Unit 2: Basic Programming Constructs

Character Set

Character set is the building blocks to form basic program elements (Tokens).
C Supports following Character Set:-

1. Alphabets:- A-Z & a-z
2. Digits :- 0-9
3. Special Characters: - *, +, /, %,!,=,==,{,},(,),:,;,.,?,->,~,@,#,$,\,<,>, _,”’, etc.
4. Escape Sequence Characters.
5. White Spaces

C language supports total 256 characters because C uses ASCII code which has 256 characters starting from 0 to 255.

C Tokens

- Tokens are nothing but symbols or programming elements.
- C Program consists of many elements, which are identified by the compiler as tokens.
- Tokens are nothing but sequence of meaningful pattern.
- Tokens supported by C or elements of C programs are as follows
    - Keywords
    - Identifiers
    - Constants
    - Strings
    - Special Symbols
    - Operators
- C Program can be written using these tokens & by following Syntax of C.
- ***Syntax (General Form)*: -** Compiler understandable form or grammar.

Keywords

- *It reserve words (tokens), whose meaning is known to compiler*
- *It is predefined identifier or token.*
- *C supports 32 keywords viz*

auto, break, case, char, const, continue,default, do, double, else, enum, extern, float, for, goto, if, long, register, return, short, signed,sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.

- ***Following are the restrictions to use keywords***
- C keywords has predefine meaning & cannot be changed by user.
- C keywords are in lowercase.
- C keywords cannot be used for other purpose such as variable or function name.
- Following table shows, where we should use each keyword.

Identifiers

- Identifiers refers to the name of variables, functions and arrays.
- Identifiers are user define.

Variables

- *Variable is a data name (memory location name) used to store a data or data value.*
- *Variables are user defined identifiers (tokens).*
- *Variable value may change at the time of program execution.*

- *Variable name can be chosen by programmer in meaningful way so as to reflect its functions or nature.*
- Examples
  int num1, num2, sum;

- **Rules for variable declarations or Rules for Identifiers.**
  Variable name may consist of letters, digits & underscore subject to following rules:
  1. Must begin with letter & may permit underscore (_)
  2. Avoid Special character except underscore (_)
  3. Don't use keywords as Variable name.
  4. Uppercase & lower case Variablenames are significant.
  5. May Contains number in between.

Types of Variable

Type of Variable	***Example***
Character	char ch;
Integer	int num1, num2;
Float	float PI=3.14
Double	double sal;

Constant

- In C constants are declared with "const" keywords.
- Constant variable name should be specified in uppercase letter.
- Constant values does not change at the time of program execution.
- eg. const float PI=3.147;

1. Integer constant:-

- It consists of integer number.
- It consists of any combination of digits taken from set {0, 9}.
- If the constant contains two or more digits, the first digit must be something other than zero. eg. 234, 87, 10, 38.

2. Real constant or floating point constant

Floating point constant can be represented as follows

Fixed Point Notation		**Exponential Notation**		
870000.0	=	8.7×10^4	=	8.7E + 05
0.0000033	=	3.3×10^{-6}	=	5.2E - 06

- A floating point constant base is 10 from {0-9} that contain either decimal point or an exponent.

Valid Real Constant:-

567.4
5.0
7.0E5
8.0E-5
.5
5**.**

3. Single character constants:-

C language supports 256 characters.
Any single character should be enclosed in single Quotation.

eg.	Valid	Invalid
	'A'	'xy'
	'B'	'xyz'
	'5'	"x" – due to double inverted quama
	'{'	

C code to illustrate character constant

Program	Output

`#include<stdio.h>`
`void main();{`
`const char ch ='A'`
`printf("%c",ch);`
`printf("%d",ch);`
`printf("%c",49);`
`printf("%d",'a');`
`}`
	A
	65
	1
	97

4. String Constant:-

"A string constant consists of double quotation marks."

eg. "Bibishan"
"Welcome"
"123"
"xyz-546"
"A"

Data type

- Data types or types are used to represent the type of data stored in variables.
- Data type is an instructions or keywords given to compiler for organizing data.
- Data types are classified into two category as follows.

