School of Computing

Department of Computer Science and Engineering

UNIT - II

Problem Solving Techniques with C and C++ - SCSA1104

UNIT II BASICS OF C PROGRAMMING

Introduction to C: Features of C - Structure of C program-Data Types-'C' Tokens-Input/output statements-Control Statement, Functions: – Types of Functions – Recursion. Algorithms: Reversing the digits of a number - Generation of Fibonacci sequence- Factorial Computation.

1. Introduction to C

C programming is a general-purpose, procedural programming language developed in 1972 by **Dennis M. Ritchie** at the Bell Telephone Laboratories to develop the UNIX operating system. It is machine-independent, structured programming language which is used extensively in various applications. Many later languages have borrowed syntax/features directly or indirectly from C language. Like syntax of Java, PHP, JavaScript, and many other languages are mainly based on C language. C++ is nearly a superset of C language (There are few programs that may compile in C, but not in C++).

1.1 History of C language

The root of all modern languages is 'ALGOL.' It was first introduced in 1960. 'ALGOL' introduced the concept of structured programming to the developer community. In 1967, a new computer programming language was announced called as 'BCPL' which stands for Basic Combined Programming Language. BCPL was designed and developed by Martin Richards, especially for writing system software. This was the era of programming languages. Just after three years, in 1970 a new programming language called 'B' was introduced by Ken Thompson that contained multiple features of 'BCPL.' This programming language was created using UNIX operating system at AT&T and Bell Laboratories. Both the 'BCPL' and 'B' were system programming languages.

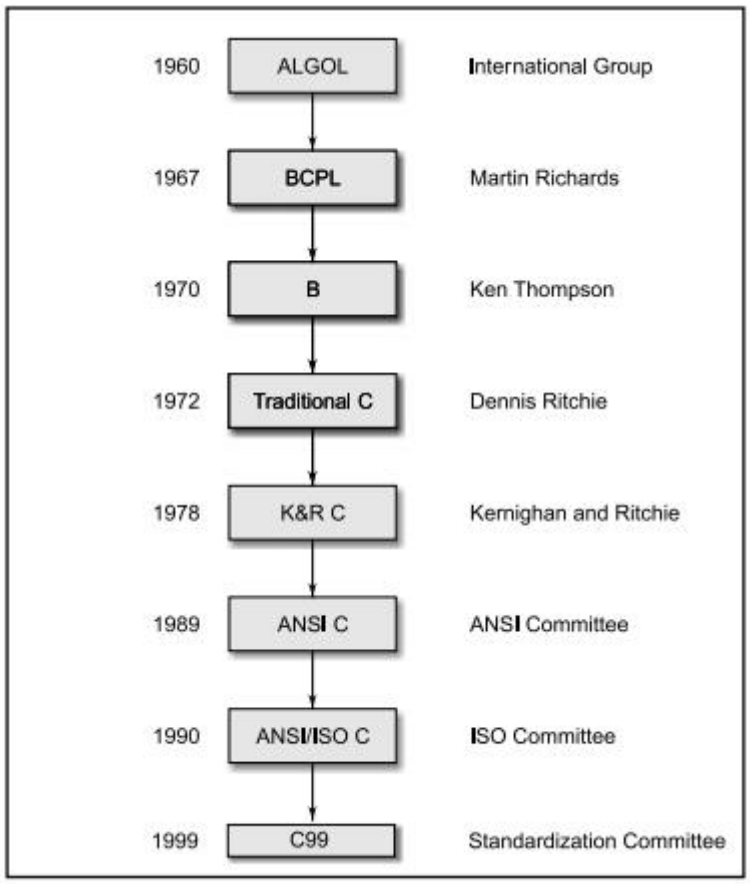


Fig. 1 History of C language

In 1972, a great computer scientist Dennis Ritchie created a new programming language called 'C' at the Bell Laboratories. It was created from 'ALGOL', 'BCPL' and 'B' programming languages. 'C' programming language contains all the features of these languages and many more additional concepts that make it unique from other languages.

American National Standards Institute (ANSI) defined a commercial standard for 'C' language in 1989. Later, it was approved by the International Standards Organization (ISO) in 1990. 'C' programming language is also called as 'ANSI C'. The standardization committee of C added a few features of C++/Java to enhance the usefulness of the language. The result was the 1999 standard for C. This version is usually referred to as C99. The history and development of C is illustrated in Fig. 1.

1.2 Where is C used? Key Applications

'C' language is widely used in embedded systems.

- ✓ It is used for developing system applications.
- ✓ It is widely used for developing desktop applications.
- ✓ Most of the applications by Adobe are developed using 'C' programming language.
- ✓ It is used for developing browsers and their extensions. Google's Chromium is built using 'C' programming language.
- ✓ It is used to develop databases. MySQL is the most popular database software which is built using 'C'.
- ✓ It is used in developing an operating system. Operating systems such as Apple's OS X, Microsoft's Windows, and Symbian are developed using 'C' language. It is used for developing desktop as well as mobile phone's operating system.
- ✓ It is used for compiler production.
- ✓ It is widely used in Internet of Things (IOT) applications.

2. Features of C

C is the widely used language. It provides many features that are illustrated in fig.2.

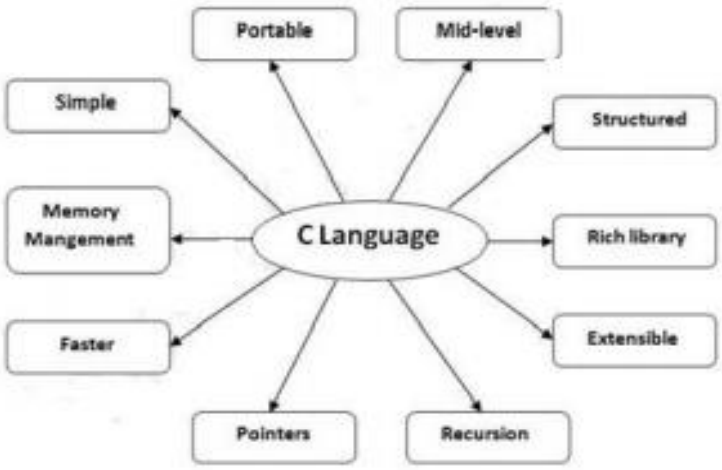


Fig 2. Features of C

2.1 Rich Library

It is a robust language whose rich set of built-in functions and operators can be used to write any complex program. C provides a lot of inbuilt functions that make the development fast.

2.2 Machine Independent or Portable

C is highly portable. This means that C programs written for one computer can be run on another with little or no modification. The compiler and preprocessor makes it possible to run on different computer or devices

2.3 Mid-level programming language

The C compiler combines the capabilities of an assembly language with the features of a high-level language and therefore it is well suited for writing both system software and business packages. In fact, many of the C compilers available in the market are written in C.

2.4 Structured programming language.

C language is well suited for structured programming, thus requiring the user to think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. This modular structure makes program debugging, testing and maintenance easier.

2.5 Recursion

In C, we can call the function within the function. It provides code reusability for every function.

2.6 Fast

Programs written in C are efficient and fast. This is due to its variety of data types, functions and powerful operators. It is many times faster than BASIC. For example, a program to increment a variable from 0 to 15000 takes about one second in C while it takes more than 50 seconds in an interpreter BASIC.

2.7 Memory Management

It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free() function.

2.8 Pointer

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array, etc.

2.9 Easy to extend

C is also an extensible language. Another important feature of C program, is its ability to extend itself. A C program is basically a collection of functions that are supported by the C library. We can continuously add our own functions to C library.

3. Structure of a C program

C program can be written in this structure only. Writing a C program in any other structure will hence lead to a Compilation Error. The structure of a C program is illustrated in figure 3.

3.1 Document Section

- ✓ A comment is an explanation or description of the source code of the program. It helps a developer explain logic of the code and improves program readability.
- ✓ At run-time, a comment is ignored by the compiler.

In C, we can use comment in two ways

1) Single line comment

2) Multi Line Comment

Example :

```
// Single line comment

//Multi Line Comment

//Line 1

//Line 2

(Or)

/* Multi line comment are used in the program

Line 1…
Line 2….
…..,
……
*/
```

3.2 Preprocessor Section or Link Section or Header File Section

- ✓ The link section provides instructions to the compiler to link functions from the library.
- ✓ A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files.

