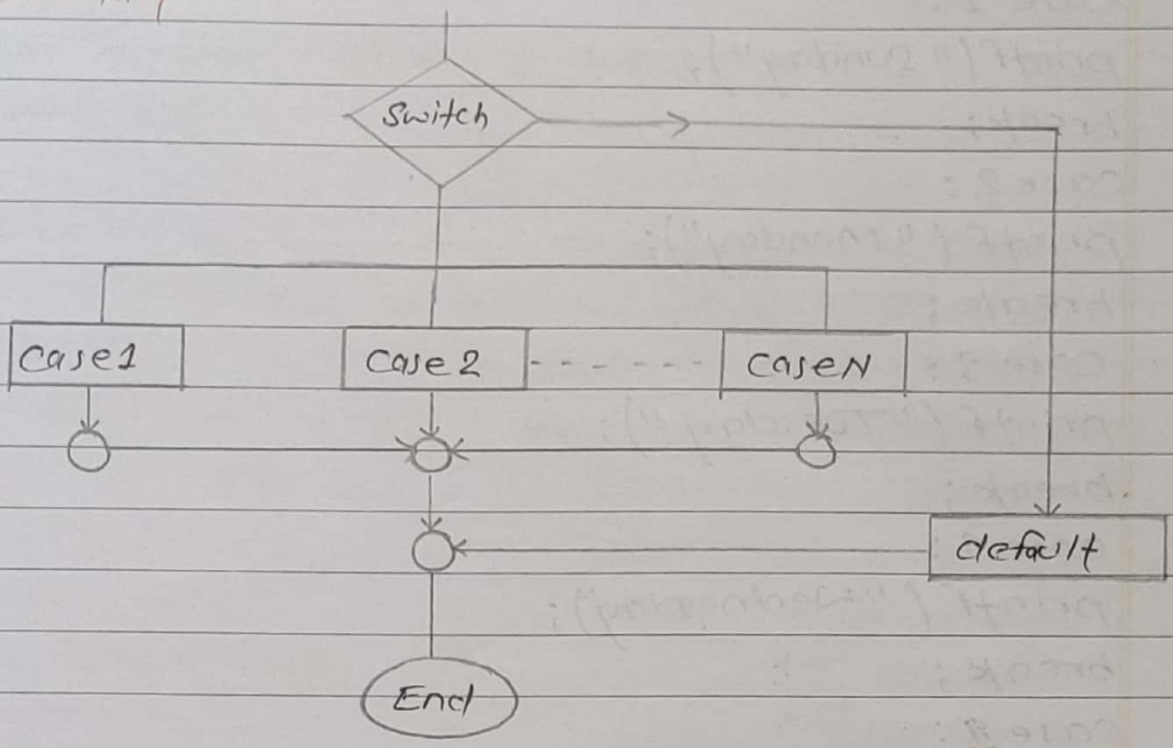
e) switch...case statement:

This is a multi-directional conditional control statement. Sometimes there is need to make choice among number of alternative. For making this choice, we use the switch statement.

syntax:

```
switch (expression)
{
case constant1:
statement1;
break;
case constant2:
statement 2;
break;
case constant N:
statement N;
break;
default:
statement;
}
```

flowchart



Program

1) Write a program to display days of a week according to the number given by user. Hint. 1 = sunday 2 = monday ---- 7 = saturday.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
 int day;
printf("Enter number between 1 and 7: ");
scanf("%d", &day);
switch (day)
{
```

```
case 1:
printf("sunday");
break;
case 2:
printf("Monday");
break;
case 3:
printf("Tuesday");
break;
case 4:
printf("wednesday");
break;
case 5:
printf("Thursday");
break;
case 6:
printf("Friday");
break;
case 7:
printf("saturday");
break;
default:
printf("wrong input");
}
getch();
}
```

Output :

Enter number between 1 and 7 : 4
Wednesday

Loop :-

Loop may be defined as a block of statements which are repeatedly executed for a certain number of times or until a particular condition is satisfied.

Types of loop

i) For loop
ii) While loop
iii) do - while loop

i) For loop : The for loop is useful to execute a statement for a number of times. When the number of repetition is known as advance, the use of this loop will be more efficient.

Syntax