Data types			
Primary OR Basic OR Fundamental OR Scalar OR Built in OR Standard OR Primitive datatype	Secondary Data type		
	Use defined	**Derived**	**Empty**
Char int float double	struct union enum typedef	Array Pointer	void

Table:- Types of data type

❖ ***Here we will discuss only Primary datatypes***

1) char 2) int 3) float 4) double

1. **char:-**
   - It is keyword used as datatype to declare character type variable are constant
   - For char type 1 byte = 8 bit memory space is allocated.
     eg.
     char ch = 'A';
     char choice = 'y';
     char esc='\n';

2. **int:-**
   - It is keyword used to declare integer datatype.
   - For int type 2 byte i.e. 16 bit memory space is allocated.
     eg. int u = 87;
     int Value = 33;

3. **float :-**
   - Float is keyword used as datatype to declare floating point variable or const.
   - For float type 4 byte i.e. 32 bit of memory space is allocated.
     eg. float z = 2.5;

4. **double :-**
   - Double is keyword, similar to float.
   - In double variable 8 byte i.e. 64 bits memory space is allocated.
     - Eg. double x = 280194.02;

❖ **User defined datatype**
   - Defined or created by user
     eg. struct, typedef, enum, union etc.

❖ **Derived datatype**
   - This datatype is derived from primary datatype.
     eg. Pointer & Array.

❖ **Empty datatype value less)**
   - It is valueless datatype
     eg. void

Operators and Expressions

- In programming languages, operators are the symbols used to manipulate data and variable values.

C Support following operators:-

Operators	***Symbols Used***
Arithmetic	+,-,*,/,%
Assignment	=,+=,-=,*=,/=,&=,>>=%=,<<= etc
Relational	>,<,>=,<=,==,!=
Logical	&&,||,!
Increment & Decrement	++, --
Bitwise	&, |, ~, ^, <<, >>
Conditional	?:
Other Special	., ->, *, &, sizeof, (type),[ ],( ),,

Arithmetic Operator:-

Addition(+)	**Subtraction(-)**	**Multiplication(*)**	**Division(/)**	**Modulus(%)**
May be unary or binary	May be unary or binary	It is binary	It is binary	It is binary
Perform addition	Perform subtraction	Perform multiplication	Perform Division	Perform Division & gives reminder
Work with any data type	Work with any data type	Work with any data type	Work with any data type	Work with int data type.

Example of Arithmetic Operator:-

```
main( ){
        printf("%d",10+8);
        printf("%d",10-8);
        printf("%d",10*8)
        printf("d",10/8);
        printf("%d",10%8);
}
```

Assignment Operator (=):-

- Binary Operator:- It is used to assign value of right side operand to left side operand
- Provided left side operand should be variable

  eg.

Valid	***Invalid***
x = y	x + y = z
z = x + y	10 = x
x = 10	x ==y
Area = 0.5 * l * b;	

- ***Syntax:-***

  identifier = expression

  - Identifier represents variable name and expression represents a constants, a variable or a complex expression.

- C also supports some compound assignment operator.

Operator Symbol	*Example*	*Meaning*
+=	x+=D	x = x + D;
-=	x-=D	x = x – D;
*=	x*=D	x = x * D;
/=	x/=D	x = x/D;
%=	x%=D	x = x%D;

Example of Assignment Operator

```
#include<stdio.h>
#include<conio.h>
void main(){
int x = 10;
x*='A' + 'B';
printf("%d",x);
getch();
}
Output:
1310
```

Demonstration
x*= 'A' + 'B'→65+66
x = x * 131
x = 10 * 131
x = 1310

Relational Operator:-

- Relational Operators are used to check single condition.
- All relational operators are binary in nature.

Operator Symbols	**Meaning**
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

- Example:-
    a. amt>=500
    b. min_bal>=1000
    c. (x+y)!=(x-y)

- **Relational Expression:-**is combination of operands with relational operator.
- The value return by relational expression is 1 or 0.
- Example of Relational Operator

```
#include<stdio.h>
#include<conio.h>
void main(){
clrscr();
printf("%d\n",5>10);
printf("%d\n",10!=9);
printf("%d\n",'a'!='A');
getch();
}
Output:-
                0
                1
```

```
1
```

Logical Operator:-

Logical Operators are used to check more than one condition(Relational expression).