Some of C Header files:

- stdio.h – Defines core input and output functions
- string.h – Defines string handling functions
- math.h – Defines common mathematical functions

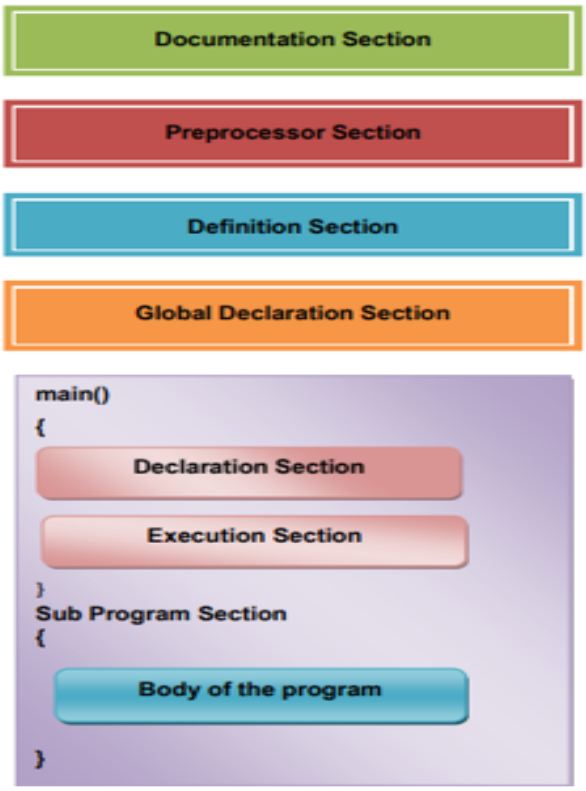


Fig 3. Structure of C program

Syntax to include a header file in C:

#include<filename>

Where,

Filename is the name of the library file that contains the required function definition.

Example:

#include<stdio.h>

#include<math.h>

#include<string.h>

3.3 Definition Section

- ✓ The definition section defines all symbolic constant.
- ✓ Symbolic constant is a way of defining a variable constant whose value cannot be changed

Syntax:

#define symbolic constant value

Example:

#define MAX 100

#define PI 3.14

3.4 Global Declaration Section:

- ✓ The variables that are used in more than one function throughout the program are called global variables.
- ✓ Should be declared outside of all the functions i.e., before main().

3.5 Main Method Declaration:

The next part of a C program is to declare the main() function. Every 'C' program must have one main() function where the program execution begins.

Syntax:

return type main()

{}

Example:

int main()

{}

void main()

{}

It contains the following two parts

1. Declaration Part
2. Executable part

3.5.1 Declaration Part

It refers to the variables that are to be used in the function. The variables are to be declared before

any operation in the function and these are called local variables

Syntax

Data type variable name;

Example:

```
void main()
{
int a;
}
```

3.5.2 Execution Part:

It contains at least one valid C Statement. The Execution of a program begins with opening brace '{' and ends with closing brace '}' .The closing brace of the main function is the logical end of the program.

3.6 Sub Program section

- ✓ Sub programs are basically functions are written by the user (user defined functions).
- ✓ They may be written before or after a main () function and called within main () function.
- ✓ This is optional to the programmer.

Constraints while writing a C program

- All statements in 'C' program should be written in lower case letters. Uppercase letters are only used for symbolic constants
- Blank space may be inserted between the words. Should not be used while declaring a variable, keyword, constant and function
- The program statements can be written anywhere between the two braces following the declaration part.
- All the statements should end with a semicolon (;)

3.7 Example to illustrate structure of C program

```
// Program to illustrate structure of C program  // Document section  //Single line comment
        #include<stdio.h>                // Link Section
        #define PI 3.14;                // Definition Section
        float area(float r);            // Global Declaration section
        int main()                       //Main Function
        {
        float r;                        // Declaration Part
        printf(" Enter the radius:n");  // Executable Part
        scanf("%f",&r);
        printf("the area is: %f",area(r));
        return 0;
        }
        float area(float r)
        {
        return pi * r * r;               //sub program or User defined function
        }
```

4. Compilation and Execution of C program

1. Creating the program
2. Compiling the Program
3. Linking the Program with system library
4. Executing the program

4.1 Creating the program:

✓ Type the program and edit it in standard ‘C’ editor and save the program with .c as an e

✓ xtension.

✓ This is the source program.

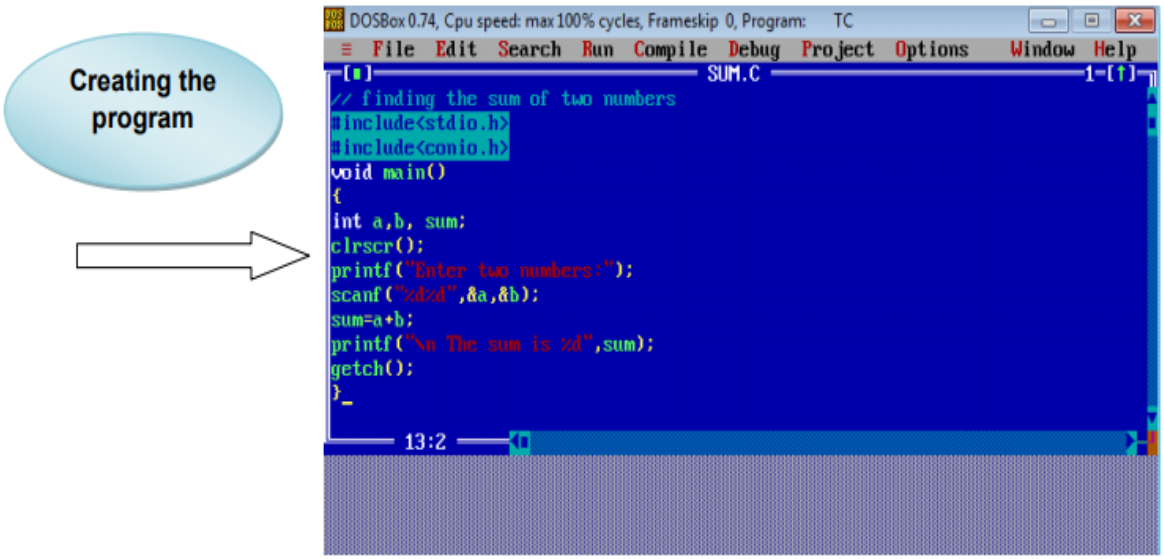


Fig.4 Creating the program

4.2 Compiling (Alt + F9) the Program:

- This is the process of converting the high level language program to Machine level
- Language (Equivalent machine instruction) -> Compiler does it!
- Errors will be reported if there is any, after the compilation
- Otherwise the program will be converted into an object file (.obj file) as a result of the compilation
- After error correction the program has to be compiled again

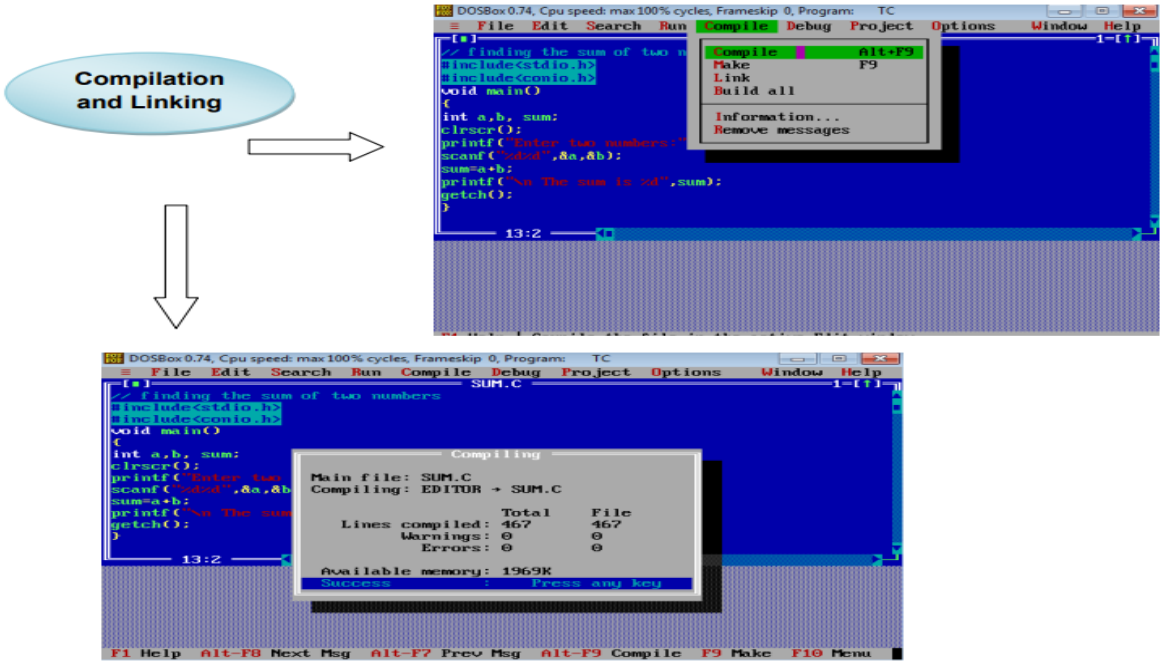


Fig. 5 Compiling and Linking the program

4.3 Linking the program with system Library

✓ Before executing a c program, it has to be linked with the included header files and other system libraries -> Done by the Linker

4.4 Executing the Program:

✓ This is the process of running (Ctrl + F9) and testing the program with sample data. If there are any run time errors, then they will be reported.