```
for (counter initialization ; test condition ; increament/decreament)
{
 statement/s ;
}
```

Flowchart

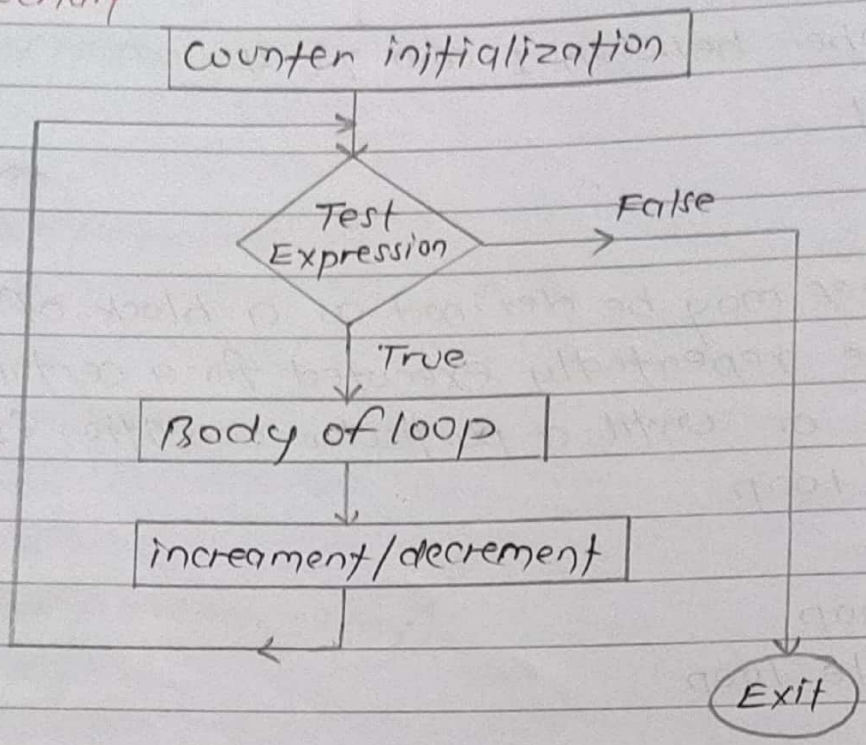


Fig: Flowchart of for loop

1) WAP to print Hello world 1000 times by using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
for(i=1; i<=1000; i++)
{
print("Hello world",i);
}
getch();
}
```

WAP to print the series:

i) 1 2 3 4 ---- 25

ii) 50 49 48 47 ---- 1

i)
```
#include<stdio.h>
#include<conio.h>
void main( )
{
 int i ;
for ( i = 1 ; i <= 25 ; i++)
 {
printf ( " %d \t ", i);
 }
getch ( );
}
```

ii)
```
#include<stdio.h>
#include<conio.h>
{void main( )
int i ;
for ( i = 50 ; i >= 1 ; i--)
{
printf ( " %d \t ", i);
}
getch ( );
}
```

iii) 2 4 6 ... 100

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

iv) 50 45 40 ... 0

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i;
for (i=50; i>=0; i-=5)
{
printf("%d \t", i);
}
getch();
}
```

v) 1 4 7 10 upto 10th term

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i, a;
  a=1;
for (i=1; i<=10; i++)
{
 printf("%d \t", a);
 a=a+3;
}
 getch();
}
```

vi) 5 10 15 20 upto 30th term

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i, a;
   a=5
for (i=1; i<=30; i++)
{
printf("%d \t", a);
 a=a+5;
}
getch();
}
```

viii) 1 4 9 upto 10th term

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, i;
a = 1
for (i=1; i<=10; i++)
{
printf("%d \t", a);
a = a * a;
}
getch();
}
```

OR

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
for (i=1; i<=10; i++)
{
printf("%d \t", i*i);
}
getch();
}
```

viii) 1 8 27 ... upto 10th term

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
for(i=1; i<=10; i++)
{
printf("%d \t", i*i*i);
}
getch();
}
```

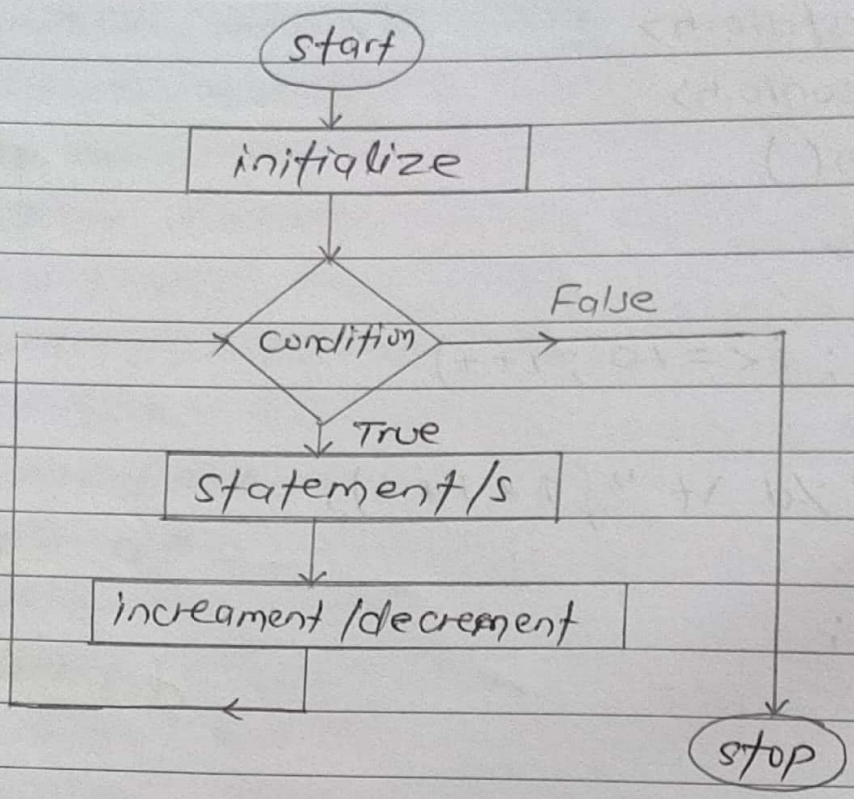
While loop:

