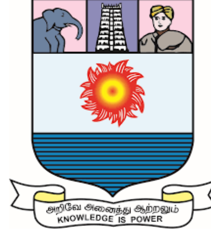


மனோன்மணியம் சுந்தரனார் பல்கலைக்கழகம்

MANONMANIAM SUNDARANAR UNIVERSITY
TIRUNELVELI- 627 012

தொலைநிலை தொடர் கல்வி இயக்ககம்

DIRECTORATE OF DISTANCE AND CONTINUING EDUCATION

B.Sc. Chemistry

GENERIC ELECTIVE - III

Programming Language C

Course Code: JECS31

Prepared By

Dr. S. Immaculate Shyla

Assistant Professor & Head,

Department of Artificial Intelligence and Data Science,

Holy Cross College (Autonomous),

Nagercoil-629 002.

B. Sc Chemistry

Programming Language C

Unit- I

Introduction – Character set – C Tokens –Keywords and Identifiers – Constants – Variables –Data types. (Chapter 2: Sections - 2.1 to 2.7)

Unit- II

Operators: Arithmetic – Relational –Logical – Assignment– Increment and Decrement – Conditional – Bitwise – Special – Precedence of Arithmetic operators – Managing input and output operation: Reading and writing a character – Formatted input and output. (Chapters 3 and 4: Sections - 3.1 to 3.9, 3.12, 4.2 to 4.5)

Unit- III

Decision making and branching: Statements: IF, IF … ELSE, Nesting of IF … ELSE, ELSE IF Ladder and Switch statements – The ?:operator – The GOTO statement – Decision making and looping: The WHILE, DO and FOR statements –Jumps in loops. (Chapters 5 & 6: Sections - 5.3 to 5.9, 6.2 to 6.5)

Unit- IV

Array: One dimensional and two-dimensional arrays– Declaration, Initialization of arrays – Multidimensional arrays Character arrays and strings: Declaring and initializing string variables – Reading and writing of strings – String handling functions.(Chapters 7 & 8: Sections 7.1 to 7.7, 8.1 to 8.8)

Unit- V

User defined functions: Definition of function –Return values and their types – Function calls – Function declaration – Category of functions – Nesting of functions – Recursion. (Chapter 9:Sections 9.5 to 9.9, 9.15, 9.16)

Recommended Text

E. Balaguruswamy- Programming in ANSIC–Tata McGraw Hill Publishing company limited III Edition, 2017.

References:

1. Yashavant Kanetkar, 2016. *Let Us C*, 15th Edition, BPB Publications.
2. Herbert Schildt, 2017. *The Complete Reference C*, 4th Edition, McGraw Hill Education.

UNIT-I

Introduction to C Programming

- C is a general purpose, procedural, case sensitive high level programming language.
- It is developed at AT & T's Bell Laboratories of USA in 1972.
- It is developed by Dennis Ritchie.
- It is an upgraded version of two earlier languages, called BCPL and B.
- It can be used to write the program for all possible application.
- All statements in 'C' program should be written in lower case letters only. Uppercase letters are only used for symbolic constants and variables.

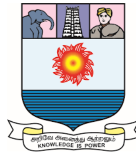
Features / Advantages of C Language:

- C is a general purpose, structured programming language.
- C programs are fast and efficient.
- C is powerful and flexible.
- C is highly portable.
- C is well suited for writing system software as well as application software.

A Simple C Program:

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

Every C program must contain a function called main. All C programs start its execution from the main function. Every program may have zero or more user defined function.



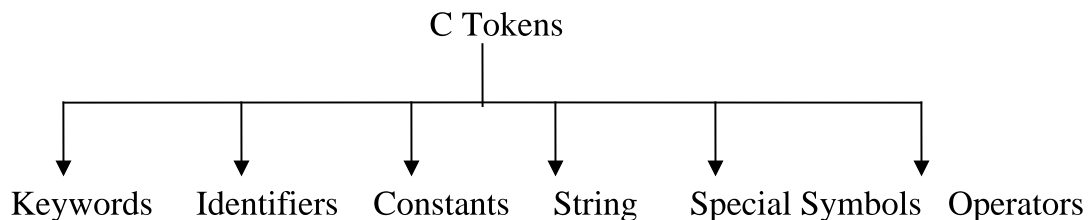
Character Set

Character set are the set of alphabets, letters and some special characters that are valid in C language. The characters in C are grouped into the following categories.

- ✓ **Alphabets**:
  - Uppercase: A B C .... X Y Z
  - Lowercase: a b c .... x y z
- ✓ **Digits**:
  - 0 1 2 3 4 5 6 7 8 9
- ✓ **Special Characters**
  - < > . _ ( ) ; $: % [ ] # ? ‘ & {} “ ^ ! * / | - \ ~
- ✓ **White Spaces**
  - blank space, new line, horizontal tab, carriage return and form feed.

C TOKENS

The smallest individual units of a C program are known as tokens. The following figure shows the different types of C tokens.



KEYWORDS

Keywords are reserved words whose meanings are fixed by the Compiler. They must be written in lower case letter. There are 32 keywords available in C. The following table shows the keywords in C.

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

IDENTIFIER

Identifiers are name of an object such as variables, functions and arrays. These are defined by the user.

Rules for an identifier:

1. It must begin with an alphabet.
2. Remaining characters must be an alphabet, number & underscore symbol.
3. No special characters allowed.
4. Both small & capital letters are permitted.
5. Small & capital letters are treated differently.
6. Maximum length of an identifier is 8.
7. Keywords cannot be used as an identifier.

CONSTANTS

The values that cannot be changed during the execution of a program are called constants.

Types of C constants:

C constants can be divided into three major categories

1) Numeric Constant
    a) Integer Constant
    b) Floating Point Constant
2) Character Constant
3) String Constant

1) Numeric Constant

a) Integer Constant

An integer constant formed with the sequence of digits. There are three types of integer constants.

Decimal constant: It is formed with decimal numbers.

Octal constant: It is formed with octal numbers.

Hexadecimal constant: It is formed with hexadecimal numbers.

Rules for Integer Constant:

- ✓ An integer constant must have at least one digit
- ✓ It must not have a decimal point
- ✓ It is either positive or negative
- ✓ Commas or blank spaces are not allowed

Example:

Decimal Constant

42

-782

Octal Constant (Starts with a leading 0 and remaining may be in between 0 to 7)

056

03

Hexadecimal Constants (Starts with a leading 0x and remaining may be in between 0 to 9 or A to F)

0x7D

0X5B3

b) Floating Point Constant

A floating point constant is made up of a sequence of numeric digits with a presence of a decimal point.

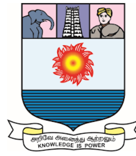
Example:

distance = 126.0;

height = 5.6;

Rules for floating point Constants:

- A floating point constant must have at least one digit.
- It must have a decimal point
- It is either positive or negative
- Default sign is positive.



- Commas or blank spaces are not allowed

2) Character Constant

The character constant contains a single character enclosed within a pair of single quote symbol.

Example: 's' , 'M' , '3' , '-'

3) String Constant

A string constant is a sequence of characters enclosed within double quote. The characters may be letters, number, special characters and blank spaces, etc. At the end of string '\0' is automatically placed.

Example: "Hi"

"Give Number of data" , "39.77" , "50"

Backslash Character Constants

C supports special backslash character constants that are used in output functions. These character combinations are known as escape sequences.

Constants	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
'\0'	Null character

VARIABLE

A variable is an identifier which is used to hold data in a program. That data value may change during the execution of a program.

