UNIT I BASICS OF C PROGRAMMING

Introduction to programming paradigms - Structure of C program - C programming: Data Types –Storage classes - Constants – Enumeration Constants - Keywords – Operators: Precedence and Associativity - Expressions - Input/Output statements, Assignment statements – Decision making statements - Switch statement - Looping statements – Pre-processor directives - Compilation process

STRUCTURE OF A C PROGRAM

In general, a C program is composed of the following sections:

Section 1: Pre-processor directives
Section 2: Global declarations
Section 3: Functions

Section 1 and 2 are optional, Section 3 is compulsory.

Structure of a C Program

```
Documentation Section

Pre-processor directives

Definition Section and Global declarations

void main()

{

        Declaration part

        Executable part

}

Sub Program Section

{

        Body of the Sub

}
```

Comments:

- Comments are used to convey a message and used to increase the readability of a program.
- They are not processed by the compiler.

There are two types of comments:

1. Single line comment
2. Multi line comment

Single line comment

A single line comment starts with two forward slashes (//) and is automatically terminated with the end of the line.

E.g. //First C program

Multi-line comment

A multi-line comment starts with /* and terminates with */.A multi-line comment is used when multiple lines of text are to be commented.

E.g. /* This program is used to find
Area of the Circle */

Section 1.Preprocessor Directive section

- The preprocessor directive section is optional.
- The pre-processor statement begins with # symbol and is also called the pre-processor directive.
- These statements instruct the compiler to include C preprocessors such as header files and symbolic constants before compiling the C program.
- The preprocessor directive is terminated with a new line character and not with a semicolon.
- They are executed before the compiler compiles the source code.

Some of the pre-processor statements are listed below:

(i)Header files

#include<stdio.h> - to be included to use standard I/O functions : prinf(),scanf()
#include<math.h> -to be included to use mathematical functions :eg)sqrt()

(ii)Symbolic constants

#define PI 3.1412
#define TRUE 1

Section 2: Global declaration Section

This section is used to declare a global variable. These variables are declared before the main() function as well as user defined functions.

Global variables are variables that are used in more than one function.

Section 3: Function Section

This section is compulsory. This section can have one or more functions. Every program written in C language must contain main () function. The execution of every C program always begins with the function main ().

Every function consists of 2 parts

1. Header of the function
2. Body of the function

1. **Header of the function**

The general form of the header of the function is

[return_type] function_name([argument_list])

2. Body of the function

The body of the function consists of a set of statements enclosed within curly brackets commonly known as braces. The statements are of two types.

1. **Non executable statements:**

These are declaration statements, which declares the entire variables used. Variable initialization is also coming under these statements.

2. **Executable statements :**

Other statements following the declaration statements are used to perform various tasks. For example printf function call statement.

Non-executable statements are written first and then executable statements are written

E.g. C Program

```
Line 1:     // First C program
Line2: #include<stdio.h>
Line3:      main()
Line4:      {
Line5       Printf("Hello");
Line6       }
```

Line1 is a comment; Line 2 is a preprocessor directive. Line3 is a header of the function main. Line 4, 5, 6 form the body of main function.

Example Program:

```
/*Addition of two numbers*/          ← Documentation Section
#include<stdio.h>                    ← Pre-processor directives
#define A 10                        ← Definition Section
int c;                              ← Global declarations
int sum(int,int);
main()                              ← Main() functions
{
        int b;
        printf("Emter the value for B:");
        scanf("%d",&b);                          ← Execution Part
        c=sum(b);
        printf("\n Answer=%d",c);
        getch();
}
int sum(int y)
{
c=A+Y;                              ← Sub Program
return(c);
}
```

PROGRAMMING RULES

1. All statements should be written in lower case letters. Upper case letters are only used for symbolic constants.
2. Blank spaces cannot be used while declaring a variable, keyword, constant and function.
3. The programmer can write the statement anywhere between the two braces following the declaration part.
4. The user can also write one or more statements in one line by separating semicolon (;).Ex:
   ```
   a=b+c;
   d=b*c;
         or
   a=b+c;  d=b*c;
   ```
5. The opening and closing braces should be balanced. For example, if opening braces are four, then closing braces should also be four.

DATA TYPES

Data type determines the possible values that an identifier can have and the valid operations that can be applied on it.

In C language data types are broadly classified as:

1. Basic data types(Primitive data types)
2. Derived data types
3. User-Defined data types

1. Basic data types:

There are five basic data types:

(i)Character - char
(ii)Integer - int
(iii)Single-Precision floating point - float
(iv)Double Precision floating point - double
(v)No value available - void

Data types and their memory requirements

S.No	Data type	No of bytes required	Range
1	Char	1	-128 to 127
2	Int	2	-32768 to 32767
3	Float	4	$3.4*10^{-38}$ to $3.4*10^{38}$
4	Double	8	$1.7*10^{-308}$ to $1.7*10^{308}$
5	Unsigned char	1	0 to 255

Derived data types

These data types are derived from the basic data types. Derived data types available in C are:

1. Array type eg. Char[ ], int [ ]
2. Pointer type eg. Char*, int*
3. Function type eg. int (int,int), float(int)

User-defined data types

The C language provides the flexibility to the user to create new data types. These newly created data types are called user-defined data types. The user defined data types available in C can be created by using:

1. Structure
2. Union
3. Enumeration

Type qualifiers and Type Modifiers

The declaration statement can optionally have type qualifiers or type modifiers or both.