The while loop keep repeting an action until an associated condition returns false. This is useful when the programmer doesn of know in advance how manytimes the loop will be traversed.

Syntax:

```
while (test condition)
{
statement/s;
increment/decrement;
}
```

flowchart



Program:

1 2 3 30

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
 int i ;
 i=1
while (i<=30).
{
printf( "%d \t", i);
 i = i+1;
}
getch( );
}
```

Program

i) 2 4 6 40

ii) 50 45 40 5

iii) 1 4 9 100

i) 2 4 6 40

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
i=2;
while(i<=40)
{
printf("%d\t",i);
i=i+2;
}
getch();
}
```

ii) 50 45 40 5

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
i=50;
while(i>=5)
```

```
{
printf("%d \t", i);
  i = i - 5;
 }
getch();
}
```

iii) 1 4 9 100

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
 i=1;
while(i<=10)
{
printf("%d \t", i*i);
 i = i+1;
}
getch();
}
```

Write a program to print the even number from 1 to 100.

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

```
int i;
 i=2
while (i<=100)
{
printf("%d \t", i);
 i=i+2;
}
getch();
}
```

Output

2 4 6 8 100

WAP to check if the number between 1 to 100 is even or not and display only the even number.

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

Output

2 4 6 8 100

WAP to display first 50 natural numbers and their sum.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int i, sum;
 i=1 ; sum=0;
while (i<=50)
{
printf("%d \t", i);
 sum = sum+ i;
   i = i+1;
}
printf("\n sum = %d ", sum);
getch( );
}
```

Output

1 2 3 4 5 50
sum = 1275

do...while loop:

The do...while loop is a post test loop structure as the condition is checked after the loop body is executed. This ensure that the loop body is executed at least once if the condition is false.

Syntax:

```
do
{
statement/s;
}
while (test condition);
```

Flowchart:

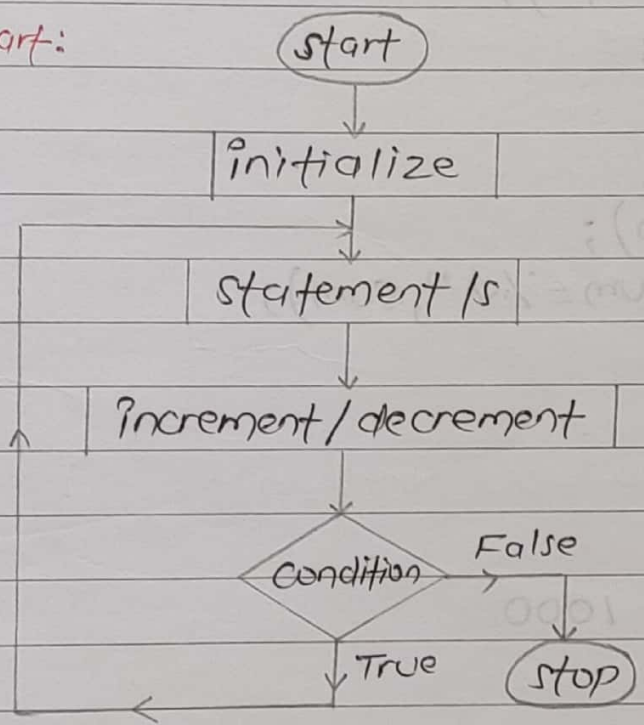


Fig: Flowchart of do.... while loop

Program

Print the given series with sum using do... while loop

i) 5 10 15 20 50

```
#include<stdio.h>
#include<conio.h>
void main( )
{
 int i, sum;
 i= 5 , sum = 0;
do
{
printf("%d \t", i);
 sum = sum+i;
 i= i+5;
}
while (i <=50);
printf ("\n sum = %d", sum);
getch ( );
}
```

ii) 1 8 27 1000

```
#
#
void main( )
{
int i , sum;
i=1 , sum=0;
```

Flair

```
do
{
printf ("%d \t ", i);
sum = sum + i ;
i = i * i * i;
}
while (i <= 1000)
printf ("\n sum = %d", sum);
getch ( );
}
```

Break statement:

The break statement is used inside loops and switch statements. In some program if we want to exit from loop before terminating the loop we can use this break statement.

Example:

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
 int n;
 for (n=1; n<=5; n++)
 {
  if (n==3)
  {
  printf ("Break executed");
  break;
  }
  printf ("Number=%d \n", n);
 }
 getch ( );
}
```

output

```
Number=1
Number=2
Break executed
```

Continue statement:

The continue statement is used when we want to go to next iteration of the loop after skipping some statements of the loop.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
for(n=1; n<=5; n++)
 {
 if (n==3)
printf("skip display of 3");
continue;
}
printf("Number = %d \n", n);
}
getch();
}
```

Output

```
Number = 1
Number = 2
skip display of 3
Number = 4
Number = 5
```

Exit statement:

Exit statement is a C library function that can be used to cause a program to end. Exit function is defined in a header file stdlib.h. We have to include the header file at the begining of our program. A non zero arguement to the exit() function tells the operating system that the program has terminated normally.

Example:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
printf("My first program");
 exit(0);
getch();
}
```

Output

My first program

§ Nested loop:

When a body part of a loop contains another loop then the inner loop is said to be nested within the outer loop.

In nested loop, for each value or pass of outer loop, inner loop is completely executed. Thus inner loop

operates fast and outer loop operates slowly.

Syntax:

```
for (counter initialization; condition; increment/decrement)
{
   for (counter initialization; condition; inc/dec)
   {
     statement/s;
   }
}
```

outer loop

inner loop

Program

```
#include<stdio.h>
#include<conio.h>
void main( )
{
  int i, j;
  for (i=1; i<=10; i++)
  {
  for (j=1; j<=10; j++)
    {
    printf("%d * %d = %d \t", i, j, i*j);
    }
  printf("\n");
  }
getch( );
}
```

Output

1 * 1 = 1 1 * 2 = 2 . . . . . 1 * 10 = 10

2 * 1 = 2 2 * 2 = 4 . . . . . 2 * 10 = 20

.

.

.

.

.

.

10 * 1 = 10 10 * 2 = 20 . . . . . 10 * 10 = 100

Print the following pattern using nested loop.

1)
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j;
 for(i=1;i<=5;i++)
 {
  for(j=1;j<=i;j++)
  {
  printf("%d \t",j);
  }
  printf("\n");
 }
getch();
}
```

2)
```
*
* *
* * *
* * * *
* * * * *
```

```
#include<stdio.h>
#include<conio.h>
void main( )
{
  int i, j;
  for (i=1; i<=5; i++)
  {
  for (j=1; j<=i; j++)
  {
   printf ("* \t");
  }
   printf ("\n");
  }
getch( );
}
```

3) 1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j;
 for(i=5;i>=1;i--)
 {
  for(j=1;j<=i;j++)
  {
  printf("%d \t",j);
  }
  printf("\n");
  }
 getch();
}
```

4) 5 4 3 2 1
5 4 3 2
5 4 3
5 4
5

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j;
 for(i=1;i<=5;i++)
 {
    for(j=5;j>=i;j--)
    {
     printf("%d \t",j);
    }
    printf("\n");
 }
getch();
}
```

5)
```
5
5  4
5  4  3
5  4  3  2
5  4  3  2  1
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j;
 for(i=5;i>=1;i--)
 {
```

```
    for (j=5; i>=i; j--)
     {
       printf ("%d \t", j);
     }
     printf ("\n");
   }
   getch ();
}
```

6)
```
1
2  2
3  3  3
4  4  4  4
5  5  5  5  5
```

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
  int i, j;
  for (i=1; i<=5; i++)
  {
    for (j=1; j<=i; j++)
    {
    printf ("%d \t", i);
    }
    printf ("\n");
  }
  getch ();
}
```

7)
```
1
1  1
1  1  1
1  1  1  1
1  1  1  1  1
```

```
#include<stdio.h>
#include<conio.h>
main( )
{
 int i, j;
  for (i=1; i<=5; i++)
  {
     for (j=1; j<=i ; j++)
      {
        printf("1 \t");
      }
     printf("\n");
  }
  getch( );
}
```

Unit-6

ARRAY AND STRING

(80)

Array

In many of the programming situations, we may require to process the data items that have common characteristic. Now in such case, it would be easier if we place this data items into one signal variable called array which is capable of storing number of data sharing common name.

The individual data item can be characters, integers and floating point number. But they must all be of same type. The individual data item in an array are called array elements.

Each array element is refer to by spacing the array name followed by one or more subscript enclosed in square bracket.

For example:

x[n]

where x = array name

n = subscript (size)

Elements x[0], x[1], x[2], ... x[n-1]

Definition: An array can be defined as a group of homogeneous elements sharing a common name.