Operator Symbols	Meaning	Truth Table

&& Logical AND

RE1	RE2	&&
1	0	0
1	1	1
0	0	0
0	1	0

|| Logical OR

RE1	RE2	&&
1	0	1
1	1	1
0	0	0
0	1	1

! Logical Not

RE	!
1	0
0	1

Logical AND(&&) and Logical OR(||) is Binary Operator
Logical NOT(!) is unary operator.
Like relational expression, logical expression also yields a value 1 or 0.

Example:-

(a>b)&&(a==20)

In above example, we are checking two conditions with the help of logical AND

Example to illustrate use of logical operator.

```
#include<stdio.h>
#include<conio.h>
void main(){
clrscr();
printf("%d\n",(5>10)&&(10<20));
printf("%d\n",5&&5);
printf("%d\n",!(10!=10));
printf("%d\n",('a'!='A')&&('a'>'A'))
getch();
}
Output:-
                0
                1
                1
                1
```

Increment & Decrement Operator:

Increment Operator:-

- Increment Operator (++):- Increment operator (++) adds 1 to its operand.
- Two sub types are as follows:-

Pre-Increment	**Post-Increment**
++X	X++
1.Increment(X=X+1)	2.Assignment(X=X)
2.Assignment(X=X)	1.Increment(X=X+1)

While processing with increment two operations are performed depending on pre or post increment.

Decrement Operator (--):- Decrement operator(--)subtract value from its operand.

Two sub types are as follows:-

Pre-Decrement	**Post-Decrement**
--X	X--
1.Decrement(X=X-1)	2.Assignment(X=X)
2.Assignment(X=X)	1.Decrement(X=X-1)

While processing with decrement two operations are performed depending on pre or post decrement.

Example to illustrate use of increment operator:-

```
#include<stdio.h>
#include<conio.h>
void main(){
int x=87;
clrscr();
printf("%d\n",x++);
printf("%d",x);
getch();
}
Output:-
                    87
                    88
```

Conditional Operator or Ternary Operator(?:)

- Conditional operator functionality is similar to the conditional control statement if-else.
- **Syntax:**

  Variable= (condition)? variable1:variable2;
- In above syntax, condition is evaluated first.

  If it is evaluated as true (1) then value of varible1 will be assign to Variable.

  If it is evaluated as false (0) then value of variable2 will be assign to Variable
- Example:-

  X=(5>3)?5:3;

  In above example, (5>3) is evaluated as true that's why 5 is assign to X.

Example to illustrate use of Ternary Operator.

```
/* program to find largest amongst two number */
#include<stdio.h>
#include<conio.h>
void main( ){
int x,y,max ;
clrscr( );
printf("Enter two no");
scanf("%d%d",&x,&y);
max=(x>y)?x:y;
printf("Largest no is %d",max);
getch( );
}
Output:-
                    Enter two no 12 15
                    Largest no is 15
```

Other or Special Operators:-

Operator Symbol	Meaning
,	Comma Operator
.	Structure Member Access
->	Structure Member Access
[ ]	Array Index operator
( )	Function Call
&	Address of operator
*	Value at operator
Sizeof	sizeof operator
(type)	Typecasting Operator.

Precedence of Operator or Priority of C Operators.

Precedence	Operator Symbol	Associativity of Operator
1.	( ) [ ] . ->	Left to Right
2.	Unary +, Unary -, ++, --, ~, ! ,*, &, sizeof, (type)	Right to Left
3.	* / %	Left to Right
4.	+ -	
5.	<< >>	
6.	<<= >>=	
7.	== !=	
8.	&	
9.	^	
10.	|	
11.	&&	
12.	||	
13.	?:	Right to Left
14.	= += -= *= /= %=	Right to Left
15.	,	Left to Right.