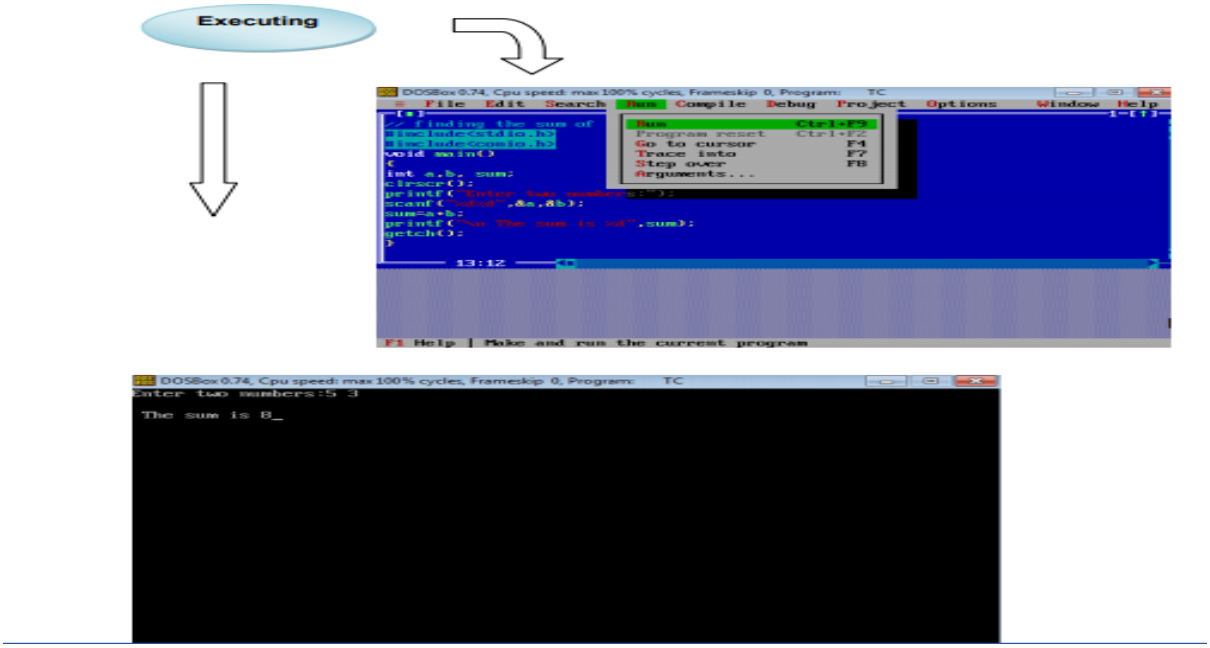


Fig. 6 Executing the program

5. 'C' TOKENS

In a passage of text, individual words and punctuation marks are called tokens. Similarly, in a C program the smallest individual units are known as C tokens. We cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language. C has six types of tokens as shown in Fig. 7. C programs are written using these tokens and the syntax of the language.

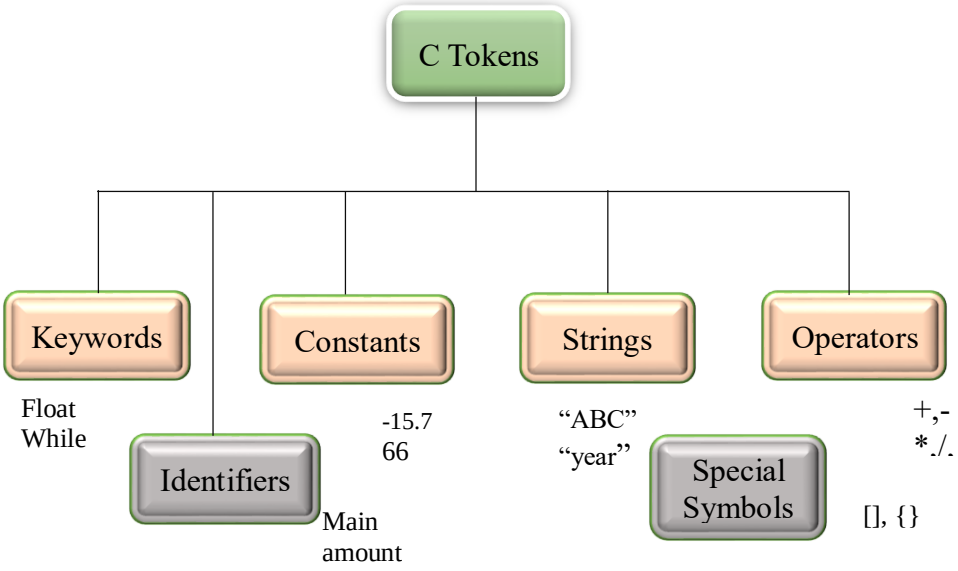


Fig.7. Tokens

5.1 Keywords

- ✓ Every C word is classified as either a keyword or an identifier. All keywords have fixed meanings and these meanings cannot be changed.
- ✓ Keywords serve as basic building blocks for program statements.
- ✓ All keywords must be written in lowercase.
- ✓ Keywords cannot be used as normal identifiers names.

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

Table 2. ANSI C Keywords

5.2 Identifiers

- ✓ Identifiers refer to the names of variables, functions and arrays. These are user-defined names and consist of a sequence of letters and digits, with a letter as a first character.
- ✓ Both uppercase and lowercase letters are permitted.
- ✓ The underscore character is also permitted in identifiers.

Rules for Identifiers

1. First character must be an alphabet (or underscore).
2. Must consist of only letters, digits or underscore.
3. Cannot use a keyword.
4. Must not contain white space

Example :

int num;

char name;

Sum() // where num, name and sum are identifiers

5.3 Constants

- ✓ Constants in C refer to fixed values that do not change during the execution of a program.
- ✓ C supports several types of constants as illustrated in Fig.8.

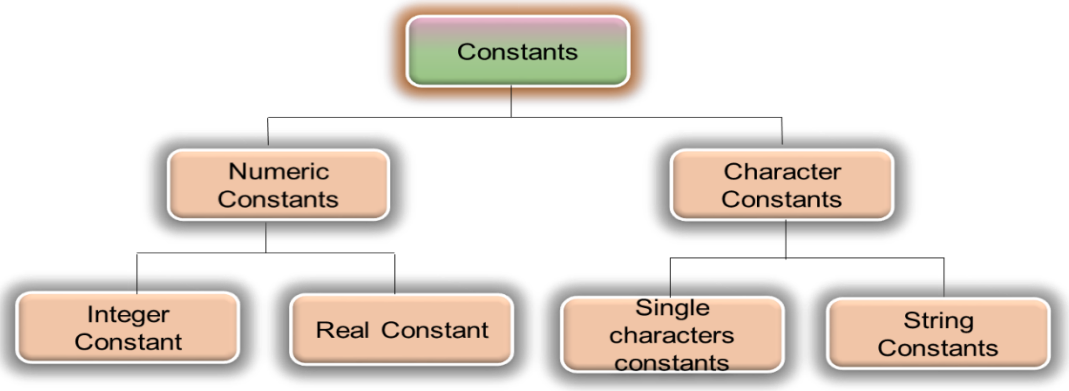


Fig.8. Constants

5.3.1 Integer Constant:

- ✓ An integer constant refers to a sequence of digits.

Example:

123

124 – 321 0 654321 +78

5.3.2 Real or floating point constants:

- ✓ Integer numbers are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices, and so on.
- ✓ These quantities are represented by numbers containing fractional parts like 17.548.
- ✓ Such numbers are called real (or floating point) constants.

Example:

0.0083 –0.75 435.36 +247.0

5.3.3 Single character constants

- ✓ A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks.

Example

'5' 'X' ';' ' '

5.3.4 String constants

- ✓ A string constant is a sequence of characters enclosed in double quotes. The characters may be letters, numbers, special characters and blank space.

Examples

"Hello!" "1987" "WELL DONE" "?...!" "5+3" "X"

Backslash character constants

- ➢ C supports some special backslash character constants that are used in output functions.
- ➢ For example, the symbol '\n' stands for newline character.
- ➢ Each one of them represents one character, although they consist of two characters.
- ➢ These characters combinations are known as escape sequences.