Example: name, Reg_No, n

Variable Declaration:

After choosing suitable variable names, we must declare them in the program.

Syntax	:	datatype v_1, v_2……v_n;
Description	:	datatype → is the type of data
		v_1, v_2……v_n→ list of variables
Example	:	int code;
		char gender;
		float price;
		char name [10];

Initializing variables:

- ✓ Initialization of variables can be done using the assignment operator (=).
- ✓ The variables can be initialized while we declare it.

Syntax	:	variable_name = constant;
		Or
		Datatype variable_name = constant;
Example	:	int total;
		total = 0;
		(or)
		char Gender = 'M';

In the above example, we initialize the value 0 to variable total and initialize the value 'M' to variable Gender.

Scope of Variables:

Variables have two types of scopes Local & Global.

i) Local Variables:

The variables which are defined inside a function are called local variables.

```
Example   :  void sample ( )
             {
             int Reg_No;
             /* body of function */
             }
```

These variables are visible only within the function only.

In the above example the variable Reg_No is defined inside the function sample. So it can be used within function sample only. ie., Reg_No is a local variable of function sample.

ii) Global / External Variables:

The variables that are declared outside any function are called the global / external variables. These variables are accessed from any function within the program.

```
Example   :  int A = 2; // global variable
             main ( )
             {

             }
```

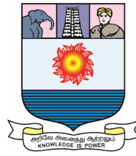
The integer variable A is a global variable, since it is declared outside the function main( ).

DATA TYPES

The data type defines 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 into

1. Basic Data type (Primitive Data Type)
2. User defined Data type
3. Derived Data type



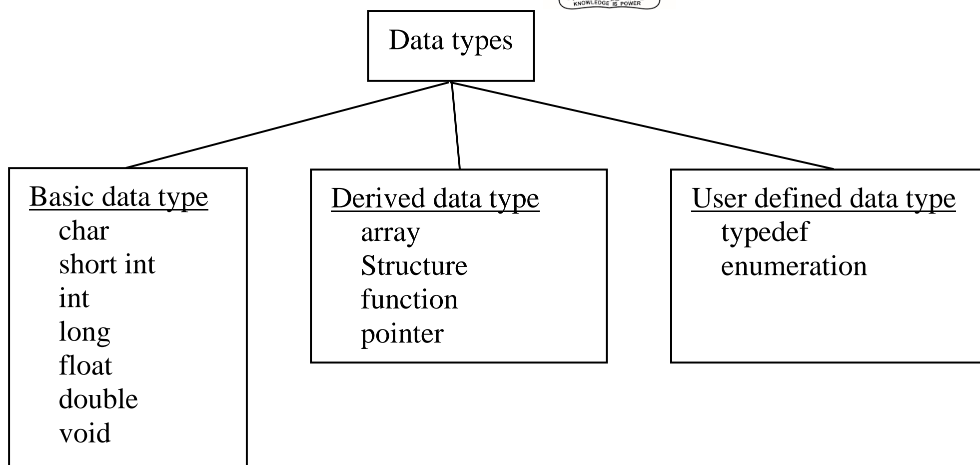


Fig: Classification of Data Types

Basic Data Types:

a) Integer Data Type:

Integer type has the following sub categories. They are short int, int and long. Each of these may be signed or unsigned.

Data Type	Size in Bytes	Range of valid values
short int	1	-128 to 127
int	2	-32768 to 32767
Long	4	-2^{31} to 2^{31}

Example:

```
short int k;
int mark;
long qty;
```

b) Character Data Type:

To store a single character declare the variables in char data type. Its size is 1 byte.

Example:

```
char choice;
choice = 'Y';
```

c) Floating Point type:

The ‘float’ data type represents single precision floating point number. Its size is 4 bytes. It uses 6 digits of precision.

The ‘double’ data type represents double precision floating point number. Its size is 8 bytes. It uses 14 digits of precision.

Example :

```
float avg;
double  k;
```

User defined Data types:

(a) typedef:

It allows the users to define an alternate (or) alias name for an existing data type, and this can be used to declare variables.

```
typedef int Marks;
Marks M1, M2;
```

Here Marks is another name for the 'int' data type. Therefore M1 and M2 are integer variables.

(b) Enumerated data type:

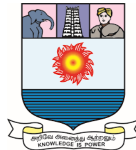
The C language provides another user defined data type called enumerated data type.

Syntax	:	enum identifier{value1, value 2….value };
Example	:	enum Day { Mon, Tue, Wed, Thu, Fri, Sat, Sun};
		enum Day D1, D2;
		D1 = Wed;
		D2 = Sun;

The identifier follows with the keyword enum is used to declare the variable that can have only one value from the enumeration constants.

Derived Data Types

a) Array



An array is a collection of similar data items that are stored under a common name.

Example : int Marks [5] ;

b) Function

A function is a self contained block of program statements that performs a particular task.

c) Pointers

The pointer variable holds the memory address of another variable. It provides a way of accessing a variable.

Example:

```
int x;
int *ptr = &x;
```

d) Structure

A structure is a collection of related data elements of different data type under a single name. In other programming languages it is called as record.

Example:

```
struct student
{
int reg_no;
char name[20];
};
```

e) Union

It is similar to structure. But the main difference between union and structure is in terms of storage. In structure each member has its own storage location, whereas in union all the members use the same location.

Example:

```
union student
{
int reg_no;
char name[20];
};
```

UNIT-II

OPERATORS

- ✓ An operator is a symbol that is used to write a mathematical, logical or relational expression.
- ✓ An expression is a sequence of operators and operands that specifies the computation.
- ✓ An operand can be a variable, constant or a function call.

Syntax

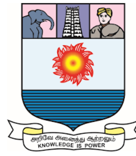
variable = expression

Example: Sum = 2 + 3

The above expression involves three operands namely Sum, 2 and 3. It have two operators = and +.

Types of 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



Arithmetic Operators:

Arithmetic operations like addition, subtraction, multiplication, division etc can be performed by using arithmetic operators.

Operator	Name	Example
+	Addition	12 + 4
-	Subtraction	a – b
*	Multiplication	2 * 9
/	Division	a / 3
%	Remainder (Modulo Division)	13 % 3

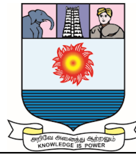
Sample Expressions

```
sum = b + c;   sub = b - c;
 mul = b * c;   div = b / c;
 rem = b % d;
 exp = b / c * d;
```

Relational Operators:

- ✓ Relational operators are used to compare two or more operands.
- ✓ In if, for and while statements we use relational expression.
- ✓ Operands may be variables, constants or expression.
- ✓ Relational expressions return either True or False.
- ✓ For example, we may compare the age of two persons, or the price of two items etc.

Syntax: expression relational_ operator expression



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

Example:

- ✓ if (A != B)
- ✓ while(i < n)

Logical Operators:

Logical operators are used to combine the results of two or more relational expressions (conditions).

Operator	Meaning	Example
!	Logical NOT	!(A < B)
&&	Logical AND	(A < B) && (A < C)
||	Logical OR	(A < B) || (A < C)

- ✓ Logical NOT is a unary operator that negates the logical value of its single operand.
- ✓ Logical NOT convert a non zero to 0, and 0 to 1.
- ✓ Logical AND produces 1 if both operands are 1, otherwise produce 0.
- ✓ Logical OR produces 0 if both operands are 0, otherwise it produces 1.

Assignment Operator:

Assignment operator '=' is used to assign a constant or a value of an expression or a value of a variable to other variable.