Expression

- Expression is a sequence of operands and operators that reduces to a single value.
- - Example 25+25*3

Question:-

Generate the C equivalent Statements for the following mathematical equations.

a) $S = \frac{a+b+c+d}{2}$

Ans:- C equivalent Statement is

$S = {(a+b+c+d)}/{2}$

b) $S = \frac{(x+y+z)}{(x*y-z)}$

Ans:- C equivalent Statement is

$S = {(x+y+z)}/{(x*y-z)}$

c) $X = \frac{b \pm \sqrt{b^2-4ac}}{2a}$

Ans:- C equivalent Statement is

$X1 = -b + {sqrt(b*b- 4*a*c)}/{(2*a)}$

$X2 = -b - {sqrt(b*b- 4*a*c)}/{(2*a)}$

➔ due $\pm$ we get two equivalent statements shown above

d) if x=60°

$x = x \times \frac{\pi}{180}$

C equivalent statement is

$x = 60 * {3.147}/{180}$

Basic Structure of C program or Blocks of C program

- C is structural or procedural programming language
- The most distinguishing feature of a structured language is that it uses ***blocks.***
- The C Language is based on ***building blocks.***
- A ***block*** is a set of Statements that are logically connected.
- Following is the structure or blocks or C program.

Block or Basic Structure	**Example**
Documentation	/* This is C program to do addition */
Link Section - Preprocessor Directives	#include<stdio.h>
Definition Section	#define MAX 100
Global Declaration Section	int add;

int main( ){
1. Declaration Part
2. Executable Part
}
	main( ){
	/* Body of main( ) */
	}

Sub Program Section - Function 1 - Function 2 - Function n	User Define Functions(if any)

1. **Documentation:-**
   - This is a comment line.
   - We can use comment anywhere in program.
   - Comment gives information about program, its aim & use of any instructions.
   - It is optional and non-executable.
   - Eg.
     /* C program to add two numbers */

2. **Link Section**
   - Preprocessor allows to include external file.
   - Preprocessor, are processed before executing the program.
   - Eg.
     #include<stdio.h>

3. **Definition Section**
   - It defines Symbolic constants.
   - Eg.
     #define MAX 100
4. **Global Declaration**
   - With this we can declare global variable which are accessible through out the program.
   - And user define functions can be define here.
   - Eg.
     int add;
5. **main( )**
   - Every C program contains one function called **main().**
   - Execution of C program always starts from main().

- main( ) is a function of operation system.

6. **Subprogram Section**
   - It contains user define function definitions.
   - User define functions are called from main( ).

All sections, except the main( ) section may be absent when they are not required.

- Example:-

```
/* C program to do addition of two number */
#include<stdio.h>
int add;
main( ){
int a=5,b=7;  /* variable declaration.
add=a+b;
printf("Addition is %d",add);
}
```

Comments

- C style comment begins with the character pair '/*' and ends with '*/'.
- The comment provides documentation in C program.
- We can include comment anywhere in C program.
- Comments are non-executable statements
- Comments are used to explain operation of a program.
- /* it is multiline comment*/
- //it is single line comment

Example.

```
/* C program to display wel-come message*/
#include<stdio.h>     //Header file inclusion.
main( ){
printf(" Wel-Come"); //it will display wel-come message.
}
```

C Control Constructs/Statements/Structures

- Control statements are used to alter (edit) the flow of program execution.
- Control statements evaluate the condition (uses relational and/or logical operators) & then control the flow of execution.
- C control constructs/statements are as follows.

Decision Making Statements or Conditional Statements	Loop Control Statements or Iterative Statements	Jump Control Instructions or Branching Statement
• if-else • Nested if-else • else if Ladder • switch-case	• for • while • do-while	• break • continue • return • exit()