Type qualifier

A type qualifier is used to indicate the special properties of data within an object. It never affects the range of values & the arithmetic properties of the declared object.

Two type qualifiers available in C are:

Const qualifier: Object value will not be changed during the execution of a program

Volatile qualifier

Type Modifiers

A type modifier modifies the base type to yield a new type. It modifies the range and the arithmetic properties of the base type.

Type modifiers and the corresponding keywords available in C are:

- Signed (signed)
- Unsigned (unsigned)
- Short (short)
- Long (long)

Constants

A constant is an entity whose value can't be changed during the execution of a program. Constants are classified as:

1. Literal constants
2. Qualified constants
3. Symbolic constants

Literal constants

Literal constant or just literal denotes a fixed value, which may be an integer, floating point number, character or a string.

Literal constants are of the following types.

1. Integer Literal constant
2. Floating point Literal constant
3. Character Literal constant
4. String Literal constant

Integer Literal constant

Integer Literal constants are integer values like -1, 2, 8 etc. The rules for writing integer literal constant are:

- An Integer Literal constant must have at least one digit
- It should not have any decimal point
- It can be either positive or negative.
- No special characters and blank spaces are allowed.
- A number without a sign is assumed as positive.
- Octal constants start with 0.
- Hexadecimal constant start with 0x or 0X

Floating point Literal constant

Floating point Literal constants are values like -23.1, 12.8, 4e8 etc.. It can be written in a fractional form or in an exponential form.

The rules for writing Floating point Literal constants in a fractional form:

- A fractional floating point Literal constant must have at least one digit.
- It should have a decimal point.
- It can be either positive or negative.
- No special characters and blank spaces are allowed.

The rules for writing Floating point Literal constants in an exponential form:

- A Floating point Literal constants in an exponential form has two parts: the mantissa part and the exponent part. Both are separated by e or E.
- The mantissa part can be either positive or negative. The default sign is positive.
- The mantissa part should have at least one digit.
- The mantissa part can have a decimal point.
- The exponent part should have at least one digit
- The exponent part cannot have a decimal point.
- No special characters and blank spaces are allowed.

Character Literal constant

A Character Literal constant can have one or at most two characters enclosed within single quotes. E.g, 'A' , 'a' , ' n '.

It is classified into:

- Printable character literal constant
- Non-Printable character literal constant.

Printable character literal constant

All characters except quotation mark, backslash and new line characters enclosed within single quotes form a Printable character literal constant. Ex: 'A' , '#'

Non-Printable character literal constant.

Non-Printable character literal constants are represented with the help of escape sequences. An escape sequence consists of a backward slash (i.e. \) followed by a character and both enclosed within single quotes.

Constant	Meaning
'\a'	audible alert
'\b'	backspace
'\f'	form feed
'\n'	new line
'\0'	null
'\v'	vertical tab
'\t'	horizontal tab
'\r'	carriage return

String Literal constant

A String Literal constant consist of a sequence of characters enclosed within double quotes. Each string literal constant is implicitly terminated by a null character.

E.g. "ABC"

Qualified constants:

Qualified constants are created by using const qualifier.

E.g. const char a = 'A'

The usage of const qualifier places a lock on the variable after placing the value in it. So we can't change the value of the variable a

Symbolic constants

Symbolic constants are created with the help of the define preprocessor directive. For ex #define PI= 3.14 defines PI as a symbolic constant with the value 3.14. Each symbolic constant is replaced by its actual value during the pre-processing stage.

Enumeration Constants:

An enumeration is a user-defined data type or constant. Enumeration is achieved by using the keyword **enum.**

The enumeration type is an integral data type.

SYNTAX:

```
enum enum_name{

const1,

 const2,

...

constN

};
```

Example:

```
#include <stdio.h>
main()
{
enum Day{ Monday =1, Tuesday, Wednesday, Thursday};
enum { A= 3, B , C , Z = 400, X, Y };
printf("Wednesday = %d\n", Wednesday);
printf("B = %d \t C = %d\n", B,C);
printf("X = %d \t Y = %d\n", X,Y);
printf("Thursday/Tuesday = %d\n", Thursday/Tuesday);
}
```

Output:

Wednesday=3

B=4 C=5

X=401 Y=402

Thursday/Tuesday=2

Enumerated Typr Declarations:

When you create an enumerated type, only blueprint for the variable is created. Here's how you can create variables of enum type.

```
enum boolean { false, true };
enum boolean check;
```

Another syntax:

```
enum boolean
{
  false, true
} check;
```

Example: Enumeration Type

```
#include <stdio.h>
enum week { sunday, monday, tuesday, wednesday, thursday, friday, saturday };
int main()
{
        enum week today;
         today = wednesday;
        printf("Day %d",today+1);
        return 0;
}
```

Output

Day 4

Operators: Precedence and Associativity – Expressions:

An expression is a sequence of operators and operands that specifies computation of a value.