Syntax : Variable = expression (or) value

Example: x = 10; //Assign a constant

c = a + b; // Assign a value of an expression

x = y; // Assign a value of a variable

i) Short hand Assignment Operator

C provides compound assignment operators to assign a value to a variable in order to assign a new value to a variable after performing a specified operation.

Operator	Example	Meaning
+=	x += y	x = x + y
-=	x - = y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

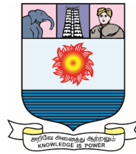
ii) Nested (or) Multiple Assignments

Using this feature, we can assign a single value or an expression to multiple variables.

Syntax : $var_1 = var_2 \ldots var_n$ = Value or expression;

Example : i = j = 1;

x = y = a * b + c;



Increment and Decrement Operators (unary):

- C has increment (++) and decrement (--) operators.
- The '++' adds one to the variables and '--' subtract one from the variable.
- These operators are called unary operators, since they act upon only one variable.
- If we use pre increment or pre decrement in an expression, the variable value is increased or decreased by one first then it takes the value of the variable for calculation.

Operator	Meaning
++ x	Pre increment
-- x	Pre decrement
x ++	Post increment
x --	Post decrement

- If we use post increment or post decrement in an expression, it takes the current value for calculation then only it increases or decreases the variable value by one.

Conditional Operator (or) Ternary Operator:

It is equivalent to simple **if then else** statement. It checks the condition and executes the exp1 if condition is true otherwise it executes exp2.

Syntax : condition ? exp1 : exp 2;

Example Program:

```
main ( )
{
```

```
int a = 5, b = 3, max;
 max = a > b ? a : b ;
  printf("Maximum is %d", max);
}
```

Output: Maximum is 5

In this example, it checks the condition 'a > b', if it is true, then the value of 'a' is assigned to 'max', otherwise the value of 'b' is assigned to 'max'.

Bitwise Operators:

- ✓ Bitwise operators are used to calculate the data at bit level.
- ✓ It operates on integers only.
- ✓ It can't be applied to floating point data.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	Bitwise NOT (or) One's complement

Special Operators:

C language supports some of the special operators.

Operators	Meaning
,	Comma operator
size of	Size of operator
& and *	Pointer operators
. and →	Member selection operators

Comma Operator

Used to separate elements.

Example :

int X, Y;

sizeof Operator

Used to return the size of the data type or variable.

Example:

```
#include<stdio.h>
void main()
{
        int x;
        printf("Size of variable x = %d" , sizeof(x);
        printf("Size of float data type is  = %d" , sizeof(float);
}
```

Output:

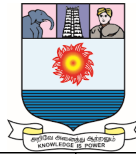
Size of variable x = 2

Size of float data type is = 4

Operator Precedence & Associativity:

- Usually, the Arithmetic operators are evaluated from the left to right using the precedence of operators when the expression is written without the parameters.

Operators	**Associativity**
( ), [ ]	Left to Right
++, - -, ! , &,	Right to Left
*, /, %	Left to Right
+, -	Left to Right



<<, >>	Left to Right
< , <=, >, >=	Left to Right
==, !=	Left to Right
&	Left to Right
^	Left to Right
|	Left to Right

Example:

Exp = x – y / 3 + z * 3 – 1

Assume x = 3, y = 9, z = 10

Then Exp = 3 – 9 / 3 + 10 * 3 - 1

This is solved in step by step as follows:

Exp = 3 – 3 + 10 * 3 -1

Exp = 3 - 3 + 30 -1

Exp = 0 + 30 - 1

Exp = 30 – 1

Exp = 29

- During evaluation of expression, the order of evaluation can be changed by putting parentheses.
- The sub expressions given within parentheses are evaluated **first**.

Example:

Exp = (x – y) / 3

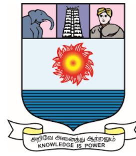
Assume x = 8, y = 2

This is solved in step by step as follows:

Then Exp = (8 – 2) / 3

= (6) / 3

= 2



Rules for Evaluation of Expression:

- The highest precedence is given to the expressions within parenthesis.
- Evaluate the inner most sub expression if the parenthesis is nested.
- Evaluate the sub expressions from left to right if parenthesized.
- Apply the associativity rule, if more operators of the same precedence occur.

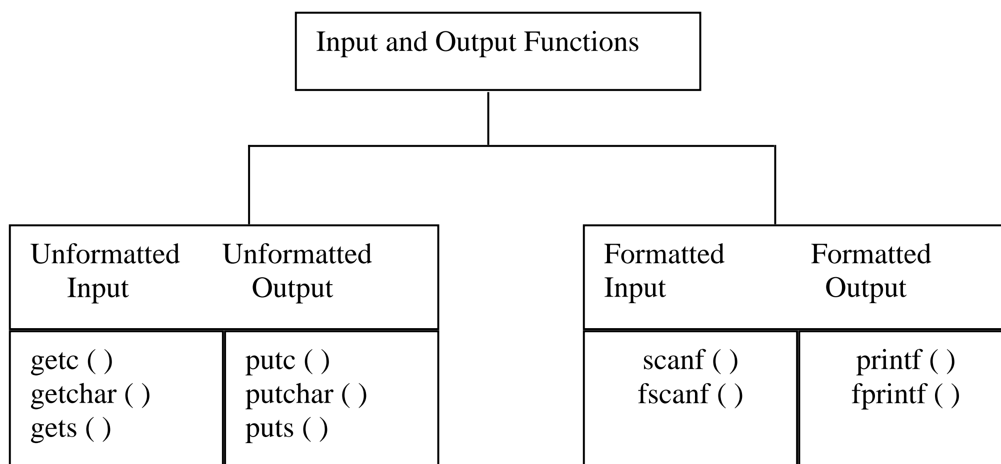
MANAGING INPUT AND OUTPUT IN C

Input, process and output are the three essential features of a computer program.

✓ The program takes some input data then process it and gives the output.

✓ In 'C' language, two types of input and output statements. They are:

- Unformatted i/o statements
- Formatted i/o statements



Reading and Writing a Character

Unformatted Input / Output Statements

In an unformatted i/o statements no need to specify the type & size of the data to be read or write.

(a) getchar()

The getchar( ) is an input function that reads a single character from the standard

input device (keyboard).

Syntax:

```
char _ variable = getchar ( );
```

Where char_variable is the name of a variable that is of char type.

Example:

```
char ch;
ch = getchar ( );
```

Example Program :

```
# include<stdio.h>
void main ( )
{
    char ch;
    printf (“Enter any one character : ”);
    ch = getchar ( );
    printf (“The character you typed is %c”, ch);
}
```

Output :

```
Enter any one character : M
The character you typed is M
```

(b) putchar()

The putchar( ) is an output function that writes a single character on the standard output device (monitor).

Syntax : putchar (char_variable);

Example Program :

```
# include<stdio.h>
void main ( )
{
    char ch;
```

```
        printf ("Enter any one character : ");
        ch = getchar ( );
        printf ("The character you typed is ");
        putchar(ch);
    }
```

Output :

Enter any one character : S

The character you typed is S

Formatted I/O Statements

The scanf ( ) & printf () functions are the formatted i/o statements. These functions are used to read & write different types of data.

(a) scanf ()

Input data can be read from standard input device (keyboard) using scanf ( ) function.

Syntax : scanf ("Control String", &var1, &var2, . . .);

Example *:* scanf ("%d %d", &a, &b);

Control String:

Control string specifies the type of data to be read and its size. The following list represents the possible control strings.