if-else

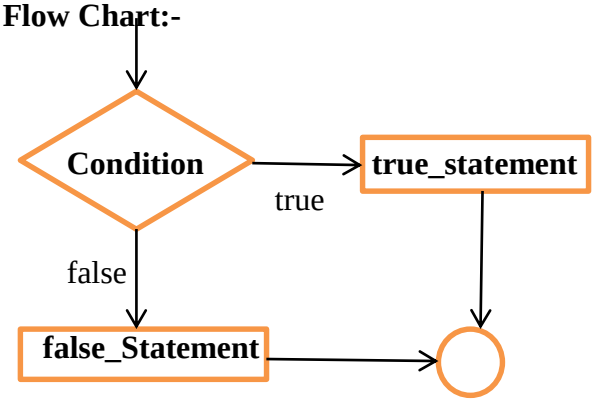
- if-else is decision making statement.
- else clause is extension to the if clause & contains false part.
- "if" & "else" are keyword used to make a decision to control the flow of execution.

Syntax:-

1. **if-else with single statement**

```
if(condition)
        true_statement;
else
        false_statement;
```

Flow Chart:-

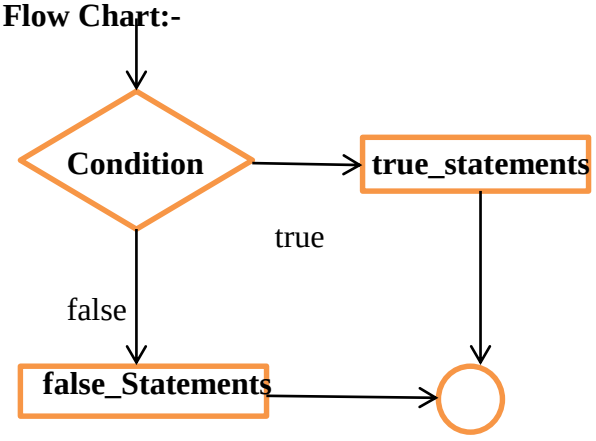


In above syntax if condition is evaluated first, if condition is evaluated as true then true_statement is getting executed. If condition is evaluated as false then false_statement is getting executed.

2. **if-else with multiple statement.**

```
if(condition){
        true_statement 1;
        true_statement 2;
        …….
        true_statement n;
}
else{
        true_statement 1;
        true_statement 2;
        …….
        true_statement n;
}
```

Flow Chart:-



The use of curly braces shows the increased scope of 'if'& 'else' clause. If condition is evaluated as true then true_statements are exeuted otherwise false_statements are executed.

Example:- Programming Example to show use of if-else

```
/* C program to check entered number is even or odd. */
#include<stdio.h>
main( ){
        int num;
        printf("Enter one number \n");
        scanf("%d",&num);
        if(num%2==0)
                printf("%d is EVEN",num);
        else
                printf("%d is ODD",num);
}
```

Output:-

```
Enter one number
33
33 is ODD
```

Nested if-else

It is complex decision making statement. if clause and/or else clause can be nested one inside another.

In nested if-else, if clause may have if-else and/or its chain. Similarly else clause may have if-else and/or its chain or both if & else clause have sub if-else clause/block.

Complex nested if-else (Multiple decision making) statement may cause problem to maintain program.

Syntax:-

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

In above syntax, inner if and/or else clause may have if-else clause in one another. It may confuse, that's why be careful when nesting if-else. Nested if-else is nothing but chained with one another.

Example:- Programming example to show use of nested if-else.

```
/* C program to find largest amongst three numbers */
    #include<stdio.h>
    main( ){
    int n1,n2,n3;
    printf("Enter three numbers\n");
    scanf("%d%d%d",&n1,&n2.&n3);
        if(n1>n2&&n1>n3){
            printf("%d is greater",n1);
        }
        else{
            if(n2>n1&&n2>n3){
                printf("%d is greater",n2);
            }
            else{
                printf("%d is greater",n3);
            }
        }
    }
```

In above example, else block/clause contains if-else.

Output

```
Enter three numbers
16      32      5
32 is greater
```

else-if Ladder

It is common programming construct used to make multiple decision. Sometime we may call this as if-else-if ladder.

It is different than that of nested if-else & less confusion than that of nested if-else.

Syntax:-

```
if(condition){
    statements:
}
else if(condition){
    statements;
}
else if(condition){
    statements;
}
:
:
else{
    statements;
}
```

In above format, condition is evaluated from top to down.