Constant	*Meaning*
'\a'	audible alert (bell)
'\b'	back space
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
'\"'	double quote
'\?'	question mark
'\\'	backslash
'10'	null

Table 1. Backslash character constants

5.4 Strings in C

Strings in C are always represented as **an array of characters** having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are **enclosed within double quotes,** while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

char a[10] = "javatpoint"; // The compiler allocates the 10 bytes to the 'a' array.

char a[] = "javatpoint"; // The compiler allocates the memory at the run time.

char a[10] = {'j','a','v','a','t','p','o','i','n','t','\0'}; // String is represented in the form of characters.

5.5 Operators in C

C supports a rich set of built-in operators. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. The data items on which the operators are applied are known as operands. Operators are applied between the operands Operators are used in programs to manipulate data and variables. Depending on the number of operands, operators are classified as follows:

- ✓ Unary Operator
- ✓ Binary Operator

5.5.1 Unary Operator

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.

5.5.2 Binary Operator

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

1. Arithmetic operators

2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditional operators
7. Bitwise operators
8. Special operators

5.5.2.1 Arithmetic Operators

✓ C provides all the basic arithmetic operators such as +, –, *, and /

✓ Integer division truncates any fractional part.

✓ The modulo division operation produces the remainder of an integer division.

Examples

a – b a / b

a + b a % b

a * b –a * b

Arithmetic Operators can be classified as:

1. Integer Arithmetic
2. Real Arithmetic
3. Mixed mode Arithmetic

Operator	*Meaning*
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Table 2. Arithmetic operators

5.5.2.1.1 Integer arithmetic

When both the operands in a single arithmetic expression such as a+b are integers, the expression is called as integer expression and the operation is called integer arithmetic.

Example

a=14, b=4;

Operation	Output
a+b	18
a-b	10
a*b	56
a/b	3(decimal part truncated)
a%b	2(remainder of division)

Table 3. Arithmetic Operation

Here in the above example we are using only integer values.

5.5.2.1.2 Real arithmetic

- ✓ An arithmetic which is involving real values (decimal values) is called real arithmetic.
- ✓ If x, y, and z are floats, then we will have:
  - x = 6.0/7.0 = 0.857143
  - y = 1.0/3.0 = 0.333333
  - z = –2.0/3.0 = –0.666667

5.5.2.1.3 Mixed mode arithmetic

When one operand is integer and another is real i.e both the types of variables (integer and real) values are used for operation is called as mixed more arithmetic.

E.g. Float a=10.5; Int b=30; C=a+b; //c=40.5

Eg. 15/10.0 =1.5

This type of expression that c=a+b is called mixed arithmetic.

Example

#include<stdio.h>

```
#include<conio.h>
main ()
  {
   int months, days ;
   printf("Enter days\n") ;
   scanf("%d", &days) ;
   months = days / 30 ;
   printf("Months = %d" , months) ;
  }
```

5.5.2.2 Relational operator

We often compare two quantities, and depending on their relation, take certain decision. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators.

Operator	*Meaning*
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Table 4: Relational Operators

An expression such as a<b or 1 < 20 containing a relational operator is termed as a relational expression. The value of a relational expression is either one or zero. It is one if the specified relation is true and zero if the relation is false.

Example 10 < 20 is true but 20 < 10 is false

When arithmetic expressions are used on either side of the relational operator, the arithmetic operator will be executed first and then the results will be compared. Relational operators are used in decision making statements.

Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a=10, b=20;
   if(a > b)
   {
      printf("A is Largest\n");
   }
      else
   {
         printf("B is Largest\n");
   } }
```

5.5.2.3 Logical operators

In addition to the relational operators, C has the following three logical operators.

&& meaning logical AND

|| meaning logical OR

! meaning logical NOT

op-1	*op-2*	*Value of the expression*	
		op-1 && op-2	*op-1 || op-2*
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Table 5 Truth Table Logical operators

The logical operators are used when we want to check more than one condition and make decision. An expression of this kind, which combines two or more relational expressions, is termed as a **logical expression** or a **compound relational expression**. Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table shown in Table 5. The logical expression given above is true only if a > b is true and x == 10 is

true. If either (or both) of them are false, the expression is false.

Example

1. if (age > 55 && salary < 1000)

2. if (number < 0 || number > 100)

Example Program

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a =20, b =10;
if(a>b && a>c)
{
 printf("a is greater");
 }
 else
 {
printf("a is not greater");
 }
```

It combines two or more relational expressions

5.5.2.4 Assignment operator

Assignment operators are used to assign the result of an expression to a variable.

Example

```
a=10
c=a+b;
```

C has a set of 'shorthand ' assignment operators of the form

v op= exp;

Where v is a variable, exp is an expression and op is a C binary arithmetic operator. The operator op= is known as the shorthand assignment operator.

The assignment statement

v op= exp;

is equivalent to

v = v op (exp);

with v evaluated only once.

Statement with simple assignment operator	*Statement with shorthand operator*
a = a + 1	a += 1
a = a – 1	a –= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

Table : 6 Statement with assignment and shorthand operator

The use of short hand operator has three advantages

- What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
- The statement is more concise and easier to read.
- The statement is more efficient.

Example Program

```
#include <stdio.h>
int main()
 {
    int number1, number2, sum;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);
    // calculating sum
```

```
    sum = number1 + number2;
    printf("%d  %d  %d", number1, number2, sum);
    return 0;
}
```

5.5.2.5 Increment and decrement operator

C has very useful operators not generally present in other languages. These are increment and decrement operators:

++ and - -

The operator ++ adds 1 to the operand while -- substracts 1. Both of them are unary operators.

++m is equivalent to m=m+1; (or m+=1)

- -m is equivalent to m=m- 1; (or m- =1)

We will be using this increment and decrement operators in for and while loop.

- ✓ A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left. Eg. ++var
- ✓ On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.eg. var++

Example:

Consider m=10

a=++m

a = 11

b = m++

b = 10

5.5.2.6 Conditional operator

The c language has an unusual operator, useful for making two way decisions. This operator is combination of ? and : and takes three operands. This operator is popularly known as conditional operator. The general syntax of this operator is.

Conditional expression ? expression1: expression2

Example:

Large=(num1>num2)?num1:num2

In the above example the largest number between num1 and num2 will be assigned to large. That is if num1 is larger then num2 num1 will be assigned to large. This is similar to if statement but the syntax differs.

Example:

```
#include <stdio.h>
int main()
{
   int a=5,b;  // variable declaration
   b=((a==5)?(3):(2)); // conditional operator
   printf("The value of 'b' variable is : %d",b);
    return 0;
}
```

5.5.2.7 Bitwise operators

C has distinction of supporting special operators known as bitwise operator for manipulating of data at bit level. These are used for testing the bits, or shifting them right or left.

Operator meaning

& bitwise AND

| bitwise OR

^ bitwise exclusive OR

<< shift left

>> shift right

~ one's complement

5.5.2.8 Special operators

C supports some special operators such as comma(,) operator, sizeof operator, pointer operator.

Comma operator can be used to link related expressions

Example:

```
int a,b, c=a+b;
```

sizeof operator is used to find the size of the variable or identifier.

Example:

M= sizeof (a);

Here in the above example the size of a will be assigned to M.

Pointer operator indicates the address of the variable.

Example:

int a;

p=&a;

Example:

//Program to illustrate the sizeof various data types

```
int main() {
    int intType;
    float floatType;
    double doubleType;
    char charType;
   // sizeof evaluates the size of a variable
    printf("Size of int: %zu bytes\n", sizeof(intType));
    printf("Size of float: %zu bytes\n", sizeof(floatType));
    printf("Size of double: %zu bytes\n", sizeof(doubleType));
    printf("Size of char: %zu byte\n", sizeof(charType));
     return 0;
}
```

Output:

Size of int: 4 bytes

Size of float: 4 bytes

Size of double: 8 bytes

Size of char: 1 byte

5.6 Data Types

- A data type specifies a type of data that a variable can store such as integer, float or character.
- ANSI C supports three classes of data types:
  - ❑ 1. Primary (or fundamental) data types
  - ❑ 2. Derived data types
  - ❑ 3. User-defined data types

Five fundamental data types:

- ❑ integer (int) - 2 byte
- ❑ character (char) -1 byte
- ❑ floating point (float) - 4 byte
- ❑ double-precision floating point (double) -8 byte
- ❑ Void
- ❑ The void type has no values. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function.

- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- **int:** As the name suggests, an int variable is used to store an integer. It requires 2 bytes of memory.

- **float:** It is used to store decimal numbers (numbers with floating point value) with single precision. It requires 4 bytes of memory.
- **double:** It is used to store decimal numbers (numbers with floating point value) with double precision. It requires 8 bytes of memory.

Data type	Range of values
char	–128 to 127
int	–32,768 to 32,767
float	3.4e–38 to 3.4e+e38
double	1.7e–308 to 1.7e+308

Table 8. Size and Range of data types on 16-bit machines

5.7 Variables

- ✓ A variable is a data name that may be used to store a data value.
- ✓ Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different time during execution.
- ✓ A variable name can be chosen by the programmer in a meaningful way.