%c - To read single character

%s - To read a string

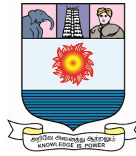
%ws - To read a string with size w.

%d - To read an integer

%wd - To read an integer with w digits.

%f - To read a floating point number

%w.pf - To read floating point data. w represents integer part size and p represents decimal part size.



Rules for scanf()

- Each variable name must be preceded by an address of variable symbol (&).
- The control string and variables data type should match each other.

(b) printf ( )

Output data can be displayed in the standard output device (monitor) using printf () function.

Syntax : printf (“Control String”, var1, var2, . . .);

Example :

printf (“%d %d”, a, b);

printf (“Factorial = %d”, fact);

Rules for printf()

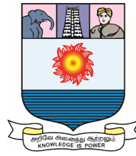
- The control string and variables data type should match each other.
- The variable must be separated by commas and need not be preceded with ‘&’ symbol.

Example Program:

```
# include<stdio.h>
void main ( )
{
        int A, B, C;
        printf (“Enter values for A and B : ”);
        scanf (“%d %d”, &A, &B);
        C = A + B;
         printf(“Sum is %d”, C)
}
```

Output:

```
Enter values of A and B :  4   3
Sum is 7
```



Unit- III

DECISION MAKING AND BRANCHING

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. Many practical situations like decision making, repetitive execution of certain task etc require alteration of the default flow of control. It is achieved by flow control statements. There are 2 types of flow control statements. They are,

- ✓ Branching Statements
- ✓ Iterative Statements

Branching Statements

- ✓ Branching statements are used to transfer the execution sequence from one point to another.
- ✓ They are categorized as:

  1. Conditional branching

  2. Unconditional branching

Conditional Branching Statements

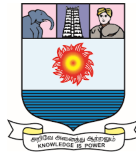
In conditional branching, program control is transferred from one point to another based upon the outcome of the condition. The conditional branching statements are:

i) if Statement
ii) if-else Statement
iii) switch Statement

Simple IF Statement

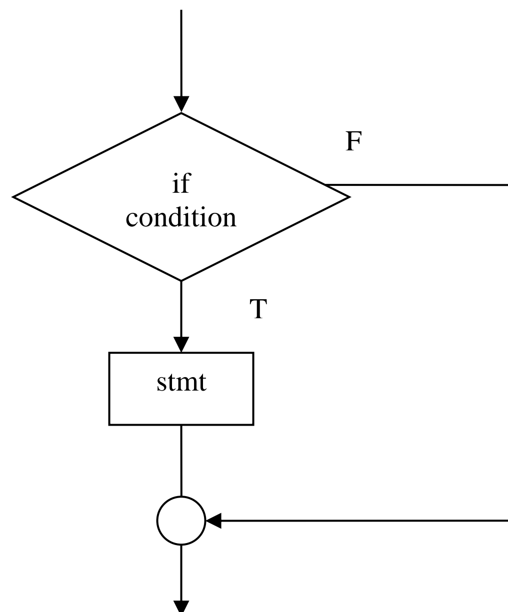
It checks the given condition in if statement and if it is true then it will execute the body of if statement (then part) otherwise it skipped the body of if statement.

Syntax:



```
if (test expression)
{
        Statement block
}
```

The statement block may be a single statement or a group of statements.



Example Program :

```
# include<stdio.h>
void main ( )
{
        int a;
        printf (“\n Enter a number : ”);
        scanf (“%d”, &
        if ( a > 0)
                printf (“The given number is positive number”);
}
```

If - Else Statement

- ✓ It is a two way branching statement.

- ✓ If the test expression is true then the True part statement block will be executed.
- ✓ If the test expression is false then the False part statement block will be executed.

Syntax:

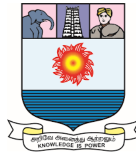
```
if ( test expression)
{
        True part Statement block
}
else
{
        False part Statement block
}
```

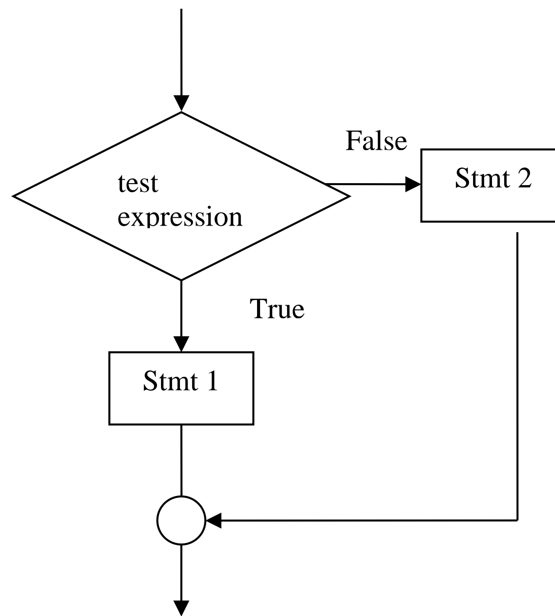
Example Program: Program to check whether the number is odd or even

```
# include<stdio.h>
void main ( )
{
        int n, r;
        printf (“\n Enter a Number:”);
        scanf (“%d”, &n);
        r = n % 2;
        if ( r == 0 )
                printf (“Given Number is Even”);
        else
                printf (“Given Number is Odd”);
}
```

Output:

```
Enter a Number:  6
Given Number is Even
```





Nesting of IF..ELSE Statement

If we give if statement within another if statement it is called nested if statement.

Syntax:

```
if ( test expression1 )
{
        if ( test expression2 )
        {
                Inner if True part Statement
        }
        else
        {
                Inner if False part Statement
        }
}
else
{
        Outer if False part Statement
}
```

It checks the test expression1 and if it is true it check the inner if test expression2. This type of nested if is useful when a series of decisions are involved.

Example Program :

```
# include<stdio.h>
void main ( )
{
        int Mark;
        printf ("Give your Mark");
        scanf ("%d", &Mark);
        if ( Mark < 50 )
                printf("Failed");
        else
        {
                if ( Mark < 60 )
                        printf("Second Class");
                  else
                        printf ("First Class");
        }
}
```

ELSE..IF Ladder

If the else part of if statement contain another if statement, then the else and the if statement can be combined. It is called else if ladder.

Syntax :

```
if ( test expression1 )
        Statement block 1
else if ( test expression2 )
        Statement block 2
else if ( test expression3 )
        Statement block 3
else
```

Statement block 4

If the test expression1 evaluated is true, the statement block1 is executed.

If the test expression2 is true, then statement block2 is executed and so on.

If none of the test expressions are true, then the statement block4 is executed, ie., the last statement.

Example Program : **To find largest among three numbers**

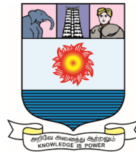
```
# include<stdio.h>
void main ( )
{
        int a, b, c;
        printf (“Enter three numbers : ”);
        scanf (“%d %d %d”, &a, &b, &c);
        if (a > b) && (a > c)
                printf(“Biggest Number is %d”, a);
        else if (b > c)
                printf(“Biggest Number is %d”, b);
        else
                printf (“Biggest Number is %d”, c);
}
```

Output :

```
Enter three numbers : 40  -50  35
Biggest Number is 40
```

Switch Statement

- ✓ It is a multi way branching statement.
- ✓ It first evaluates the expression in switch statement. That result is compared with each case value one by one.
- ✓ Whenever a match found execute the statements given in the corresponding case statement.
- ✓ If none of the case value matches with the result it executes the default section.