A one dimensional array can be declared as given

Syntax:

data_type arrayname[size];

where,
data-type is the type of array i.e int, float, char, etc.
name is the valid identifier.
size specifies the number of array elements.
e.g:
```
int x[10];    // 10 integers
char name[30];    // 30 character
float list[35];    // 35 floats
```

Initialize an array

Array can be initialize as other variables during its declaration this means array declaration can include the initial values as per requirements.

syntax:
```
data-type arrayname[size] = { val1, val2, ... valn };
```
e.g: i) `int digit[5] = {1, 2, 3, 4, 5};`
ii) `float salary[4] = {3000, 6000, 7000, 8005.50};`
iii) `char color[3] = {'R', 'G', 'B'};`
iv) `int no[] = {22, 34, 61, 46};`

Assessing array elements:

The elements of array can be access by spacifying the array name followed by subscript in brackets. In c the array subscript start from zero.

e.g:
```
int x[5];    //size of array is 5.
```
elements of this array:

x[0]
x[1]
x[2]
x[3]
x[4]

Processing array elements:

Suppose arr[10] is an array of type int.

i) Reading values in arr[10]

```
for (i=0; i<10; i++)
{
  scanf("%d", &arr[i]);
}
```

ii) Displaying value of arr[10].

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

iii) Adding all elements of arr[10]

```
int sum=0;
for (i=0; i<10; i++)
{
  sum = sum + arr[i]; // adding
}
```

Programs:

Write a program to input value into an array and display them.

```
#include<stdio.h>
#include<conio.h>
main( )
{
  int arr[10], i;
  for (i=0; i<10; i++)
  {
     printf("Enter the value of arr[%d]: ", i);
     scanf("%d", &arr[i]);
  }
  printf("The array element are: \n");
  for (i=0; i<10; i++)
  {
     printf("%d ", arr[i]);
  }
  getch( );
}
```

Output

Enter the value of arr[0] = 60
Enter the value of arr[1] = 76
:
Enter the value of arr[9] = 71
The array elements are
60 76 - - - - 71

Write a program to find the largest and smallest value in array.

```
#include<stdio.h>
#include<conio.h>
main( )
{
    int i, arr[10]={18,45,90,67,23,45,23,34,67,78};
    int max, min;
    min = max = arr[0];
    for (i=0 ; i<10; i++)
    {
       if (arr[i] <min)
       {
          min = arr[i];
       }
       if(arr[i] >max)
       {
          max = arr[i];
       }
    }
printf ("Maximum value = %d ", max);
printf ("\n Minimum value = %d ", min);
getch( );
}
```

Output

Maximum value = 90
Minimum value = 18

Write a program to read salary of n employes and count the number of employes getting salary from 5000 to 1000.

```
#include<stdio.h>
#include<conio.h>
main( )
{
 int i, count=0, n;
 float salary[100];
 printf("Enter no. of employee less than 100");
 scanf("%d", &n);
 for (i=0; i<n; i++)
 {
  scanf("%f", &salary[i]);
 }
 for (i=0; i<n; i++)
 {
 if (salary[i]>=5000 && salary[i]<=1000)
 {
  count++;
 }
 }
 printf("No. of employee counted %d", count);
 getch( );
}
```

Output

Enter no. of employee less than 100 = 5

5000 11000 14000 8000 6000

No. of employee counted 3

§ Multidimensional array:

When we declare array with more than one dimension it is known as multidemensional array. We usually used upto two dimension only. Two dimensional array consists of two subscript in which first gives no. of rows and second gives no. of column size.

Syntax:

```
datatype array_name [row_size][column_size];
```

Eg. `int mat[3][2]; //two dimensional`
`float threeD[3][2][3]; //three dimensional`

1) `mat[3][2] = { {3,2}`
`{1,5}`
`{6,7} };`

	[0]	[1] ↓ columns
[0]	[0][0]	[0][1]
[1]	[1][0]	[1][1]
[2]	[2][0]	[2][1]

→ rows

Program

WAP to input 3x3 matrix and display it.

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

```
int i, j, mat[3][3];
printf("Enter the elements of 3x3 matrix:\n");
for (i=0; i<3; i++)
{
    for (j=0; j<3; j++)
    {
    scanf("%d", &mat[i][j]);
    }
}
printf("Matrix is:\n");
for (i=0; i<3; i++)
{
    for (j=0; j<3; j++)
    {
     printf("%d \t", mat[i][j]);
    }
    printf("\n");
}
 getch();
}
```

output

```
Enter the elements of 3x3 matrix:
 1   3   -4
 0   -2   3
-4   0    2
Matrix is:
 1    3    -4
 0    -2    3
-4    0     2
```

Program

Write a program to read the element of two 3x3 matrix and perform the matrix addition.

```
#include<stdio.h>
#include<conio.h>
main()
{
  int i,j, mat1[3][3], mat2[3][3], sum[3][3];
  printf("Enter the elements of matrix 1:\n");
  for(i=0; i<3; i++)
  {
     for(j=0; j<3; j++)
     {
        scanf("%d", &mat1[i][j]);
     }
  }
  printf("Enter the elements of matrix 2:");
  for(i=0; i<3; i++)
  {
     for(j=0; j<3; j++)
     {
        scanf("%d", &mat2[i][j]);
     }
  }
  //adding mat1 and mat2
  for(i=0; i<3; i++)
  {
     for(j=0; j<3; j++)
```

```
        {
            sum[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
    printf("sum of two matrices : \n");
    for (i=0; i<3; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d", sum[i][j]);
        }
        printf("\n");
    }
    getch();
}
```