For e.g, a=2+3 is an expression with three operands a,2,3 and 2 operators = & +

Operands

An operand specifies an entity on which an operation is to be performed. It can be a variable name, a constant, a function call.

E.g: a=2+3 Here a, 2 & 3 are operands

Operator

An operator is a symbol that is used to perform specific mathematical or logical manipulations.

For e.g, a=2+3 Here = & + are the operators

Simple Expressions & Compound Expressions

An expression that has only one operator is known as a simple expression.

E.g: a+2

An expression that involves more than one operator is called a compound expression.

E.g: b=2+3*5

Precedence of operators

- The precedence rule is used to determine the order of application of operators in evaluating sub expressions.
- Each operator in C has a precedence associated with it.
- The operator with the highest precedence is operated first.

Associativity of operators

The associativity rule is applied when two or more operators are having same precedence in the sub expression.

An operator can be left-to-right associative or right-to-left associative.

Category	Operators	Associativity	Precedence
Unary	+, -, !, ~, ++, - -, &, sizeof	Right to left	Level-1
Multiplicative	*, /, %	Left to right	Level-2
Additive	+, -	Left to right	Level-3
Shift	<< , >>	Left to right	Level-4
Relational	<, <=, >, >=	Left to right	Level-5

Rules for evaluation of expression

• First parenthesized sub expressions are evaluated first.

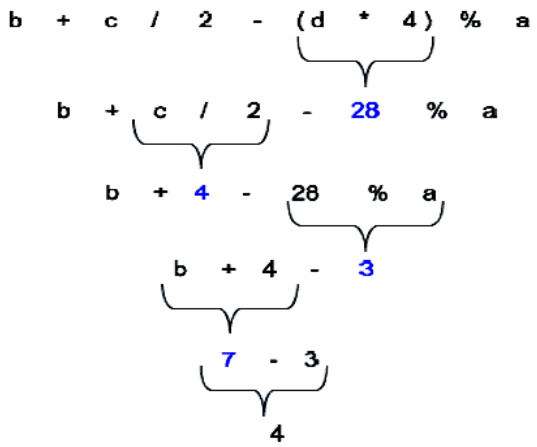
• If parentheses are nested, the evaluation begins with the innermost sub expression.

• The precedence rule is applied to determine the order of application of operators in evaluating sub expressions.

• The associability rule is applied when two or more operators are having same precedence in the sub expression.

Evaluate the following expression:

- Using a = 5, b = 3, c = 8 and d = 7



Classification of Operators

The operators in C are classified on the basis of

1) The number of operands on which an operator operates
2) The role of an operator

Classification based on Number of Operands

Types:

1. **Unary Operator**: A unary operator operates on only one operand. Some of the unary operators are,

Operator	Meaning
-	Minus
++	Increment
--	Decrement
&	Address- of operator
sizeof	sizeof operator

2. **Binary Operator**: A binary operator operates on two operands. Some of the binary operators are,

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modular Division
&&	Logical AND

3. **Ternary Operator**

A ternary operator operates on 3 operands. Conditional operator (i.e. **?:**) is the ternary operator.

Classification based on role of operands

Based upon their role, operators are classified as,

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Miscellaneous Operators
7.

1. **Arithmetic Operators**

They are used to perform arithmetic operations like addition, subtraction, multiplication, division etc.

A=10 & B=20

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	The division operator is used to find the quotient.	B / A will give 2
%	Modulus operator is used to find the remainder	B%A will give 0
+,-	Unary plus & minus is used to indicate the algebraic sign of a value.	+A, -A

Increment operator

The operator ++ adds one to its operand.

- ++a or a++ is equivalent to a=a+1
- Prefix increment (++a) operator will increment the variable BEFORE the expression is evaluated.
- Postfix increment operator (a++) will increment AFTER the expression evaluation.
- E.g.
  1. c=++a. Assume a=2 so c=3 because the value of a is incremented and then it is assigned to c.
  2. d=b++ Assume b=2 so d=2 because the value of b is assigned to d before it is incremented.

Decrement operator

The operator – subtracts one from its operand.

- --a or a-- is equivalent to a=a+1
- Prefix decrement (--a) operator will decrement the variable BEFORE the expression is evaluated.
- Postfix decrement operator (a--) will decrement AFTER the expression evaluation.
- E.g.
  1. c=--a. Assume a=2 so c=1 because the value of a is decremented and then it is assigned to c.
  2. d=b-- Assume b=2 so d=2 because the value of b is assigned to d before it is decremented.