- ✓ In switch, selection expression must be integer or character type only.
- ✓ Only equality condition can be applied. Other relational operators cannot be used.
- ✓ It allows the more complicated decision statements in a simple manner.

Syntax:

```
        switch (Expression)
        {
case value 1:
    Statement block 1
    break;
case value 2:
    Statement block 2
    break;
    . . .
case value n:
    Statement block n
    break;
 default:
    Default Statement block
        }
```

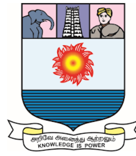
Rules for Writing 'switch' Statement

- ✓ The expression used in switch statement must be an integer or a character data.
- ✓ The case labels must be character or integer constant.
- ✓ Each case block must be terminated by break statement. Otherwise, all statements that are followed by matched cases are executed.
- ✓ The default clause is optional & usually placed at the end.
- ✓ The case keyword must terminate with colon (:)
- ✓ No two case constants are identical.

Example Program:

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

```
#include<conio.h>
int main()
{
char ch;
clrscr();
printf("\t\tVowel or Consonant\n");
printf("\t\t================\n\n");
printf("Enter any Character: ");
scanf("%c",& ch);
switch(ch)
        {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u':
                case 'A':
                case 'E':
                case 'I':
                case 'O':
                case 'U':
                printf("The given letter is a Vowel");
                break;
                default:
                printf("The given letter is a Consonant");
        }
getch();
return 0;
}
```



Conditional Operator (? :)

The conditional operator is also known as a **ternary operator**. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, such as, '?' and ':'.

As conditional operator works on three operands, so it is also known as the ternary operator.

The behavior of the conditional operator is similar to the 'if-else' statement as 'if-else' statement is also a decision-making statement.

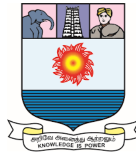
Syntax : *Expression1 ? expression2: expression3;*

Explanation:

- In the above syntax, the expression1 is a Boolean condition that can be either true or false value.
- If the expression1 results into a true value, then the expression2 will execute.
- The expression2 is said to be true only when it returns a non-zero value.
- If the expression1 returns false value then the expression3 will execute.
- The expression3 is said to be false only when it returns zero value.

Example:

```
#include <stdio.h>
int main()
{
   int age;  // variable declaration
   printf("Enter your age");
   scanf("%d",&age);   // taking user input for age variable
   (age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting"));  //
conditional operator
   return 0;
}
```



Unconditional Branching Statement Or Jumps in Loops

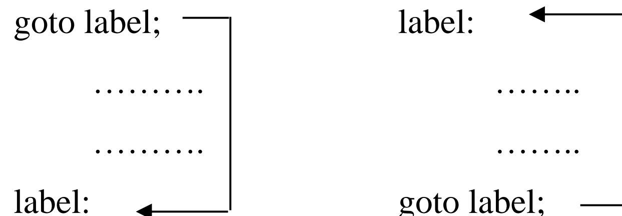
In an unconditional branching, program control is transfer from one point to another without checking the condition. Following are the unconditional branching statements.

i) goto
ii) break
iii) continue
iv) return

goto Statement

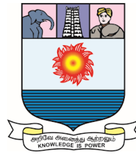
- ✓ 'C' provides the goto statement to transfer control unconditionally from one place to another place in the program.
- ✓ The goto statement can move the program control almost anywhere in the program.
- ✓ The goto statement requires a label.
- ✓ The label is a valid variable name and must end with colon (:)

Syntax:



Example Program : **Check the given number is Prime or Not using goto & return.**

```
# include<stdio.h>
# include<conio.h>
void main ( )
{
        int No, i;
        printf ("Give the number : ");
        scanf ("%d" , &No);
        for ( i = 2 ; i <= No / 2; i++ )
        {
                if ( No / i == 0 )
```



```
                    goto stop;
          }
          printf (" Given Number is a  Prime Number");
          return;
     stop :      printf (" Given Number is not a  Prime Number");
     }
```

Output:

```
Give the number :  17
Given Number is a Prime Number
```

break Statement

- ✓ It is used within a looping statement or switch statement.
- ✓ The break statement is used to terminate the loop.
- ✓ When the break statement is used inside any looping statement, control is automatically transferred to the first statement after the loop.
- ✓ In switch statement each case block must be terminated with break statement to exit from switch.

Syntax:

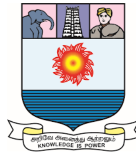
```
break;
```

Example:

Refer switch example program.

***Example Program**:*

```
#include<stdio.h>
#include<conio.h>
int main()
{
char ch;
clrscr();
printf("\t\tVowel or Consonant\n");
printf("\t\t===============\n\n");
```



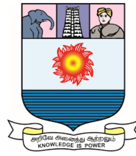
```
printf("Enter any Character: ");
scanf("%c",& ch);
switch(ch)
        {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u':
                case 'A':
                case 'E':
                case 'I':
                case 'O':
                case 'U':
                printf("The given letter is a Vowel");
                break;
                default:
                printf("The given letter is a Consonant");
        }
getch();
return 0;
}
```

continue Statement

- ✓ It is used within looping statements.
- ✓ When the continue statement is used inside the loop, it skip the statements which are available after this statement in the loop and go for the next iteration.

Syntax:

```
continue;
```



Example Program : **To display 1 to 10 except 5**

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

Output:

1 2 3 4 6 7 8 9 10

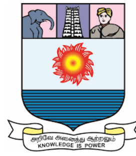
S. No	Break	Continue
1	Break statement takes the control to the outside of the loop	Continue statement takes the control to the beginning of the loop.
2	It is used both in loop and switch statements	This can be used only in loop statements

return Statement

The general form of a return statement is

```
return;
   OR
return expression;
   OR
return(expression);
```

- A return statement without an expression can appear only in a function whose return type is void.
- A return statement with an expression should not appear in a function whose return type is void.
- A return statement terminates the execution of a function and returns the control to the calling function.



Decision Making and Looping

- ✓ The loop is defined as the block of statements which are repeatedly executed for a specified number of times or until a particular condition is satisfied.
- ✓ If there is a single statement in the loop, the blocking braces is not necessary. If more than one statement in the loop then the loop statements must be placed within braces.

The following are the loop structures in 'C'.

1. for
2. while
3. do – while

for Loop

If we know exactly how many times the loop statements are repeated then the best choice is for loop.

Syntax:

```
for ( initialization; test expression; increment / decrement)
  {
        Statements
    }
```

Syntax Explanation:

Initialization:

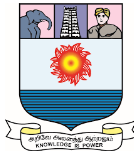
It has the initial value for the counter variable. It may be skipped. In a for loop initialization is executed first. It is executed only once i.e., for the first iteration only.

Condition:

The condition represents a test expression. Here we test the counter variable value and if the condition is true then repeat the loop otherwise exit from loop.

Incrementing / updating:

After completing every iteration, the counter variable must be increased or decreased. Otherwise, it may lead to an infinite loop.