Example

- Average
- Height
- Total
- Counter_1
- class_strength

variable names may consist of letters, digits, and the underscore(_) character

Conditions to write variable name

- ✓ They must begin with a letter.
- ✓ Length should not be normally more than eight characters, since only the first eight characters are treated as significant by many compilers.
- ✓ Uppercase and lowercase are significant.
- ✓ It should not be a keyword.
- ✓ White space is not allowed

5.7.1 Declaration of Variables

- ✓ After designing suitable variable names, we must declare them to the compiler.
- ✓ Declaration does two things:

1. It tells the compiler what the variable name is.

2. It specifies what type of data the variable will hold.

The declaration of variables must be done before they are used in the program.

Syntax:

data_type variable name; // For single variable declaration

e.g: int count;

data-type v1,v2,....vn ; // For multiple variable declaration

eg: int number, total;

float price, height;

5.7.2 Assigning values to the variables:

Values can be assigned to variables using the assignment operator = as follows:

Syntax:

variable_name = constant;

Example

a=10;

It is also possible to assign a value to a variable at the time the variable is declared.

Syntax:

data-type variable_name = constant;

Examples

int final_value = 100;

char yes = 's';

- The process of giving initial values to variables is called initialization.
- C permits the initialization of more than one variables in one statement using multiple assignment operators.

Example

p = q = s = 0;

x = y = z = MAX;

5.8 Input and Output Statements

Input means to provide the program with some data to be used in the program and Output means to display data on screen or write the data to a printer or a file. C programming la

nguage provides many built-in functions to read any given input and to display data on screen when there is a need to output the result.

The standard input-output header file, named stdio.h contains the definition of the functions printf() and scanf(), which are used to display output on screen and to take input from user respectively.

printf() statement

printf( ) is predefined ,standard C function for printing output.

The general form of printf() is,

printf("<format string>",<list of variables>);

<format string> could be,

%f for printing real values.

%d for printing integer values.

%c for printing character values.

Eg: printf("%f",si);

printf("%d",i);

scanf() statement

scanf( ) is a predefined,standard C function used to input data from keyboard.

The general form of scanf() is

scanf(“<format string>”,<address of variables>);

Eg: scanf(“%d”,&n);

scanf(“%f”,&interest);

& is pointer operator which specifies the address of that variable.

```
#include<stdio.h>
int main()
{
int number;
printf("enter a number:");
scanf("%d",&number);
printf("cube of number is:%d ",number*number*number);
return 0;
}
```

getchar() & putchar() functions

The getchar() function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. You can use this method in a loop in case you want to re

ad more than one character.

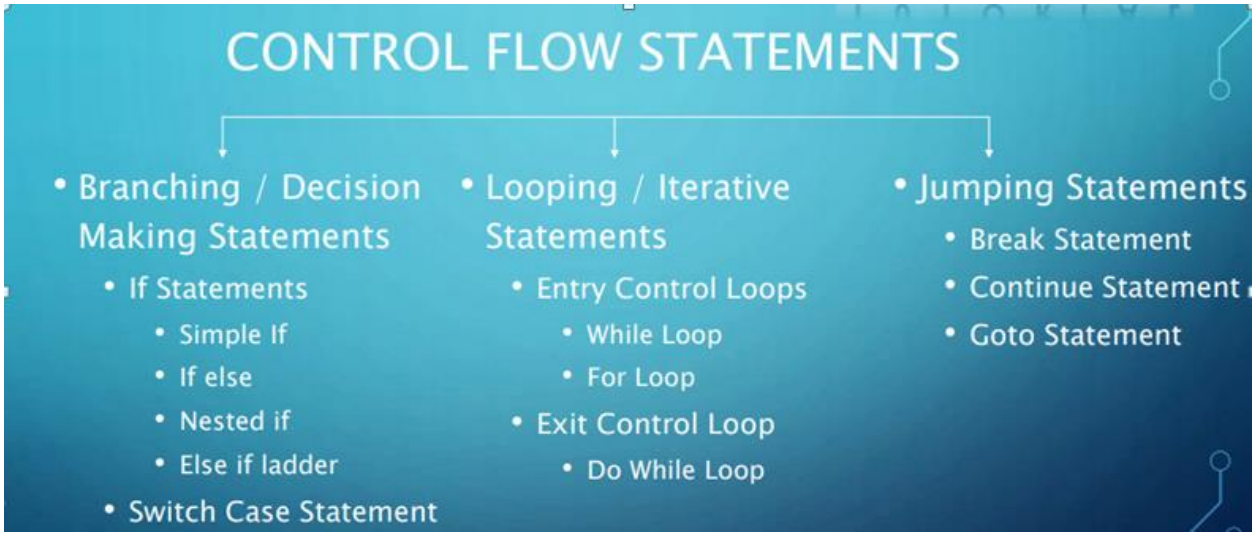
The putchar() function displays the character passed to it on the screen and returns the same character. This function too displays only a single character at a time. In case you want to display more than one characters, use putchar() method in a loop.

5.9 Control Statements

A control flow statements is a primary concept in most high level programming languages. It is a block of code that control the flow of a program.

There are three types of control flow statements:

- Branching / Decision Making Statements
- Iterative / Looping Statements
- Jumping Statements



5.9.1 Decision Making Statements

C program executes program sequentially. Sometimes, a program requires checking of certain conditions in program execution. C provides various key condition statements to check condition and execute statements according conditional criteria. These statements are called as '**Decision Making Statements**' or '**Conditional Statements**.'

C language possesses such decision-making capabilities by supporting the following statements:

1. if statement
2. switch statement
3. Conditional operator statement
4. goto statement

These statements are popularly known as decision-making statements. Since these statements 'control' the flow of execution, they are also known as control statements.

Decision Making with IF Statement

The if statement is a powerful decision-making statement and is used to control the flow of execution of statements. The different forms of if statement are:

1. Simple if statement
2. if.....else statement
3. Nested if....else statement
4. else if ladder.

5.9.1.1 Simple if statement

If the expression is true then the statement or block of statements gets executed otherwise these statements are skipped. The 'statement-block' may be a single statement or a group of statements. If the test expression is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x. Remember, when the condition is true both the statement-block and the statement-x are executed in sequence. This is illustrated in Fig. 9.

Syntax:

```
if(test expression)
  {
 Statement_block;
  }
statement–x;
```

Example:

```
#include<stdio.h>
void main()
{
  int a=5;
  if(a%2==0)
    {
        printf("number %d is even.",a);
    }}
```

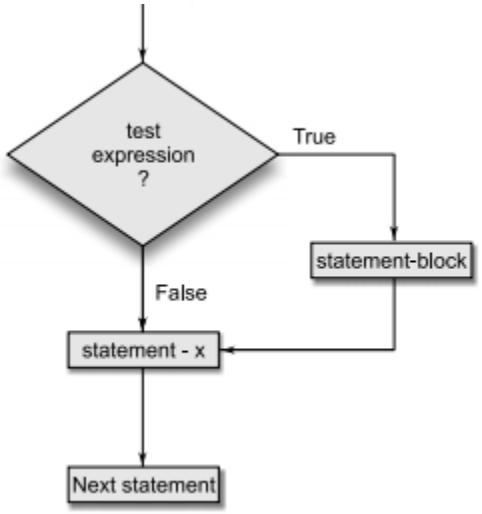


Fig.9. Flowchart of Simple if

5.9.1.2 If Else Statements

If the test expression is true, then the true-block statement(s), immediately following the if statements

are executed; otherwise, the false-block statement(s) are executed. In either case, either true-block or false-block will be executed, not both. This is illustrated in Fig. 10. In both the cases, the control is transferred subsequently to the statement-x.