2. Relational Operators

- Relational operators are used to compare two operands. There are 6 relational operators in C, they are

Operator	Meaning of Operator	Example
==	Equal to	5==3 returns false (0)
>	Greater than	5>3 returns true (1)
<	Less than	5<3 returns false (0)
!=	Not equal to	5!=3 returns true(1)
>=	Greater than or equal to	5>=3 returns true (1)
<=	Less than or equal to	5<=3 return false (0)

- If the relation is true, it returns value 1 and if the relation is false, it returns value 0.
- An expression that involves a relational operator is called as a condition. For e.g a<b is a condition.

3. Logical Operators

- Logical operators are used to logically relate the sub-expressions. There are 3 logical operators in C, they are
- If the relation is true, it returns value 1 and if the relation is false, it returns value 0.

Operator	Meaning of Operator	Example	**Description**
&&	Logial AND	If c=5 and d=2 then,((c==5) && (d>5)) returns false.	It returns true when both conditions are true
||	Logical OR	If c=5 and d=2 then, ((c==5) || (d>5)) returns true.	It returns true when at-least one of the condition is true
!	Logical NOT	If c=5 then, !(c==5) returns false.	It reverses the state of the operand

Truth tables of logical operations

condition 1 (e.g., X)	**condition 2 (e.g., Y)**	**NOT X (~ X)**	**X AND Y (X && Y)**	**X OR Y (X || Y)**
False	false	true	false	false
False	true	true	false	true
true	false	false	false	true
true	true	false	true	true

4. Bitwise Operators

C language provides 6 operators for bit manipulation. Bitwise operator operates on the individual bits of the operands. They are used for bit manipulation.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Truth tables

p	**Q**	**p & q**	**p | q**	**p ^ q**
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0

1	0	0	1	1

E.g.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bit Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

3 << 2 (Shift Left)
0011
1100=12

3 >> 1 (Shift Right)
0011
0001=1

5. Assignment Operators

To assign a value to the variable assignment operator is used.

Operators		Example	Explanation
Simple assignment operator	=	sum=10	10 is assigned to variable sum
Compound assignment operators Or Shorthand assignment operators	+=	sum+=10	This is same as sum=sum+10
	-=	sum-=10	sum = sum-10
	*=	sum*=10	sum = sum*10
	/+	sum/=10	sum = sum/10
	%=	sum%=10	sum = sum%10
	&=	sum&=10	sum = sum&10
	^=	sum^=10	sum = sum^10

```
E.g.
var=5  //5 is assigned to var
a=c;   //value of c is assigned to a
```

6. Miscellaneous Operators

Other operators available in C are

a. Function Call Operator(i.e. ( ))
b. Array subscript Operator(i.e [ ])
c. Member Select Operator
   i. Direct member access operator (i.e. **.** dot operator)
   ii. Indirect member access operator (i.e. -> arrow operator)

d. Conditional operator (?:)
e. Comma operator (,)
f. sizeof operator
g. Address-of operator (&)

a) Comma operator

- It is used to join multiple expressions together.
- E.g.

```
int i , j;
i=(j=10,j+20);
```

In the above declaration, right hand side consists of two expressions separated by comma. The first expression is j=10 and second is j+20. These expressions are evaluated from left to right. ie first the value 10 is assigned to j and then expression j+20 is evaluated so the final value of i will be 30.

b) sizeof Operator

- The sizeof operator returns the size of its operand in bytes.
- Example : size of (char) will give us 1.
- sizeof(a), where a is integer, will return 2.

c) Conditional Operator

- It is the only ternary operator available in C.
- Conditional operator takes three operands and consists of two symbols ? and : .
- Conditional operators are used for decision making in C.

Syntax :

(Condition? true_value: false_value);

For example:

```
c=(c>0)?10:-10;
```

If *c* is greater than 0, value of *c* will be 10 but, if *c* is less than 0, value of *c* will be -10.

d) Address-of Operator

It is used to find the address of a data object.

Syntax

&operand

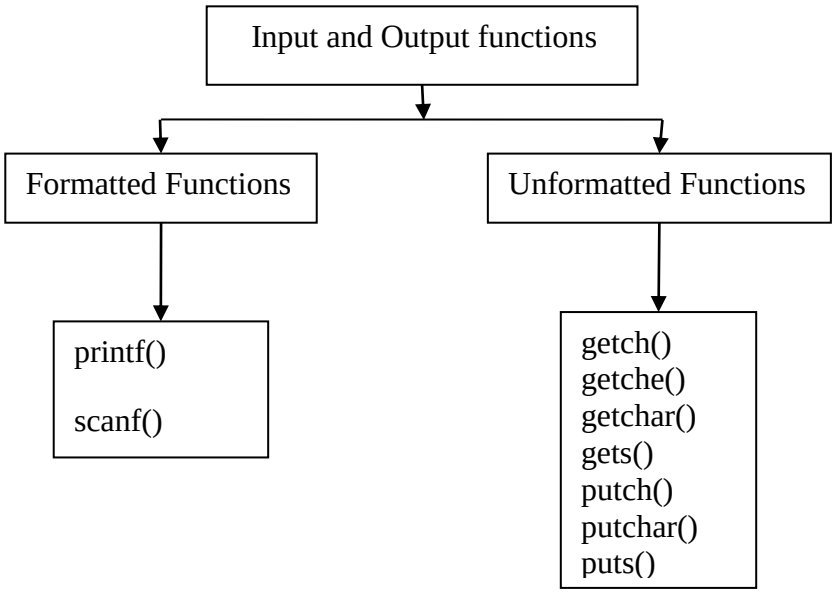
E.g.