Output

```
Enter the elements of matrix 1 :
  1   -4   3
  6    1   0
 -2    1   2
Enter the elem
  2    2   -1
  3    0    1
  1    7   -4
Sum of two matrices
  3   -2    2
  9    1    1
 -1    8   -2
```

Write a program to input elements of two 3*4 matrices A and B. The perform the matrix subtraction

```
#include<stdio.h>
#include<conio.h>
main ( )
{
   int i, j, matA[3][4], matB[3][4], sub[3],[4];
   printf("Enter the element of matrixA: \n");
   for (i=0; i<3; i++)
   {
      for (j=0; j<4; j++)
      {
         scanf("%d", &matA[i][j]);
      }
   }
   printf("Enter the element of matrixB: \n");
   for (i=0; i<3; i++)
   {
      for (j=0; j<4; j++)
      {
         scanf("%d", &matB[i][j]);
      }
   }
//subtracting matA and matB
for (i=0; i<3; i++)
{
   for (j=0; j<4; j++)
```

```
{
    sub[i][j] = matA[i][j] - matB[i][j];
}
printf("subtraction of two matrices: \n");
for (i=0; i<3; i++)
{
    for (j=0; j<4; j++)
    {
        printf("%d", sub[i][j]);
    }
    printf("\n");
}
getch();
}
```

Output

```
Enter the element of matrix A:
1   2   3
5   6   4
3   4   9
Enter the element of matrix B:
4   9   5
6   3   1
7   8   2
subtraction of two matrices:
-3  -7  -2
1   3   3
-4  -4  7
```

5. Character array (string) :

String is a group of character which can be represent by character array. strings are defined within double quotation symbols. The last character of string is marked by NULL ("\0") character.

Example:

C	O	M	P	U	T	E	R

Declaration: char subject[10];

Initialization:

```
char name[10] = {'C','O','M','P','U','T','E','R','\0'};
```

OR

```
char name[10] = {"COMPUTER"};
```

Reading string:

- Using gets() or scanf()
- Eg (i) scanf("%s", name);
  (ii) gets(name);

Writing string:

- Using puts() or printf()
- E.g (i) printf("%s", name);
  (ii) puts(name);

Program

Write a program to input and output using gets() and puts():

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

```
{
    char name[20];
    printf("Enter your name: \n");
    gets(name);
    //scanf("%s", name);
    printf("Your name is: ");
    puts(name);
    //printf("%s", name);
    getch();
}
```

Output

```
Enter your name
  Prakriti
Your name is: Prakriti
```

Array of string:

We can assign the array of string using two dimensional array.

Eg.

```
Char name[5][10] = {"Ram", "Hari", "Sita", "Prakriti",
                    "Rita"};
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
name[0]	R	a	m	\0						
name[1]	H	a	r	i	\0					
name[2]	S	i	t	a	\0					
name[3]	P	r	a	k	r	i	t	i	\0	
name[4]	R	i	t	a	\0					

String functions:

There are various string library functions defined under the header file string.h. some of the common string functions are:

1) strlen(): This function gives the number of characters as an output, excluding the terminating null character.

Syntax:

strlen (string or variable)

```
/* determine length of string */
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    char str[40];
    int len;
    printf("Enter a string");
    scanf("%s", str);
    len = strlen(str);
    printf("string %s has %d character", str, len);
    getch();
}
```

Output

```
Enter a string: kathmandu
string kathmandu has 9 character.
```

2) strcat( ): This function combines two string together to form a single string. Two strings are passed as arguments into this functions.

Syntax:

strcat (destination string, source string)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
  char str1[10], str2[10];
  printf("Enter first string:");
  scanf("%s", str1);
  printf("Enter second string:");
  scanf("%s", str2);
  strcat(str1, str2);
  printf("The concatenated string is %s", str1);
  getch();
  }
```

output

Enter first string: Dang
Enter second string: _Nepal
The concatenated string is: Dang_Nepal

3) strcmp( ): This function compares two strings such that if two strings are equal then this function return 0 else if returns -1 if first string is smaller and 1 if first

string is larger.

syntax:

strcmp(string1, string2)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    int x;
    char str1[10], str2[10];
    printf("Enter first string:");
    scanf("%s", str1);
    printf("Enter second string:");
    scanf("%s", str2);
     x = strcmp(str1, str2);
   if (x == 0)
   printf("strings are equal");
   else if (x == -1)
   printf("string1 < string2");
   else
   printf("string1 > string2");
   getch();
   }
```

Output