Example:- Programming example to show use of else-if ladder

```
/* C program to find largest amongst five numbers */
    #include<stdio.h>
    main( ){
    int n1,n2,n3,n4,n5;
    printf("Enter five numbers\n");
    scanf("%d%d%d%d%d",&n1,&n2.&n3,&n4,&n5);
```

```
if(n1>n2&&n1>n3&&n1>n4&&n1>n5)
        printf("%d is largest",n1);
else if(n2>n1&&n2>n3&&n2>n4&&n2>n5)
        printf("%d is largest",n2);
else if(n3>n1&&n3>n2&&n3>n4&&n3>n5)
        printf("%d is largest",n3);
else if(n4>n1&&n4>n3&&n4>n4&&n4>n5)
        printf("%d is largest",n4);
else
        printf("%d is largest",n5);
}
```

Output

```
Enter five numbers
5       7       3       9       21
21 is largest
```

switch case

- It is multiple branching statement.
- It checks for equality not condition.

Advantages:-

- Easy to use
- Easy to find out errors(if any) & debug.
- Complexity of program is minimized.

Syntax:

```
switch(equality_constant or variable or expression){
case constant1:
        statement1;
        break;
case constant2:
        statement2;
        break;
:
:
case constant_n:
        statement2;
        break;
default:
        default_statement;
}
```

In above syntax:

- equality_constant or variable or expression should be of type int or char.
- default is optional

Example:- Programming example to show use of switch-case

```
#include<stdio.h>
main( ){
int choice=2;
        switch(choice){
                case 1:
                        printf("Pune\n");
```

```
                    break;
                case 2:
                    printf("Shirpur\n")
                    break;
                case 3:
                    printf("Delhi\n");
                    break;
                case 4:
                    printf("Mumbai\n");
                default:
                    printf("Any other");
            }
}
```

Output

Shirpur

Nested switch case:

- outer switch block may contain inner switch block, i. e. switch with in switch.
- The inner and/or outer switch may contain same equality constant.

Syntax:-

```
switch(equality_constant or variable or expression){
    case constant1: statement1; break;
    case constant2: statement2; break;
    :
    case constant_n: statement2;  break;
    default:  default_statement;
        switch(equality_constant or variable or expression){
            case constant1: statement1; break;
            case constant2: statement2; break;
            :
            Case constant_n: statement2; break;
            default:  default_statement;
        }

    }
```

Loop Control Statements Or Iterative Statements or Loop Control Constructs

Looping???

Statements in a block are repeatedly executed for certain number of times or period.

Steps used in Loop???

1. **Initialization**
2. **Condition or Test Expression**
3. **Update expression.**

Initialization:-

- Initial value i.e. starting value is assigned to loop variable(i.e. index variable)
- Executed only ones in life time of loop.

Condition:-

- Every time condition is evaluated, if it is evaluated as true then control will get entry in loop otherwise loop will be terminated.
- Condition is either true or false as we use relational and/or logical operators to write condition.

Update Expression:-

- Any expression to update loop variable, after updating loop variable control will check condition again & this will be iterative till condition becomes false.

Loops are categorized into

1. Entry Controlled Loop
   i. for
   ii. while
2. Exit Controlled Loop
   i. do while

for

- it is iterative or loop controlled statement.
- for is keyword used as loop control statement.

Syntax:-

```
for( initialization; condition ;  update_expression ){
        statements;              //body of for
}
```

Note:- write about initialization, condition & update_expression.