&a will give address of a.

<u>Managing Input and Output operations</u>

The I/O functions are classified into two types:

- Formatted Functions
- Unformatted functions



Unformatted Functions:

They are used when I/P & O/P is not required in a specific format.

C has 3 types I/O functions.

a) Character I/O
b) String I/O
c) File I/O

a) Character I/O:

1. ***getchar()*** This function reads a single character data from the standard input. (E.g. Keyboard)

Syntax :

```
variable_name=getchar();
```

eg:

```
char c;
c=getchar();
```

2. putchar() This function prints one character on the screen at a time.

Syntax :

```
putchar(variable name);
```

eg

```
char c='C';
putchar(c);
```

Example Program

```
#include <stdio.h>
main()
{
int i;
char ch;
ch = getchar();
putchar(ch);
}
```

Output:

A
A

3. getch() and getche() : getch() accepts only a single character from keyboard. The character entered through getch() is not displayed in the screen (monitor).

Syntax

```
variable_name = getch();
```

Like getch(), getche() also accepts only single character, but getche() displays the entered character in the screen.

Syntax

```
variable_name = getche();
```

4 putch(): putch displays any alphanumeric characters to the standard output device. It displays only one character at a time.

Syntax

```
putch(variable_name);
```

b) ***String I/O***

1.gets() – This function is used for accepting any string through stdin (keyboard) until enter key is pressed.

Syntax

```
gets(variable_name);
```

2. puts() – This function prints the string or character array.

Syntax

```
puts(variable_name);
```

3. cgets()- This function reads string from the console.

Syntax

```
cgets(char *st);
```

It requires character pointer as an argument.

4. cputs()- This function displays string on the console.

Syntax

```
cputs(char *st);
```

Example Program

```
void main()
{
char ch[30];
clrscr();
printf("Enter the String : ");
gets(ch);
puts("\n Entered String : %s",ch);
puts(ch);
}
```

Output:

Enter the String : WELCOME
Entered String : WELCOME

Formatted Input & Output Functions

When I/P & O/P is required in a specified format then the standard library functions scanf( ) & printf( ) are used.

O/P function printf( )

The printf( ) function is used to print data of different data types on the console in a specified format.

General Form

```
printf("Control String", var1, var2, …);
```

Control String may contain

1. Ordinary characters
2. Conversion Specifier Field (or) Format Specifiers
   They denoted by %, contains conversion codes like %c, %d etc. and the optional modifiers width, flag, precision, size.

Conversion Codes

Data type	Conversion Symbol
char	%c
int	%d
float	%f
Unsigned octal	%o
Pointer	%p

Width Modifier: It specifies the total number of characters used to display the value.
Precision: It specifies the number of characters used after the decimal point.
E.g: printf(" Number=%7.2\n",5.4321);
Width=7
Precesion=2
Output: Number = 5.43(3 spaces added in front of 5)
Flag: It is used to specify one or more print modifications.

Flag	Meaning
-	Left justify the display
+	Display +Ve or –Ve sign of value
Space	Display space if there is no sign
0	Pad with leading 0s

E.g: printf("Number=%07.2\n",5.4321)
Output: Number=0005.43

Size: Size modifiers used in printf are,

Size modifier	Converts To
l	Long int
h	Short int
L	Long double

%ld means long int variable
%hd means short int variable

3. Control Codes

They are also known as Escape Sequences.

E.g:

Control Code	Meaning
\n	New line
\t	Horizontal Tab
\b	Back space

Input Function scanf( )

It is used to get data in a specified format. It can accept data of different data types.

Syntax

```
scanf("Control String", var1address, var2address, …);
```

Format String or Control String is enclosed in quotation marks. It may contain

1. White Space
2. Ordinary characters
3. Conversion Specifier Field

It is denoted by % followed by conversion code and other optional modifiers E.g: width, size.

Format Specifiers for scanf( )

Data type	Conversion Symbol
char	%c
int	%d
float	%f
string	%s

E.g Program:

```
#include <stdio.h>
void main( )
{
   int num1, num2, sum;
   printf("Enter two integers: ");
   scanf("%d %d",&num1,&num2);
   sum=num1+num2;
   printf("Sum: %d",sum);
}
```

Output

```
Enter two integers: 12 11
Sum: 23
```

Decision making statements - Switch statement – Looping statements

Decision making statements in a programming language help the programmer to transfer the control from one part to other part of the program.