Syntax:

```
if(test -Expression)
  {
    true statements;
  }
 else
  {
    false statements;
  }
Statement-x;
```

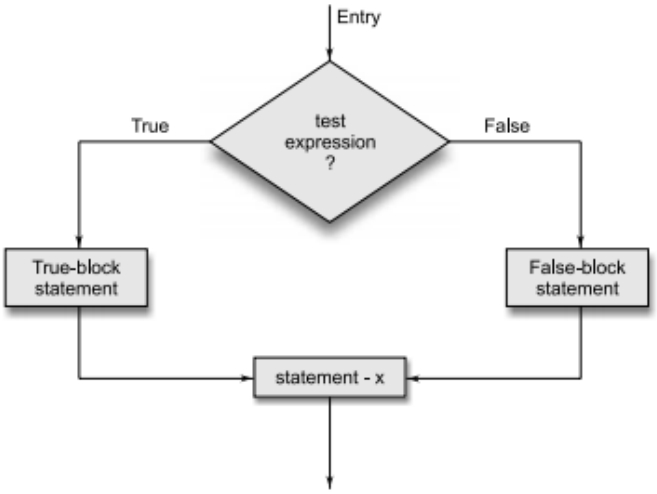


Figure : 10 Flowchart of If – Else

Example 1:

//Program to print the given number is odd or even

```
#include<stdio.h>
void main()
{
int a=5;
if(a%2==0)
{
printf("number %d is even",a);
}
Else
{
printf("number %d is odd",a);
}}
```

Example 2:

//Program to calculate gross salary

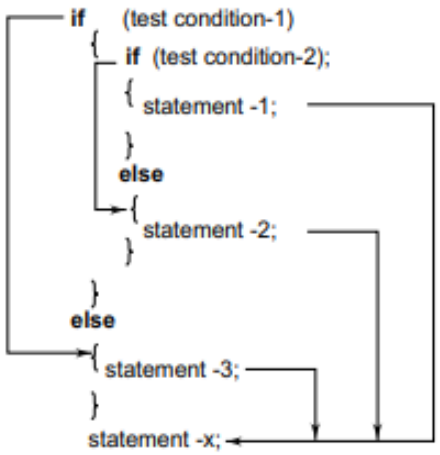
```
#include<stdio.h>
#inlcude<conio.h>
void main()
{
```

```
    float bs,gs,da,hra;
    printf("Enter basic salary");
    scanf("%f",&bs);
    if(bs<1500)
      {
        hra=bs*10/100;
        da=bs*90/100;
      }
    else
      {
        hra=500;
        da=bs*98/100;
      }
   gs=bs+hra+da;
   printf("gross salary =Rs %f\n",gs);
  return 0;
 }
```

5.9.1.3 Nested If Statements

When a series of decisions are involved, we may have to use more than one if...else statement in nested form as shown below:

Syntax:



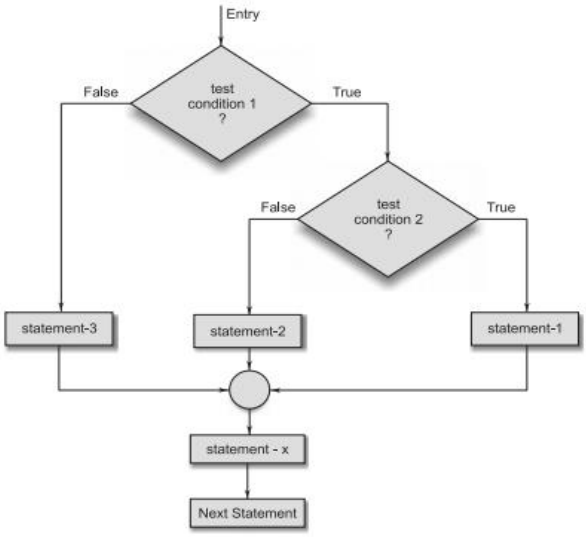


Figure 11. Flowchart of Nested if

The logic of execution is illustrated in Fig.11. If the condition-1 is false, the statement-3 will be executed;

otherwise it continues to perform the second test. If the condition-2 is true, the statement-1 will be evaluated; otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.

Example:

```
#include<stdio.h>
void main()
{
int a,b,c;
a=5, b=6, c=8;
if(a>b){
if(a>c){
printf("a is big");
}
else{
printf("c is big");
}
else{
if(b>c){
printf("b is big");
}
else{
printf("c is big");
}
getch();
}
}
```

Example 2:

A commercial bank has introduced an incentive policy of giving bonus to all its deposit holders. The policy is as follows: A bonus of 2 per cent of the balance held on 31st December is given to every one, irrespective of their balance, and 5 per cent is given to female account holders if their balance is more than Rs. 5000.

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

```
{
int balance,bonus;
char gender='f';
clrscr();
printf("Enter gender and balance\n");
scanf("%d %d",&gender,&balance);
if(gender=='f')
{
        if(balance>5000)
                bonus=0.05*balance;
        else
                bonus=0.02*balance;
}
else
{
        bonus=0.02*balance;
}
balance=balance=0.02*balance;
}
balance=balance+bonus;
printf("%d",balance);
getch();
}
```

5.9.1.4 Else if ladder

There is another way of putting ifs together when multipath decisions are involved. A multipath decision is a chain of ifs in which the statement associated with each else is an if. It takes the following general form:

Syntax:

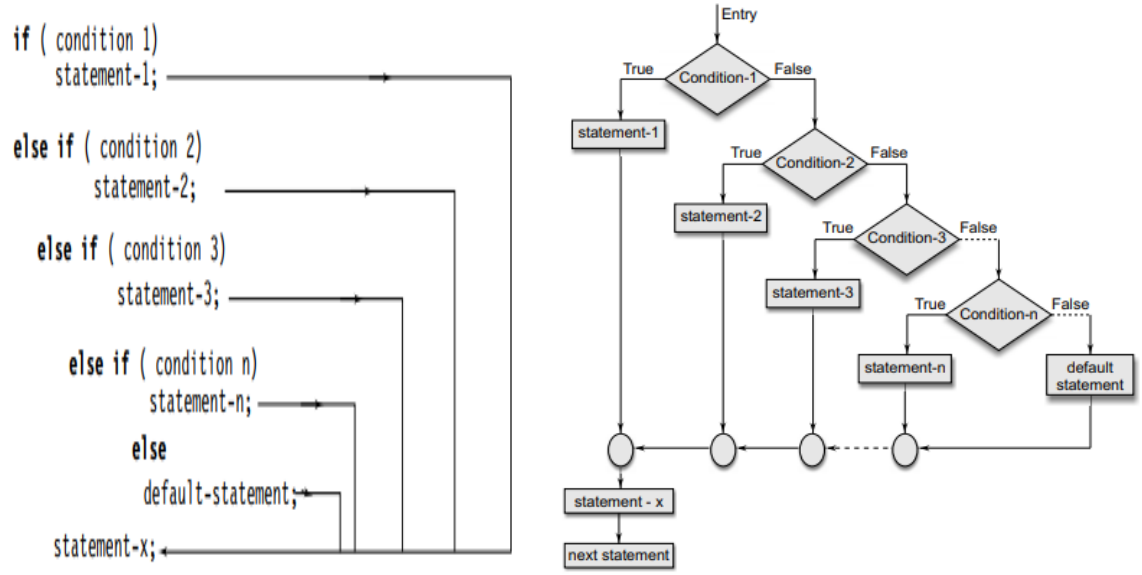


Figure 12. Flowchart of Else if Ladder

This construct is known as the else if ladder. The conditions are evaluated from the top (of the ladder), downwards. As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x (skipping the rest of the ladder). When all the n conditions become false, then the final else containing the default-statement will be executed. Fig. 12 shows the logic of execution of else if ladder statements.

Example:

```
void main()
{
int a,b,c;
a=5, b=6, c=7;
if(a>b && a>c){
printf("a is big");
}
```

```
else if(b>c){
printf(“b is big”);
}
else{
printf(“c is big”);
}
getch();
}
```

Example 2:

```
//C Program to print grade of a student using If Else Ladder Statement
#include<stdio.h>
#include<conio.h>
 int main()
{
    int marks;
   printf("Enter your marks between 0-100\n");
    scanf("%d", &marks);
    if(marks >= 90){
        /* Marks between 90-100 */
        printf("YOUR GRADE : A\n");
    } else if (marks >= 70 && marks < 90){
        /* Marks between 70-89 */
        printf("YOUR GRADE : B\n");
    } else if (marks >= 50 && marks < 70){
        /* Marks between 50-69 */
        printf("YOUR GRADE : C\n");
```

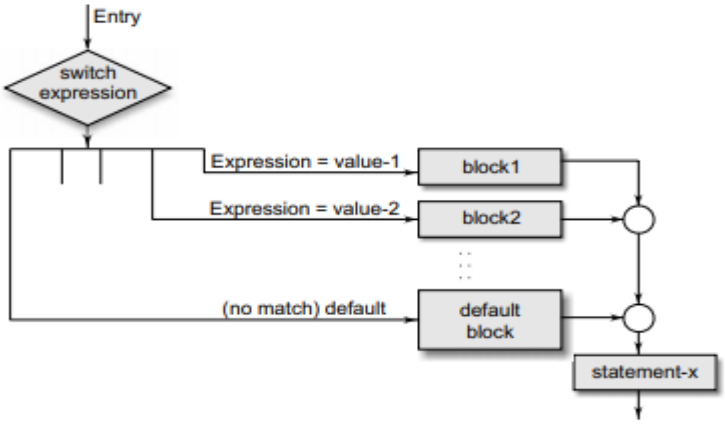
```
    } else {
       /* Marks less than 50 */
       printf("YOUR GRADE : Failed\n");
    }

    getch();
    return(0);
}
```