Example:- Programming example to show use of for loop.

```
/* Write a c program to print first 'n' numbers. */
        #include<stdio.h>
        main( ){
        int i,n;
        printf("Enter value of n:");
        scanf("%d",&n);
                for(i=1;i<=n;i++)
                printf("%d\t",i);
        }
```

Output

```
Enter value of n:8
1       2       3       4       5       6       7       8
```

/* Write a c program to print sum of first n numbers. */

```
/* i. e. sum=1+2+3+……+n*/

        #include<stdio.h>
        main( ){
        int i,n,sum=0;
        printf("Enter value of n:");
        scanf("%d",&n);
                for(i=1;i<=n;i++)
                sum+=i;
        printf(" Sum of numbers is %d",sum);
        }
```

Output

```
Enter value of n:8
1       2       3       4       5       6       7       8
Sum of numbers is 36
```

/* C program to find factorial of a number */

```
        main( ){
        int n,fact=1,i;
        printf("Enter number to find factorial");
        scanf("%d",&n);
        if(n==0){
                printf("Factorial of Zero is %d",fact);
        }
        else{
                if(n>0){
                        for(i=0;i<=n;i++)
                                fact*=i;
                        printf("Factorial of %d is %d",n,fact);
                }
                else{
                        printf("As number is negative, factorial is undefined\n");
                }
        }}
```

Output

```
Enter number to find factorial 5
Factorial of 5 is 120
```

while

- it is loop control statement or iterative statement.
- while is keyword & used as loop control statement.

Syntax:-

```
Initlialization;
while(condition){
        //while body
        update_expression;
}
```

In above syntax, condition is evaluated first, if it is evaluated as true then body of loop is executed along with update_expression, again condition will be checked & if it is true then body of loop is executed again, otherwise execution will be terminated.

Example:- Programming example to show use of while loop.

```
/* Write a c program to print first 'n' numbers. */
#include<stdio.h>
main( ){
int i,n;
printf("Enter value of n:");
scanf("%d",&n);
        i=1;    //initial value of loop variable…
        while(i<=n){
        printf("%d\t",i);
        i++;  //update expression to update loop variable…
        }
}
```

Output

```
Enter value of n:9
1       2       3       4       5       6       7       8       9
```

/* Write a c program to print sum of digits of a given number. */

```
#include<stdio.h>
main( ){
int n,d,sum=0;
printf("Enter one number to find its sum of digits");
scanf("%d",&n)     //with this input function we will get initial value of loop variable.
        while(n!=0){
                d=n%10;
                sum+=d;
                n/=10; // Update expression.
        }
printf("Sum of individual digits is %d",sum);
}
```

Output

```
Enter one number to find its sum of digits 2357
Sum of individual digits is 17
```

/* Read an integer through keyboard. Sort odd and even numbers(digits) by using while loop. Write a program to perform sum of odd and even numbers(digits) separately and display the result*/

```
#include<stdio.h>
        main( ){
        int n,d,esum=0,osum=0;
        printf("Enter one number to find  sum of even and odd digits\n");
        scanf("%d",&n)
                while(n!=0){
                        d=n%10;
                                if(d%2==0) esum+=d;
                                else      osum+=d;
                        n/=10;
                }
        printf("Sum of even  digits is %d \n Sum of odd digits is %d ", esum, osum);
        }
```

Output

```
Enter one number to find sum of even and odd digits
2356
Sum of even digits is 8
Sum of odd digits is 8
```

/* Write a c program to print reverse of a given number. */

```
        #include<stdio.h>
        main( ){
        int n,d,rev=0;
        printf("Enter one number\n");
        scanf("%d",&n)      //with this input function we will get initial value of loop variable.
                while(n!=0){
                        d=n%10;
                        rev=rev*10+d;
                        n/=10; // Update expression.
                }
        printf("Reverse number is %d",rev);
        }
```

Output

```
Enter one number
2357
Reverse number is 7532
```

do-while

- it is exit controlled loop
- it is loop control statement or iterative statement.
- do & while is keyword & used as loop control statement.

Syntax:-

```
Initlialization;
do{
//do body
update_expression;
} while(condition);
```

In above syntax, first loop is executed before checking condition & then here after condition is evaluated for every iteration, if it is evaluated as true then body of loop is executed along with update_expression, again condition will be checked & if it is true then body of loop is executed again, otherwise execution will be terminated.

Example:- Programming example to show use of do-while loop.

```
/* Write a c program to print first 'n' numbers. */
#include<stdio.h>
main( ){
int i,n;
printf("Enter value of n:");
scanf("%d",&n);
        i=1;     //initial value of loop variable…
        do {
        printf("%d\t",i);
        i++;   //update expression to update loop variable…
        } while(i<=n);
}
```

Output

```
Enter value of n:9
1       2       3       4       5       6       7       8       9
```

Jump Control Instructions or Branching Statement

Followings are the jump control or branching statement supported by C.