Flow of control: The order in which the program statements are executed is known as flow of control. By default, statements in a c program are executed in a sequential order.

The default flow of control can be altered by using flow control statements. These statements are of two types.

1.Branching
- Selection-Conditional Branching
- Jump-unconditional Branching

2.Iteration statements

Branching statements

Branching statements are used to transfer program control from one point to another.

2 types

a) **Conditional Branching:**- Program control is transferred from one point to another based on the result of some condition
Eg) if, if-else, switch

b) **Unconditional Branching:-** Pprogram control is transferred from one point to another without checking any condition
Eg) goto, break, continue, return

a) Conditional Branching : Selection statements

Statements available in c are

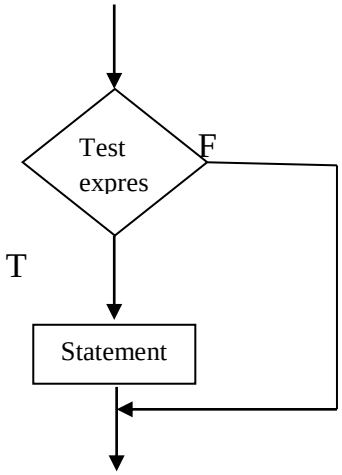
- The **if** statement
- The **if-else** statement
- The **switch case** statement

(i)The if statement

C uses the keyword **if** to execute a statement or set of statements when the logical condition is true.

Syntax:

```
if (test expression)
Statement;
```



- If test expression evaluates to true, the corresponding statement is executed.
- If the test expression evaluates to false, control goes to next executable statement.
- The statement can be single statement or a block of statements. Block of statements must be enclosed in curly braces.

Example:

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

```
printf("enter the number:");
scanf("%d",&n);
if(n>0)
printf("the number is positive");
getch();
}
```

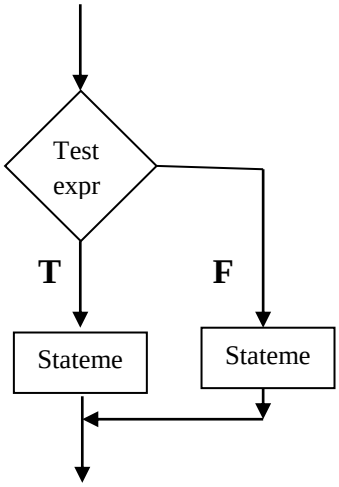
Output:

enter the number:50
the number is positive

(ii)The if-else statement

Syntax:

```
if (test expression)
Statement T;
else
Statement F;
```



- If the test expression is true, statementT will be executed.
- If the test expression is false, statementF will be executed.
- StatementT and StatementF can be a single or block of statements.

Example:1 Program to check whether a given number is even or odd.

```
#include <stdio.h>
void main()
{
int n,r;
clrscr();
printf("Enter a number:");
scanf("%d",&n);
r=n%2;
if(r==0)
        printf("The number is even");
else
        printf("The number is odd");
getch();
}
```

Output:

Enter a number:15
The number is odd

Example:2 To check whether the two given numbers are equal

```
void main()
{
int a,b;
clrscr();
```

```
printf("Enter a and b:" );
scanf("%d%d",&a,&b);
if(a==b)
        printf("a and b are equal");
else
        printf("a and b are not equal");
getch();
}
```

Output:

```
Enter a and b: 2 4
a and b are not equal
```

(iii) Nested if statement

If the body the if statement contains another if statement, then it is known as nested if statement

Syntax:

```
if (test expression)
        if (test expression)
```

(Or)

```
if (test expression)
{
        statement;
        if (test expression)
        ….
        statement;
}
```

any level.

Syntax:

```
if (test expression1)
{
      …..
     if (test expression2)
      {
       ….
       if (test expression3)
            {
           ….
           if (test expression n)
}}}
```

This is known as if ladder

iv) Nested if-else statement

Here, the if body or else body of an if-else statement contains another if statement or if else statement

Syntax:

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

```
statement 3;
if (condition)
statementnest;
statement 4;
}
```

Example:
Program for finding greatest of 3 numbers

```
#include<stdio.h>
void main()
{
int a,b,c;
printf("Enter three numbers\n");
scanf("%d%d%d",&a,&b,&c);
if(a>b)
{
        if(a>c)
        {
        printf("a is greatest");
        }

}
else
{
        if(b>c)
        {
        printf("b is greatest");
        }
        else
        {
        printf("C is greatest");
        }
}
}
```