Example Program: **To print the sum of the series 1 + 2 + 3 + 4 . . . up to N terms**

```
#include<stdio.h>
void main ( )
{
        int i, sum = 0, n;
        printf (“Enter the number of terms”);
        scanf (“%d”, &n);
        for (i = 1; i <= n; i++ )
                sum += i;
        printf (“\n Sum =  %d”, sum);
}
```

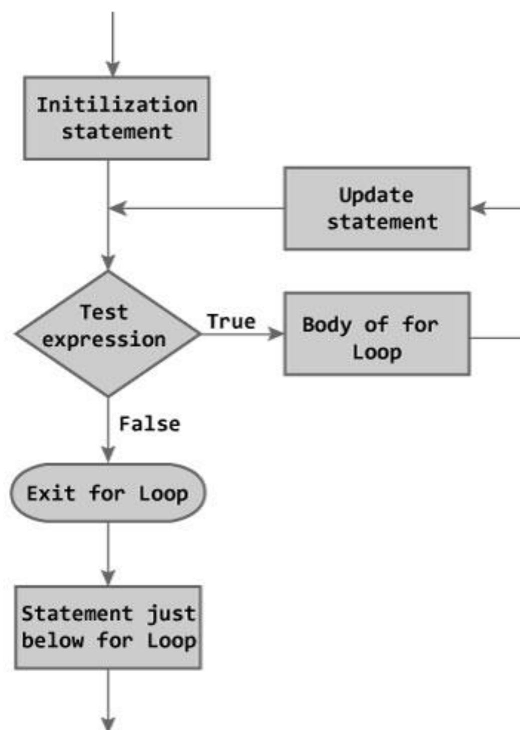
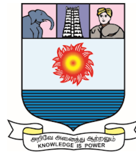


Figure: Flowchart of for Loop

while Loop

- ✓ It is a pre testing loop.
- ✓ The conditional expression is tested before the body is executed.
- ✓ If the condition is true the loop will be repeated otherwise stop the iteration.



- ✓ If the very first time itself the condition failed, the loop will not be executed at least one time.

Syntax:

```
while (test expression)
{
        Body of the loop
}
```

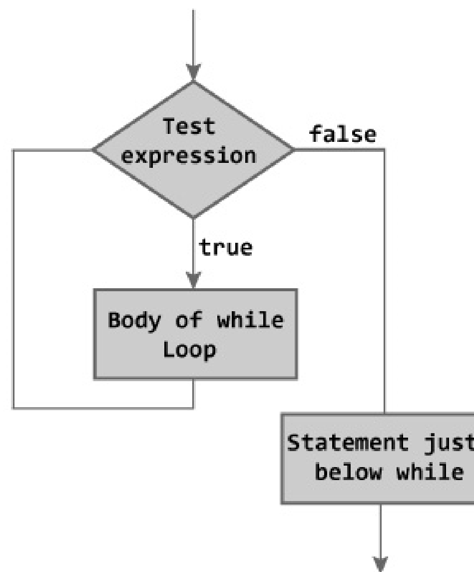
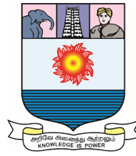


Figure: Flowchart of while Loop

***Example Program:* To read 5 subject marks and find the total**

```
#include<stdio.h>
void main ( )
{
        int i, mark, total = 0;
        i = 1;
        printf("Give 5 Subjects Mark : \n");
        while (i <= 5)
        {
                scanf("%d",  &mark);
                total = total + mark;
```



```
            i++;
        }
        printf ("\Total Mark = %d", total);
}
```

do…while Loop

- ✓ It is an exit checking loop.
- ✓ In do...while loop the test condition is given at the end of the loop. Therefore the body of the loop will be executed at least once.
- ✓ If the test condition is true, then repeat the body of the loop otherwise exit from loop.

Syntax:

```
do
{
        Body of loop statements
} while(test expression);
```

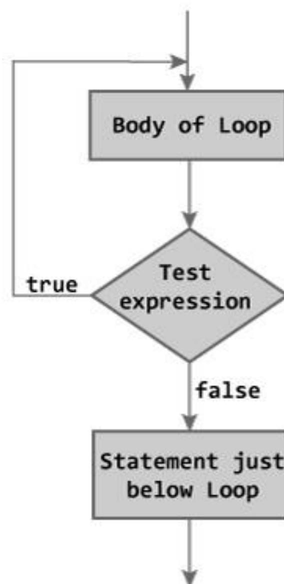
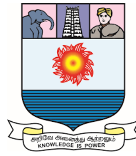


Figure: Flowchart of do...while Loop

Example Program: **Addition of numbers upto 5 by using do…while loop**

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



```
void main ( )
{
        int i = 1, sum = 0;
        do
        {
                sum = sum + i;
                i ++;
         } while (i <= 5)
         printf ("Sum of numbers up to 5 is …%d", sum);
}
```

Output:

Sum of numbers up to 5 is 15

Unit- IV

ARRAYS

Introduction

- ✓ An array is a collection of homogeneous (similar) data items that are stored under one common name.
- ✓ Individual data item (array elements) in an array is identified by index or subscript enclosed in square brackets with array name.
- ✓ The elements in an array are stored in continuous memory location.

DECLARATION OF AN ARRAY

- ✓ Like other variable an array must be declared before they are used so that the compiler can allocate space for them in memory.
- ✓ The syntax for array declaration is:

  data type array_name [size];

- ✓ The data type specifies the array elements data type.
- ✓ Size indicates the maximum number of elements that can be stored in the array.

- ✓ For example

float height [10];

The above array declaration represents the array name is height, we can store a maximum of 10 elements and the array elements are floating point data type.

ARRAY INITIALIZATION

The array elements can be initialized when they are declared like ordinary variables otherwise they will take garbage values.

Syntax: data type array_name [size] = {value 0, value 1, . . . , value n-1)

The initialized values are specified within curly braces separated by commas.

Example: int Marks [3] = {70, 80, 90};

This statement declares the variable Marks as an array of 3 elements and will be assigned the values specified in list as below.

Value	Element
70	Marks [0]
80	Marks [1]
90	Marks [2]

Like ordinary variables, the values to the array can be initialized as follows.

```
int Marks[3];
Marks [0] = 70;
Marks [1] = 80;
Marks [2] = 90;
```

Character array can be initialized as follows:

```
char gender[2] = {'M','F'};
```

Classification of Array

- ✓ One Dimensional Array
- ✓ Two Dimensional Array

One dimensional array:

If the array has only one subscript then it is called one dimensional or single dimensional array.

Syntax:

data type array_name [size];

Declaration Example:

int height[10], weight[10];

char name[20];

Initialization Example:

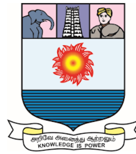
int marks[3] = {20,60,67};

Characteristics of One Dimensional Array

- ✓ Array size must be positive number.
- ✓ Array elements are counted from 0 to size-1.
- ✓ String arrays are terminated with null character ('\0').

Example Program 1:

```
  // Program to print total marks of a student
# include <stdio.h>

void main()
{
        int marks[10], i, n, Total = 0;
        printf ("Enter 5 subjects marks");
        for( i = 0 ;  i < 5 ; i++)
        {
                scanf("%d", &marks[i]);
        }
        for( i = 0 ;  i < 5 ; i++)
        {
                Total = Total + marks[i];
        }
```



```
        printf ("Total mark = %d", Total);
    }
```

Two Dimensional Arrays

If the array has two subscripts then it is called two dimensional array or matrix. Two dimensional arrays are used in situation where a table of values needs to be stored.