- break
- continue
- return

- Jump control statement transfer the control from one position to another position in program while execution.

break:-

- 'break' is used to exit the execution of any loop.
- 'break' is used to stop execution of remaining *cases* in *switch*.

Syntax:-

```
break;
```

In above syntax, we have only break keyword followed by comma, and we can write this in switch and/or any loops in C.

Example:- Programming example to show use break.

```
#include<stdio.h>
main(){
    int i;
        i=1;
        while(1){
        if(i==4) break;
        printf("%d\t",i);
        i++;
        }
}
```

In this example, we have used 'break' keyword to break while loop… and execution of while loop will be terminated whenever condition *if(i==4)* is evaluated as true & execution of while will be terminated.

Output

```
1    2    3    4
```

Continue:-

- 'continue' is used to continue the execution of next iteration by skipping current iteration of any loop.

Syntax:-

```
continue;
```

In above syntax, we have only 'continue' keyword followed by comma, and we can write this in any loop.

Example:- Programming example to show use continue.

```
#include<stdio.h>
main( ){
        int i;
                for(i=1;i<=5;i++){
                if(i==4) continue;
                printf("%d\t",i);
                }
}
```

In this example, we have used 'continue' keyword to continue iteration of loop. And whenever condition *if(i==4)* is evaluated as true & next iteration will be continued.

Output

```
1       2       3       5
```

return:-

- It is jump control instruction.
- return is used to return the value to calling place,(function call)
- it is keyword.

Syntax:-

```
return(expression);
```

- returning more than one value is impossible.

Example:-

i. return 0;
ii. return (a+b);
iii. return a;
iv. return (3.147*r*r);

/* Programming example to show use of return. */

```
#include<stdio.h>

int main( ){

printf("Default return type of main( ) is integer");

return 0;

}
```

Output

```
Default return type of main( ) is integer
```

Write a program that will calculate the sum of every third integer beginning with i=2 and for all values of i that are less than 100 using i. for iii while iii. Do while loop.

Using For	Using While	Using Do While

main( ){
int i , sum=0;
for(i=2; i<100; i=i+3){
sum = sum + i;
}
printf("Sum is %d",sum);
}
	main( ){
	int i=2 , sum=0;
	while(i<100){
	sum = sum + i;
	i = i + 3;
	}
	printf("Sum is %d",sum);
	}
		main( ){
		int i=2 , sum=0;
		do{
		sum = sum + i;
		i = i + 3;
		} while(i<100);
		printf("Sum is %d",sum);
		}

Write a program to generate and print the first N terms of Fibonacci series 1,2,3,5,….up to N terms.

```
#include <stdio.h>
int main( )
{
  int n,F,F1=0,F2=1,i=2;                          // variable declaration & initialization
  printf("Enter How many terms \n");
  scanf("%d",&n);                                 //read n terms
  printf("First %d of Fibonacci series is \n",n);  // display first two terms 1 & 2
  printf("%d\t%d",F1,F2);
        while(i < n){                             // loop to display 3rd term onward up to n.
                        F = F2 + F1;              // Fibonacci of current term is addition of previous two terms
                        printf("\t%d",F);         //print terms
                        F1=F2;                    //last second term is assigned to last first
                        F2=F;                     // current term is assigned to F2
                        i++;                      // I in incremented to go up to n.
        }
   return 0;
}
```

Output:-

Enter How many terms 5

First 4 of Fibonacci series is
0 1 1 2

Write a C program to check whether given number is prime number or not.

```
main( ){
        int i=2,n,flag=0;

        printf("Enter Number\n");
        scanf("%d",&n);
                while(i<n) {
                        if(n%i==0){
                                flag=1;
                                break;
                        }
                        i++;
                }
                if(flag = = 0)
                        printf("Number is Prime");
                else
                        printf("Not Prime No");
}
```