Output
Enter three numbers 2 4 6
C is greatest

v) Switch statement

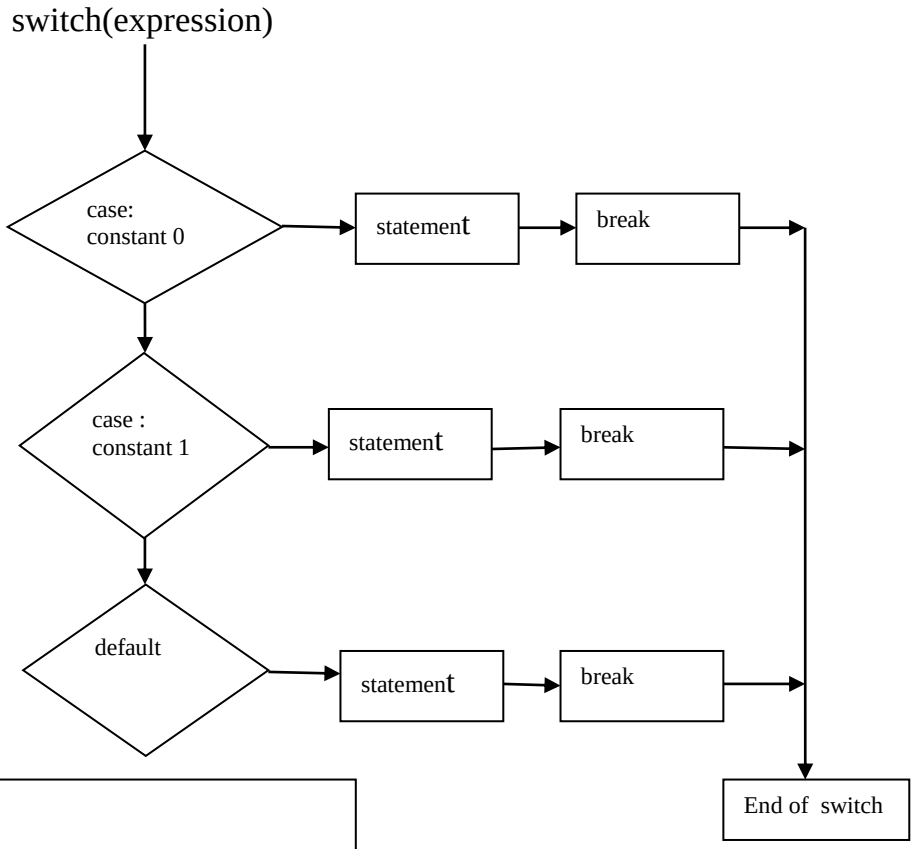
- It is a multi way branch statement.
- It provides an easy & organized way to select among multiple operations depending upon some condition.

Execution

1. Switch expression is evaluated.
2. The result is compared with all the cases.
3. If one of the cases is matched, then all the statements after that matched case gets executed.
4. If no match occurs, then all the statements after the default statement get executed.

- Switch ,case, break and default are keywords
- Break statement is used to exit from the current case structure

Flowchart



Syntax

```
switch(expression)
        {
        case value 1:
                        program statement;
                        program statement;
                        ……
                        break;
        case value 2:
                        program statement;
                        Program statement;
                        ……
                        break;
        …
        case value n:
                        program statement;
                        program statement;
                        ……
                        break;
        default:
                        program statement;
                        program statement;
}
```

Example1

```
void main()
{
char ch;
printf("Enter a character\n");
scanf("%c",&ch);
```

```
switch(ch)
        {
                case 'A':
                        printf("you entered an A\n");
                        break;
                case 'B':
                        printf("you entered a B\n");
                        break;
                default:
                        printf("Illegal entry");
                        break;
        }
  getch();
}
```

Output

```
Enter a character
A
You entered an A
```

Example2

```
void main()
{
int n;
printf("Enter a number\n");
scanf("%d",&n);
switch(n%2)
        {
                case 0:printf("EVEN\n");
                        break;
                case 1:printf("ODD\n");
                        break;
        }
getch();
}
```

Output:

```
Enter a number
5
ODD
```

b)Unconditional branching statements

i)The goto Statement

This statement does not require any condition. Here program control is transferred to another part of the program without testing any condition.

Syntax:

```
goto label;
```

- label is the position where the control is to be transferred.

Example:

```
void main()
{
printf("www.");
goto x;
y:
printf("mail");
goto z;
x:
printf("yahoo");
goto y;
z:
printf(".com");
getch();
}
```

Output: www.yahoomail.com

b)break statement

- A break statement can appear only inside a body of , a switch or a loop
- A break statement terminates the execution of the nearest enclosing loop or switch.

Syntax

```
break;
```

Example:1

```
#include<stdio.h>
void main()
{
int c=1;
while(c<=5)
{
        if (c==3)
           break;
        printf("\t %d",c);
        c++;
}
}
```

Output : 1 2

Example :2

```
#include<stdio.h>
void main()
{
int i;
for(i=0;i<=10;i++)
{
  if (i==5)
```

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

Output :1 2 3 4

iii) continue statement

- A continue statement can appear only inside a loop.
- A continue statement terminates the current iteration of the nearest enclosing loop.