```
Enter first string: Dang
Enter second string: Dang
strings are equal.
```

4) strrev( ): This function is used to reverse the given string.
syntax: strrev (string)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main( )
{
 char str[10];
 printf("Enter string:");
 scanf("%s", str);
 strrev(str);
 printf("Reverse string is : %s", str);
 getch( );
}
```

Output
Enter string: Nepal
Reverse string: lapeN

5) strcpy( ): This function is used to copy source string to destination.
Syntax:
strcpy (destination_str, source_str);

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

```
main()
{
    char s1[10], s2[10];
    printf("Enter string:");
    scanf("%s", s1);
    strcpy(s2, s1);
    printf("copied string is: %s", s2);
    getch();
}
```

output

Enter string: Pokhara

copied string is: Pokhara

6> strupr(): This function is used to convert the lower case character into uppercase.

syntax:

strupr(string)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    char s1[10];
    printf("Enter string:");
    scanf("%s", s1);
    strupr(s1);
    printf("uppercase string is: %s", s1);
    getch();
}
```

output
Enter string : PoKHaRA
Uppercase string is: POKHARA

7) strlwr( ): This function is used to convert the upper case character into lowercase.
syntax:
strlwr(string)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    char s1[10];
    printf("Enter string: ");
    scanf("%s", s1);
    strlwr(s1);
    printf("Lower string is: %s", s1);
    getch();
}
```

Output
Enter string: POKHara
Lowercase string is: pokhara

Program

Write a program to input a word and check if it is palindrome or not. (Hint. mom, pop, madam, liril)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
   char word[30], word1[30];
   printf("Input a word:");
   scanf("%s", word);
   strcpy(word1, word);
   if (strcmp(word1, strrev(word))==0)
   {
     printf("%s is palindrome.", word1);
   }
   else
   {
     printf("%s is not palindrome.", word1);
   }
   getch();
}
```

Output

```
Input a word: madam
madam is palindrome.
```

Unit-4

Functions

101

Introduction:

A function is a self contained program segments or sub program that perform some well defined task. we should use main function in every program.

Types of functions:

1) **Library function:** Library functions are present in C library and they are predefined. we can directly use this functions by including the respective header files.
e.g: sqrt( ), strcpy( ), scanf( ), strupr( ), printf( ) etc.

2) **User defined function:** User can create their own functions for performing any specific task. These type of functions created by user are called user defined functions. To create and use this functions we should know three things.

i) function definition
ii) function declaration
iii) function call

1) **Function definition:** It consists of whole description and code of a function. It tells what the function is doing and what are its input and output. It consists of two parts

- function header
- function body

syntax:

return_type, function_name(argument list) ; header

```
{
    local variable ls;
    statement ls;
    - - - - - -
    - - - - - -
    return (expression);
}
```

body

Note:

- return_type: data type of result, void indicates no return values.
- function_name: valid identifier
- argument list: types and number of arguments

E.g:

```
int sum (int a, int b)
{
    int c;
    c = a+b;
    return c;
}
```

ii) Function declaration or function prototype:

It is a declaration statement that specifies the function name, the no. and type of arguments as well as the return type of function.

Syntax:

```
return_type, function_name (argument list);
```

E.g:
```
int product (int a, int b, int c);
float sum (int x, int y);
void name( char name[]);
```

Note:
- function declaration is not required if we define the function before main( ) function.
- It is done after the header files are included.

Function call: When a function is called its code get executed. Function calling means using or invoking a function. Function call a specifies the name followed by list of arguments enclosed in parenthesis and separated by comma.

E.g: a) s= sum (a,b);
b) x = sqrt (81);

Program
WAP to square a given number by using user defined function.

```
#include <stdio.h>
#include <conio.h>
int square (int n);     //function declaration
void main( )
{
  int n, x;
```

```
    printf("Enter a no.: \n");
    scanf("%d", &n);
     m = square(n); //function call
    printf("\n square of %d = %d", n, m);
    getch();
}
 int square(int n)
 {
    int sq  //local variable
    sq = n * n;
    return sq;
 }
```

output

Enter a no:

5

square of 5 = 25

Program

Write a program to find sum of two numbers by using user defined functions.

```
#include<stdio.h>
#include<conio.h>
int sum(int a, int b);
void main()
{
 int a, b, s;
 printf("Enter two numbers: \n");
```

```
scanf("%d %d", &a, &b);
s = sum(a,b);
printf("\n sum of these no. s = %d", s);
getch();
}
int sum(int x, int y)
{
   return (x+y);
}
```

output

Enter two numbers:

15 7

sum of these no. s = 22

Program:

WAP to find sum of an integer number using function.