Syntax:

```
data type  array_name  [row size] [col size];
```

Declaration Example:

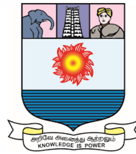
```
int  matrix[5] [5];
char name[10] [20];   //  10 rows 20 columns
```

Initialization Example:

```
int matrix[2][3] = { {2, 6, 7} , {10,  -50, 3} };
```

Example Program:

```
// Program to find the addition of two matrix
#include <stdio.h>
void main()
{
    int i, j, row, col;
    int A[5][5], B[5][5], C[5][5];
    printf("Give the number of Rows & Columns ");
    scanf("%d %d", &row, &col);
    printf("Give A matrix elements row by row\n");
    for ( i = 0 ;  i < row ; i++)
        for ( j = 0 ;  j < col ; j++)
            scanf("%d", &A[i][j]);
```



```
printf("Give B matrix elements row by row\n");
for ( i = 0 ;  i < row ; i++)
        for ( j = 0 ;  j < col ; j++)
                scanf("%d", &B[i][j]);
for ( i = 0 ;  i < row ; i++)
        for ( j = 0 ;  j < col ; j++)
                C[i][j] = A[i][j] + B[i][j];
printf("\n Result Matrix A + B: \n");
for ( i = 0 ;  i < row ; i++)
{
        for ( j = 0 ;  j < col ; j++)
                printf("%d\t", C[i][j]);
        printf("\n");
}
}
```

STRING (ARRAY OF CHARACTERS)

String is a collection of characters. In C language, array of characters are called string. It is enclosed within double quotes. Example: "Give the number of rows"

Declaration of a String

Strings can be declared like a one dimensional array.

Syntax:

```
char string_name[size];
```

For **example**,

```
char name[30];
char dept[20];
```

String Initialization

The string can be initialized as follows:

char dept [] = {‘C’, ‘S’, ‘E’,’\0’};

In the above example, ‘\0’ is a null character and specifies end of the string. Here string is assigned character by character.

OR

char dept[ ] = “CSE”;

In the above example, the size of the string variable is not mentioned. It will be automatically allocated based on the initialized string size.

Functions for Reading a String

The following functions are used to read a single character or a string from the keyboard.

- ✓ scanf()
- ✓ getchar()
- ✓ gets()

i) **scanf()**

The scanf() function is a formatted input function to read a single character or a word from the keyboard. It uses the %s control string. The scanf() automatically terminate the input when it encounter a blank space, tab, new line or carriage return. There is no & operator in scanf () statement to read a string.

Example:

char name[10];

scanf(“%s”, name);

To read a specified number of characters we can use the length of the string in the control string.

scanf(“%10s”, name);

ii) **getchar()**

It is an unformatted single character read function. This function returns

the given input value. So, the returned value must be assigned to a variable.

Syntax:

Variable name = getchar();

Example:

char choice;

choice = getchar();

iii) **gets()**

It is an unformatted string input function. It reads a group of characters from keyboard until an enter key is pressed.

Syntax:

gets(string _variable);

Example:

char line[80];

gets(line);

Functions for Printing a String

The following functions are used to print a single character or a string to the output device (monitor).

- ✓ printf()
- ✓ putchar()
- ✓ puts()

i) printf()

The printf() function is a formatted output function to write a single character or a word to the output device. It uses the %s control string.

Example:

char dept[] = “CSE”;

printf("%s", dept);

printf("%10s",dept); // print the dept string within 10 length space.

ii) putchar()

It is an unformatted single character output function.

Syntax:

char variable_name;

putchar(variable_name);

Example:

char choice = 'Y';

putchar(choice); // it display the character Y

iii) puts()

It is an unformatted string output function.

Syntax:

puts(string _variable);

Example:

char line[80];

puts(line);

Note:

If we use scanf(), printf(), getchar(), putchar(), gets() or puts() function in a program, then we must include the header file stdio.h in our program.

String Handling / Manipulation Function:

Function	Purpose
strlen ()	Used to find length of a string
strcpy ()	Used to copy one string to another
strcat ()	Used to concatenate two strings
strcmp ()	Used to compare characters of two strings
strlwr ()	Convert strings into lower case
strupr ()	Convert strings into upper case
strrev ()	Used to reverse a string

(i) strlen()

It is used to count and return the number of characters present in a string i.e., to find the length of the string. It will not count the end of string (null) value.

Syntax:

```
variable = strlen(string);
```

(ii) strcpy ()

It is used to copy the contents of one string to another string variable.

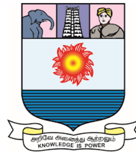
Syntax:

```
strcpy (string1, string2);
```

Here string2 content is copied into string1.

Example Program 5:

```
#include<stdio.h>

#include<string.h>

void main ()

{
```



```
        char source[] = "COMPUTER";
        char target [10];
        strcpy (target, source);
        printf("\n Source string is %s", source);
        printf("Target string is %s, target);
    }
```

Output:

```
Source string is COMPUTER
Target string is COMPUTER
```

(iii) strcat():

It is used to concatenate or combine two strings together.

Syntax:

```
strcat (string1, string2);
```

String2 is concatenated at the end of string1 and the result is stored in string1.

Example Program 6:

```
#include <stdio.h>
#include <string.h>
void main ()
{
    char target[] = " Computer";
    char source [15] = " Programming";
    strcat( target, source);
    printf("After concatenation target string is : %s", target);
}
```

Output:

After concatenation target string is: Computer Programming

(iv) strcmp():

This function compares two strings to check whether they are same or different.

The two strings are compared character by character until end of one string is reached or a mismatch character found.

If two strings are identical, strcmp() returns a value zero

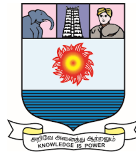
If they are not equal it returns the numeric difference between the first non-matching characters. Therefore, if the strcmp() returns positive then string1 is greater and negative means string2 is greater.

Syntax:

strcmp(string1, string2)

Example Program 7:

```
#include<stdio.h>

#include<string.h>

void main ()

{

    char name1[] = "computer";

    char name2[] = "computer";

    int diff;

    diff = strcmp (name1, name2);

    if (diff == 0 )

        printf("Both strings are identical");

    else

        printf("Both strings are not identical");
```



```
}
```

Output:

Both strings are identical

(v) strrev ()

The strrev() function takes one string argument and return its reverse string.

Syntax:

strrev(string);

Example Program 8:

```
#include<stdio.h>

#include<string.h>

main ()

{

        char str[20] = “Computer”;

        printf(“Given string = %s\n”, str);

        printf (“The reverse string = %s”, strrev(str));

}
```

Output:

Given string = Computer

The reverse string = retupmoC

(vi) strncpy (s1, s2, n)

This function copies the first 'n' number of characters from string s2 into the target string s1.

(vii) strncmp (s1, s2, n)

It compares the leftmost 'n' characters of s1 with s2 and returns zero if both are equal or return negative if s1 < s2 or return positive number if s1 > s2.

(viii) strncat (s1,s2,n)

It appends first 'n' characters of s2 at the end of s1.

(ix) strlwr (s1)

Convert string s1 into lower case alphabet.

(x) strupr (s1)

Convert the string s1 into uppercase alphabet.

Program To check a given string is palindrome or not

```
# include<stdio.h>

#include<string.h>

void main ()

{

        int diff;

        char input[20],  copy[20];

        printf (“Enter input string :”);

        scanf (“%s”, input);

        strupr(input);   // convert into upper case letter

        strcpy(copy, input);  // take a copy of the input string

        strrev(copy);    // reverse the input string

        diff = strcmp(input, copy); //compare input & reverse copy

        if(diff == 0)

                printf (“Given string is a palindrome”);

        else

                printf(“Given string is not a palindrome”);

}
```

Output:

`   Enter input string: malayalam

Given string is a palindrome

Unit-V

FUNCTION

A function is a self-contained block of program statements that performs a specific task. Functions are also called sub programs.