Syntax:

```
continue;
```

Example :1

```
#include<stdio.h>
void main()
{
int c=1;
while(c<=5)
{
  if (c==3)
    continue;
  printf("\t %d",c);
  c++;
}
}
```

Output : 1 2 4 5

Example :2

```
#include<stdio.h>
main()
{
int i;
for(i=0;i<=10;i++)
{
  if (i==5)
    continue;
  printf(" %d",i);
}
}
```

Output :1 2 3 4 6 7 8 9 10

iv) return statement:

A return statement terminates the execution of a function and returns the control to the calling function.

Syntax:

```
return;
```

(or)

```
return expression;
```

Iteration Statements (Looping Statements)

Iteration is a process of repeating the same set of statements again and again until the condition holds true.

Iteration or Looping statements in C are:

1. for
2. while
3. do while

Loops are classified as

- ❖ Counter controlled loops
- ❖ Sentinel controlled loops

Counter controlled loops

- The number of iterations is known in advance. They use a control variable called loop counter.
- Also called as definite repetitions loop.
- E.g: for

Sentinel controlled loops

- The number of iterations is not known in advance. The execution or termination of the loop depends upon a special value called sentinel value.
- Also called as indefinite repetitions loop.
- E.g: while

1. for loop

It is the most popular looping statement.

Syntax:

```
for(initialization;condition2;incrementing/updating)
{
        Statements;
}
```

There are three sections in for loop.

a. **Initialization section –** gives initial value to the loop counter.
b. **Condition section –** tests the value of loop counter to decide whether to execute the loop or not.
c. **Manipulation section -** manipulates the value of loop counter.

Execution of for loop

1. Initialization section is executed only once.
2. Condition is evaluated
   a. If it is true, loop body is executed.
   b. If it is false, loop is terminated
3. After the execution of loop, the manipulation expression is evaluated.
4. Steps 2 & 3 are repeated until step 2 condition becomes false.

Ex 1: Write a program to print 10 numbers using for loop

```
void main()
{
int i;
clrscr();
for(i=1;i<=10;i++)
{
        printf("%d",i);
}
getch();
}
```

Output:

1 2 3 4 5 6 7 8 9 10

Ex2: To find the sum of n natural number. 1+2+3…+n

```
void main()
{
int n, i, sum=0;
clrscr();
printf("Enter the value for n");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
        sum=sum+i;
}
printf("Sum=%d", sum);
getch();
}
```

Output:

```
Enter the value for n
4
sum=10
```

2. while statement

- They are also known as Entry controlled loops because here the condition is checked before the execution of loop body.

Syntax:

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

Execution of while loop

a. Condition in while is evaluated
   i. If it is true, body of the loop is executed.
   ii. If it is false, loop is terminated
b. After executing the while body, program control returns back to while header.
c. Steps a & b are repeated until condition evaluates to false.
d. Always remember to initialize the loop counter before while and manipulate loop counter inside the body of while.

Ex 1: Write a program to print 10 numbers using while loop

```
void main()
{
int i=1;
clrscr();
while (num<=10)
{
        printf (“%d”,i);
        i=i+1;
}
getch();
}
```

Output:

1 2 3 4 5 6 7 8 9 10

Ex2: To find the sum of n natural number. 1+2+3…+n

```
void main()
{
int n, i=1, sum=0;
clrscr();
printf(“Enter the value for n”);
scanf(“%d”,&n);
while(i<=n)
{
        sum=sum+i;
        i=i+1;
}
printf(“Sum=%d”, sum);
getch();
}
```

Output:

Enter the value for n

4

Sum=10

3. do while statement

- They are also known as Exit controlled loops because here the condition is checked after the execution of loop body.

Syntax:

```
do
{
statements;
}
while(expression);
```

Execution of do while loop

a. Body of do-while is executed.
b. After execution of do-while body, do-while condition is evaluated
    i. If it is true, loop body is executed again and step b is repeated.
    ii. If it is false, loop is terminated
c. The body of do-while is executed once, even when the condition is initially false.

Ex1: Write a program to print 10 numbers using do while loop

```
void main()
{
int i=1;
clrscr();
do
{
        printf ("%d",i);
        i=i+1;
} while (num<=10);
getch();
}
```

Output:

1 2 3 4 5 6 7 8 9 10

Ex2: To find the sum of n natural number. 1+2+3…+n

```
void main()
{
int n, i=1, sum=0;
clrscr();
printf("Enter the value for n");
scanf("%d",&n);
do
{
        sum=sum+i;
        i=i+1;
} while(i<=n);
printf("Sum=%d", sum);
getch();
}
```

Output:

Enter the value for n
4
Sum=10

Three main ingredients of counter-controlled looping

1. Initialization of loop counter
2. Condition to decide whether to execute loop or not.
3. Expression that manipulates the value of loop.

Nested loops

- If the body of a loop contains another iteration statement, then we say that the loops are nested.
- Loops within a loop are known as nested loop.
- Syntax

```
while(condition)
{
        while(condition)
        {
        Statements;
        }
Statements;
}
```