```
#include<stdio.h>
#include<conio.h>
int sum(int n);   //function declaration
void main()
{
   int n,x;
   printf("Enter how many numbers:\n");
   scanf("%d", &n);
   x = sum(n);      //function call
   printf("\n sum of %d number = %d", n, x);
   getch();
}
```

```
int sum (int n)
{
  int i, num, s=0;
  // local variables
  for (i=1; i<=n; i++)
  {
     printf("Enter number:");
     scanf("%d", &num);
     s = s+num;
  }
  return s;   //return statement
}
```

output

```
Enter how many numbers:
4
Enter number 40
Enter number 20
Enter number 20
Enter number 20
Sum of 4 number = 100
```

5. Types of user defined function

1) Function with no argument and no return value:
2) Function with argument and no return value.
3) Function with argument and return value.
4) Function with no argument and return value

1) Function with no argument and no return value:
There is no data transfer between calling and called function.

```
#include<stdio.h>
#include<conio.h>
void sumofDigits( );  //declaration
void main( )
{
   sumofDigits( );   //function calling
   getch( );
}
void sumofDigits( )
{
   int num, sum=0, rem;
   printf("\n Enter a number \n");
   scanf("%d", &num);
   while (num>0)
   {
      rem = num%10;
      sum = sum + rem;
      num = num/10;
   }
   printf("\n sum of digits is %d", sum);
}
```

Output

```
Enter a number
1234
Sum of digits is 10
```

2) Function with argument but no return value:

The argument of calling function is called actual argument and that of called function is called formal argument. In this type of function there is one way transfer of data from calling to called function.

```
#include<stdio.h>
#include <conio.h>
void sumofDigit (int);
void main ()
{
  int num;
  printf("Enter a no: \n");
  scanf(" %d", &num);
  sumofDigit (num);
  getch();
}
void sumofDigit (int num)
{
    int sum=0, rem;
    while (num>0)
    {
      rem= num%10;
      sum=sum + rem;
      num = num/10;
    }
  printf("\n sum of digits is %d", sum);
}
```

output
Enter a number
3207
sum of digits is 12

3) Function with argument and return value:
This catagory of function passes value from actual argument of calling function to the formal argument of called function and the computed value by the called function is return back to the calling function.

```
#include<stdio.h>
#include<conio.h>
int sumofDigit (int num);    // declaration
void main ( )
{
  int num, result;
  printf ("Enter the data \n");
  scanf ("%d", &num);
  result = sumofDigits (num);    //calling
  printf("\n sum of digit is %d", result);
  getch();
}
  int sumofDigits (int n)
  {
     int sum=0, rem;    //local variables
    while (n>0)
  {
```

```
    rem = n%.10;
    sum = sum + rem;
     n = n/10;
  }
  return sum;
}
```

Output :

```
Enter the data
6153
sum of digit is 15
```

4) Function with no argument and return value :-

This type of function do not pass value from main program through argument to called function and return the value.

```
#include<stdio.h>
#include<conio.h>
int sumofDigit();
void main()
{
    int result;
    result = sumofDigit()
    printf("\n sum of digit is %d", result);
    getch();
}
int sumofDigit()
{
```

```
int sum=0, rem, n;
printf("Enter a no.:");
scanf("%d", &n);
while(n>0)
{
 rem = n%.10;
 sum = sum + rem;
  n = n/10;
  }
  return sum;
}
```

Output

Enter a no. 8162

sum of Digit is 12.

g. Local variables : Variables declared within functions are called local variables. The name and value are valid only within the function in which it is declared.

e.g:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a,b,c;   //local variable
   - - - -
   - - - -
}
```

Here a,b,c are local variables declared within main functions.

5. Global variable or External variable :-

A Global variable is a variable whose name, value and existance are known throughout the program. The global variable is accessable to all the functions defined in the program. It is declared at the beginig of the programs, before the decl of the main function. By default it is initialize to zero.

```
#include<stdio.h>
#include<conio.h>
int  a=10 , b=20;
   //global variables
 void main()
{
  int x,y;   //local variables
  - - - -
  - - - -
}
```

Use of global variable should be done less because it can be modified by any of the functions and fixing the error is difficult.

1) 1 1 2 3 5 8 ... upto 20th term (113)

2) WAP to ~~print~~ read two integers n_1 and n_2 where $n_1 < n_2$. Display all even numbers between these two number.

3) WAP to print the multiplication table from 1 to 20 using nested loop.

#

#

```
main()
{
  int a, b, c, i;
   a=1, b=1;
  printf("%d \t %d ", a, b);
for (i=1; i<=18; i++)
 {
  c=a+b;
     printf("\t%d", c);
  a=b;
  b=c;
 }
getch();
}
```

```
for (i=n1, i<=n2, i++)
{
 if (i%2==0)
 {
  printf("%d \t", i);
 }
```

i) 2 4 6 ... 40

ii) 50 45 40 ... 5

iii) 1 4 9 100

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int i,j;
for (i=1; i<=5; i++)
{
for (j=1; j<=i; j++)
{
printf("%d ",j)
print ("\n")
}
getch( );
}
```

output

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```