5.9.1.5 Switch Case Statements

We have seen that when one of the many alternatives is to be selected, we can use an if statement to control the selection. However, the complexity of such a program increases dramatically when the number of alternatives increases. The program becomes difficult to read and follow. At times, it may confuse even the person who designed it. Fortunately, C has a built-in multiway decision statement known as a **switch.** The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed. The general form of the switch statement is as shown below:

Syntax:

```
switch(expression)
{
  case value-1:
        statement;
        break;
  case value-2:
        statement;
        break;
   default;
        statement;
        break;
```



}

Statement-x;

The expression is an integer expression or characters. Value-1, value-2 ..... are constants or constant expressions and are known as **case labels.** When the switch is executed, the value of the expression is successfully compared against the values value-1, value-2,.... If a case is found whose value matches with the value of the expression, then the block of statements that follows the case are executed. The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring the control to the statement-x following the switch. The default is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values. If not present, no action takes place if all matches fail and the control goes to the statement-x

Example 1:

```
#include <stdio.h>
int main()
{
    int num=2;
    switch(num+2)
    {
       case 1:
         printf("Case1: Value is: %d", num);
       case 2:
         printf("Case1: Value is: %d", num);
       case 3:
         printf("Case1: Value is: %d", num);
       default:
         printf("Default: Value is: %d", num);
   }
   return 0;
     }
```

Output:

Default: value is 2

Example 2:

```
#include<stdio.h>
void main()
{
char operator;
int first, second;
   printf("Enter an operator (+, -, *,): ");
   scanf("%c", &operator);
   printf("Enter two operands: ");
   scanf("%d %d", &first, &second);
  switch (operator) {
   case '+':
      printf("%d + %d = %d", first, second, first + second);
      break;
   case '-':
      printf("%d + %d = %d ", first, second, first - second);
      break;
   case '*':
      printf("%d + %d = %d ", first, second, first * second);
      break;
 case '/':
      printf("%d + %d = %d ", first, second, first / second);
      break;
      // operator doesn't match any case constant
```

```
    default:
        printf("Error! operator is not correct");
    }
    return 0;
}
getch();
}
```

5.9.2 Looping Statements

'A loop' is a part of code of a program which is executed repeatedly. A loop is used using condition. The repetition is done until condition becomes true.

Depending on the position of the control statement in the loop, (that is the condition located/placed at the beginning or end of the loop) a control structure may be classified either

1. the entry-controlled loop or pre-test loop
2. the exit-controlled loop or post-test loop

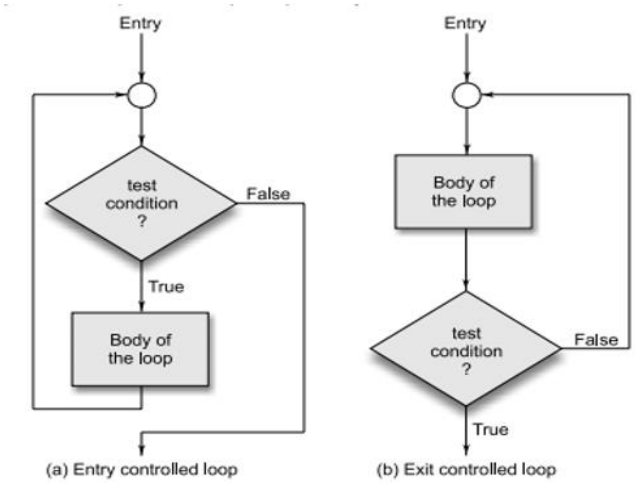
- Entry controlled loop

  While Loop

  For Loop

- Exit controlled loop

  Do While Loop



5.9.2.1 Entry Control Loop

In the entry-controlled loop,

- a condition is checked before executing the body of a loop
- If the conditions are not satisfied, then the body of the loop will not be executed.

In the case of an exit-controlled loop,

- the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time.

A looping process, in general, would include the following four steps:

1. Setting and initialization of a condition variable.
2. Execution of the statements in the loop.
3. Test for a specified value of the condition variable for execution of the loop.
4. Incrementing or updating the condition variable.

While Loop :

This is an entry controlled looping statement. It is used to repeat a block of statements until condition becomes true.

Syntax:

```
while (condition)
{
Statements;
Increment/decrement;
}
```

Example:

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

Example 2:

//Program to calculate simple interest for 3 sets of p,n and r

```
#include<stdio.h>
int main()
{
int p,n,count;
float r,si;
count=1;
while(count<=3)
{
printf("\nEnter values of p,n and r");
scanf("%d %d %d",&p,&n,&r);
si=p*n*r/100;
printf("SI=Rs.%f\n",si);
count=count+1;
getch();
}
```

For Loop :

This is an entry controlled looping statement. In this loop structure, more than one variable can be initialized. One of the most important feature of this loop is that the three actions can be taken at a time like variable initialization, condition checking and increment/decrement. The execution procedure is illustrated in fig13.

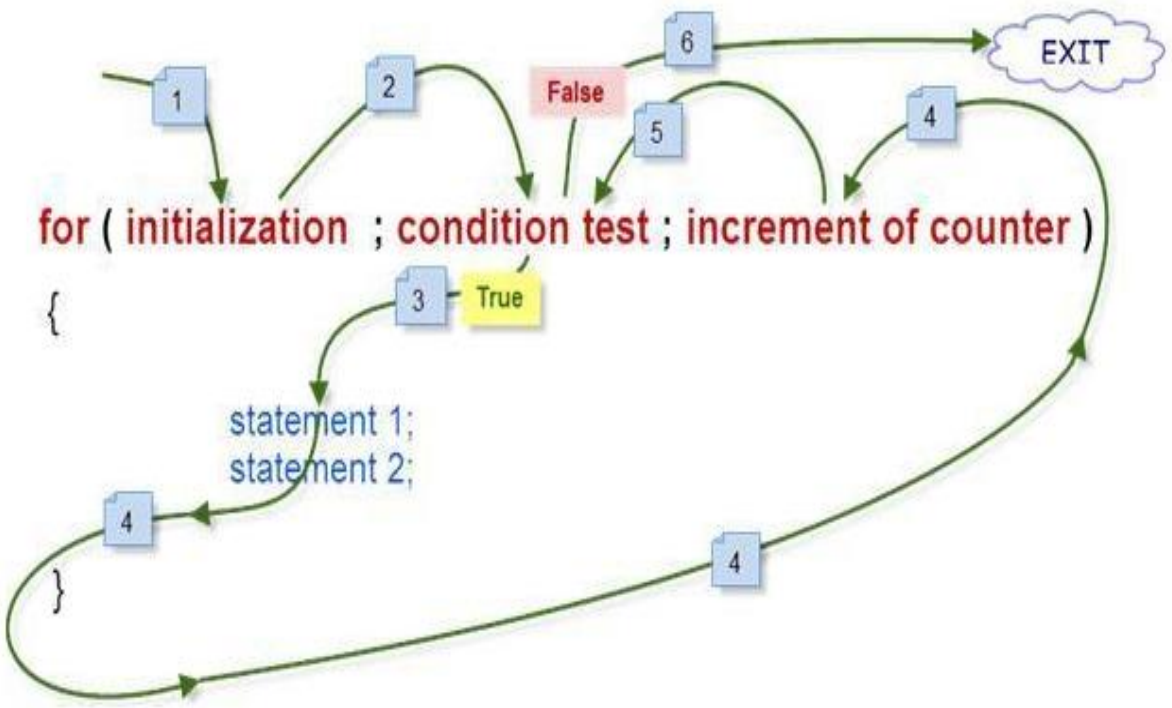


Figure 13.Flow of For loop excution

Syntax:

for(initialization; test-condition; increment/decrement)

{

Statements;

}

Example

//Program to print the number from 1-5

```
#include<stdio.h>

void main() {

int i;

clrscr();

for(i=1;i<=5; i++) {

printf("%d\t",i);
```

Initialization i = 1	Condition (i < 5)	Output Print i	Increment i++ i=i+1
i=1	1<5 (T)	1	i=1+1=2
i=2	2<5(T)	2	i=2+1=3
i=3	3<5(T)	3	i=3+1=4
i=4 i=5	4<5(T) 5<5(F)	4	i=4+1=5

```
}
getch();
}
```

5.9.2.2 Exit Control

Do While Loop:

This is an exit controlled looping statement. Sometimes, there is need to execute a block of statements first then to check condition. At that time such type of a loop is used. In this, block of statements are executed first and then condition is checked.

Syntax:

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

Example:

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

5.9.3 JUMPING STATEMENTS

There are three types of jumping statements:

- Break
- Continue
- Goto

It is necessary to exit immediately from a loop as soon as the condition is satisfied. When break statement is used inside a loop, then it can cause to terminate from a loop. The statements after break statement are skipped.

Syntax:

```
if(condition)
{
break;
}
```

```
#include<stdio.h>
void main() {
int i;
for(i=1;i<=10; i++){
If(i==5){
break;
}
else{
printf("%d\t",i);
}
}
getch();
}
```

Continue:

It is required to skip a part of a body of loop under specific conditions. The working structure of 'continue' is similar as that of that break statement but difference is that it cannot terminate the loop.

It causes the loop to be continued with next iteration after skipping statements in between. Continue statement simply skips statements and continues next iteration.

Syntax:

```
if(condition)
{
 continue;
}
```

Example

```
#include<stdio.h>
void main() {
int i;
for(i=1;i<=10; i++){
If(i==5){
continue;
}
else{
printf("%d\t",i);
}
}
getch();
}
```

GOTO STATEMENT

- The goto statement is a jump statement which jumps from one point to another point within a function or program. The goto statement is marked by label statement. Label statement can be used anywhere in the function above or below the goto statement. Simple break statement cannot work here properly. In this situation, goto statement is used.

Syntax:

```
if(condition)
{
 goto step;
}
step: statements;
```

Example

```
#include<stdio.h>
void main() {
Int i;
for(i=1;i<=10;i++){
If(
i==5){
goto step:
}
else{
printf("%d\t",i);
}
}
step:
printf("Error");
}
```

5.10 Functions

A function is a group of statement that is used to perform a specified task which repeatedly occurs in the main program. By using function, we can divide the complex problem into a manageable problem. A function can help to avoid redundancy.

Function can be of **two types**, there are

1. Built-in Function (or) Predefined Function (or) Library Function

2. User defined Function

   Difference between Predefined and User-defined Functions

Predefined Function	User-defined function
Predefined function is a function which is already defined in the header file (Example: math.h, string.h, etc)	User- Defined function is a function which is created by the user as per requirement of its owner
Predefined Function is a part of a header file, which are called at runtime	User- Defined function are part of the program which are compiled at runtime
The Predefined function name is given by the developer	User- Defined function name created by the user
Predefined Function name cannot be changed	User defined Function name can be changed

Elements of User-Defined Function

1. Function Declaration

2. Function Call

3. Function Definition

Function Declaration

- ✓ Like normal variable in a program, the function can also be declared before they defined and invoked
- ✓ Function declaration must end with semicolon (;)
- ✓ A function declaration must declare after the header file
- ✓ The list of parameters must be separated by comma.
- ✓ The name of the parameter is optional, but the data type is a must.
- ✓ If the function does not return any value, then the return type void is must.
- ✓ If there are no parameters, simply place void in braces.
- ✓ The data type of actual and formal parameter must match.

Syntax:

Return_type function_name (datatype parameter1, datatype parameter2,…);

Description:

Return type : type of function

Function_name : name of the function

Parameter list or argument list : list of parameters/variables that the function can convey.

Example:

int add(int x,int y,int z);

Function Call

The function can be called by simply specifying the name of the function, return value and parameters if presence.

Syntax:

function_name();

function_name(parameter);

return_value =function_name (parameter);

Description:

function_name : Name of the function

Parameter : Actual value passed to the calling function

Example

fun();

fun(a,b);

fun(10,20);

c=fun(a,b);

e=fun(2.3,40);

Function Definition

It is the process of specifying and establishing the user defined function by specifying

all of its element and characteristics.

Syntax:

Return_type function_name (datatype parameter1, datatype parameter2)

{}

Function Parameter

The Parameter provides the data communication between the calling function and called function. There are two types of parameters.

Actual parameter: passing the parameters from the calling function to the called function i.e the parameter, return in function is called actual parameter

Formal parameter: the parameter which is defined in the called function i.e. The parameter, return in the function definition is called formal parameter

Example:

```
main()
{
…….. Where …….. a,b are the actual Fun(a,b);
……..
parameters
……..
} x,y are formal parameter
Fun(int x,int y)
{
……
……
}
```

Function Prototype (or) Function Interface

The functions are classified into four types depends on whether the arguments are present or not, whether a value is returned or not. These are called function prototype. In 'C' while defining user defined function, it is must to declare its prototype. A prototype states the compiler to check the return type and arguments type of the function.

A function prototype declaration consists of the function's return type, name and argument. It always ends with semicolon. The following are the function prototypes

- ✓ Function with no argument and no return value.
- ✓ Function with argument and no return value.
- ✓ Function with argument and with return value.
- ✓ Function with no argument with return value.

Function with no argument and no return value

 In this prototype, no data transfer takes place between the calling function and the called function. i.e., the called program does not receive any data from the calling program and does not send back any value to the calling program.

Syntax:-

main() void Fun()

{ {

……..

……..

……..

……..

Fun();

…….. }

……..

}

Example program 1

//Program to add two numbers using function

#include<stdio.h>

void sum();

void main()

{

```
    printf("\nGoing to calculate the sum of two numbers:");
    sum();
}
void sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    printf("The sum is %d",a+b);
}
```

Example program 2

```
//Program for finding the area of a circle using Function with no argument
and no return value
#include<stdio.h>
#include<conio.h>
void circle();
void main()
{
circle();
}
void circle()
{
int r;
float cir;
printf("Enter radius");
scanf("%d",&r);
```

```
cir=3.14*r*r;
printf("The area of circle is %f",cir);
}
```

Output:

```
Enter radius 5
The area of circle 78.500000
```

Function with argument and no return value

In this prototype, data is transferred from the calling function to called function. i.e., the called function receives some data from the calling function and does not send back any values to calling function. It is one way data communication.

Syntax:-

```
main() void Fun(x,y)
{ {
…….. ……..
…….. ……..
Fun(a,b);
…….. }
……..
}
```

Example program 1:

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

```
int a,b;
printf(“Enter two values”);
scanf(“%d%d”,&a,&b);
add(a,b);
getch();
}
void add(int x,int y)
{
int c;
c=x+y;
printf(“add=%d”,c);
}
```

a and b are the actual parameters

x and y are formal parameters

Output:

Enter two values 6 4

add=10

Example program 2:

```
//Program to find the area of a circle using Function with argument and no return value
#include<stdio.h>
#include<conio.h>
void circle(int);
void main()
{
int r;
clrscr();
```

```
printf("Enter radius");
scanf("%d",&r);
circle(r);
}
void circle(int r)
{
float cir;
cir=3.14*r*r;
printf("The area of circle is %f",cir);
getch();
}
```

Output:

Enter radius 5

The area of circle 78.500000

Function with argument and with return value.

- In this prototype, the data is transferred between the calling function and called function. i.e., the called function receives some data from the calling function and sends back returned value to the calling function. It is two way data communication

Syntax:-

```
int Fun(x,y);
main()
{
…….. ……..
……..
Fun(x,y);
```

……..

c=Fun(a,b);

return(z);

…….. }

}

Example program 1:

//Program to add two numbers using function with argument and with return value

```
#include<stdio.h>
#include<conio.h>
void add(int,int);
void main()
{
clrscr();
int a,b,c;
printf("Enter two values");
scanf("%d%d",&a,&b);
c=add(a,b);
printf("Add=%d",c);
getch();
}
void add(int x,int y)
{
int m;
m=x+y;
return m;
}
```

Example Program 2

// Program to find the area of a circle using Function with argument and with return value

```
#include<stdio.h>
#include<conio.h>
float circle(int);
void main()
{
int r,area;
clrscr();
printf("Enter radius");
scanf("%d",&r);
area=circle(r);
printf("the area of circle is %f",area);
getch();
}
float circle(int r)
{
float cir;
cir=3.14*r*r;
return cir;
}
```

Output:

Enter radius 5

the area of circle 78.500000

Function with no argument with return value

 In this prototype, the calling function cannot pass any arguments to the called function, but the

called program may send some return value to the calling function. It is one way data communication

Syntax:-

```
main() int Fun()
{ {
……..
……..
……..
……..
Fun();
return(z);
…….. }
……..
}
```

Example program 1

```
#include<stdio.h>
#include<conio.h>
int add();
void main()
{
clrscr();
int z;
z=add();
printf("Add=%d",z);
getch();
}
```

```
int add()
{
int a,b,c;
printf(“Enter two values”);
scanf(“%d%d”,&a,&b);
c=a+b;
return c;
}
```

Output:

Enter two values 6 4

Add=10

Example Program 2

// Program to the area of a circle using no argument with a return value

```
#include<stdio.h>
#include<conio.h>
float circle();
void main()
{
int area;
clrscr();
area =circle(r);
printf("the area of circle is %f",circle());
getch();
}
float circle()
{
```

```
float cir;
int r;
printf("Enter radious");
scanf("%d",&r);
cir=3.14*r*r;
return cir;
}
```

Output:

```
Enter radius 5
the area of circle 78.500000
```