Advantages of Using Function

1. Function code can be reused
2. Redundant code is avoided
3. Better Readability
4. Easy to test & correct errors
5. Easy to maintain

Types of Function:

- ✓ Library Function or Built in Function
- ✓ User defined function

Built in or Library Function

Library functions are also called predefined functions or built in functions. They have been already written by the person who developed the compiler. If we want to use that function include the appropriate library header file and use it.

Example:

```
pow(x,y); // used to find x power y
sqrt (x);  // used to find square root of x
```

Related functions are grouped together and stored within a header file. For example, string handling functions such as strcmp(), strlen(), strcat() etc are stored in string.h header file. So, if we want to use any string handling function, include string.h in the program.

User Defined Function

The user defined functions are created by the user according to their requirements. For example the user can create a function 'read_mat' to read a matrix.

Component of a Function

There are three components in a user-defined function

- ✓ Function Declaration or function prototype
- ✓ Function Definition
- ✓ Function call (or) function invocation

FUNCTION DECLARATION

The function declaration statement is used to declare a function before define & call the function. It identifies a function with its name, list of arguments and the type of data returned. Semicolon is used at the end of a function prototype.

Syntax:

return_type function_name(Parameter list);

Return type may be any one of the data type such as int, float etc or void. Void means the function will not return any data.

Function name is the identifier used to identify the function. Parameter list represent zero or more parameters separated by commas.

The parameter names do not need to be the same in the prototype declaration and the function definition. The data types & order of parameter in the function definition & prototype must match each other. Use of parameter name in declaration is optional.

Example:

int MAX(int A, int B, int C);

FUNCTION DEFINITION

- ✓ The collection of program statements that describe the specific task done by the function is called a function definition.
- ✓ It consists of a function header and a function body.

Syntax:

```
return_datatype functionname (datatype var1, datatype var2…..)   // header
{
        Local  Variable declarations;
        Set of statements;
        return(returndata);
}
```

The Function Header:

The function header consists of three parts.

- ✓ The data type of the returned value
- ✓ The name of the function
- ✓ The formal parameters of the function enclosed between parentheses.

List of parameters specified in the function header of definition is referred as **formal parameters.** Parameter specified in the function call is called as **actual parameter.** Parameters are also called as **arguments**.

If the function does not return a value then return type is specified by the keyword void. The keyword void is also used to indicate the absence of parameters. But it is optional.

A function that has no parameters and does not return a value would have the following header.

```
void  function_name (void)
        ( or )
void function_name()
```

Function Body

- ✓ The body of the function consists of a set of statements enclosed within braces.
- ✓ The body of function can have both executable and non-executable statements.
- ✓ A function can optionally have special executable statements known as return statements.

Return Statement

The return statement is used to return (send) back the result to the calling function.

The general form of the return statement is as follows.

return (expression);

Where 'expression' data type must match with the return type specified in the function header.

If the return type is void, then return statement can be omitted or use empty return statement as follows.

return;

FUNCTION CALL

The functions are called from the main () function or from another function.

The function call statement invokes or calls the function, which means the program control passes to that function.

Once the function completes its task, the program control is passes back to the calling function.

The general syntax is

function name (parameter list)

Example: maximum = MAX(a, b, c);

PARAMETERS

Parameter provides the data communication between the calling function and called function.

i) **Actual parameter:**

These are the parameters used in the function call from the calling function to transfer the data to the called function.

ii) **Formal parameter:**

These are the parameters used in the called function header (function definition).

Example:

```
main ()     // Calling function
{
        --------
        fun1(p,q);    // Function call. Here p & q are actual parameters
        --------
```

```
}
void fun1 (int x, int y) // Called function. Here x & y are formal parameters
{
        --------
}
```

PARAMETER PASSING METHODS

There are 2 ways of parameter passing. They are,

- ✓ Call by Value or Pass by Value
- ✓ Call by Reference or Pass by Reference

Call by Value (Pass by Value)

While calling a function, the values of actual parameters are passed to the formal parameters. Therefore the called function works on the copy and not on original values of actual parameters.

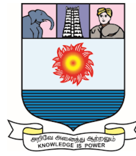
When arguments are passed by value, C allocates separate memory for formal arguments and copy the actual argument value in that location. Therefore the changes made on formal parameters will not affect the actual parameter.

Syntax for call by value method of calling a function:

functionname(arguments separated by comma);

Example:

```
#include<stdio.h>
int calsum (int x,  int y,  int z)
void main ()
{
        int a, b, c, sum;
        printf ("\n Enter any three numbers");
        scanf ("%d%d%d", &a, &b, &c);
        sum = calsum (a, b, c);
        printf("\n Sum = %d",sum);
}
int calsum (int  x, int  y, int  z)
{
```



```
        int d;
        d = x + y + z;
        return (d);
    }
```

Sample output:

```
Enter any three numbers: 10   20   40
    Sum = 70
```

The variables a, b and c are called actual arguments whereas the variables x, y and z are called formal arguments.

Call by Reference (Pass by Reference)

In this method, the address of the actual argument in the calling function are copied into the formal arguments of the called function. That is the actual & formal arguments refer same memory location. Therefore the changes made on formal parameters will affect the actual parameter of the calling function.

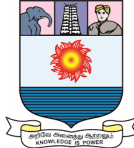
Example:

// Swapping of numbers:

```
#include<stdio.h>
void swap(int *, int *)
void  main()
{
        int A = 10, B = 20;
        printf("Before swap A = %d, B = %d \n", A,B);
        swap (&A, &B);
        printf("After swap A = %d, B = %d", A, B);
}
void swap (int *x, int *y)
{
        int temp;
        temp = *x;
        *x = *y;
        *y = temp;
}
```

Sample output:

```
Before swap A = 10, B = 20
After   swap A = 20, B = 10
```



Difference between Call by Value & Call by Reference

Call by Value	Call by Reference
Actual & formal parameters refer **different memory** location	Actual & formal parameters refer **same memory** location
Changes made in formal parameters **not reflected** back to the calling function	Changes made in formal parameters **reflected** back to the calling function

CATEGORY OF FUNCTIONS

Depending on whether arguments are present or not and whether a value is returned or not, functions are categorized into −

- Functions with no arguments and no return values
- Functions with arguments and no return values
- Functions with arguments and one return value
- Functions with no arguments but return a value
- Functions that return multiple values

NESTING OF FUNCTIONS

A nested function is a term used to describe the use of one or more functions inside another function. In C language, defining a function inside another one is not possible. In short, nested functions are not supported in C. A function may only be declared (not defined) within another function.

When a function is declared inside another function, it is called lexical scoping. Lexical scoping is not valid in C because the compiler cannot reach the correct memory location of inner function.

Example

```
#include <stdio.h>
#include <math.h>
double myfunction (double a, double b);
int main()
```

```
{
  double x = 4, y = 5;
  printf("Addition of squares of %f and %f = %f", x, y, myfunction(x, y));
  return 0;
}

double myfunction (double a, double b){
  auto double square (double c) { return pow(c,2); }
  return square (a) + square (b);
}
```

In this program, a function square() is nested inside another function myfunction(). The nested function is declared with the auto keyword.

Output

Addition of squares of 4.000000 and 5.000000 = 41.000000

RECURSION

- A function calls the same function is called recursion.

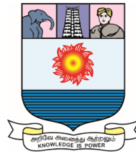
Advantages of Recursion

- ✓ It helps to write simple version of a program.
- ✓ Recursion will reduce the program size.

Program Using Recursion

Example Program 1 : Find factorial using recursion

```
#include<stdio.h>

int fact(int);
void main()
{
        int n, Result;
```



```
        printf("\n Enter any number:");
        scanf("%d", &n);
        Result = fact(n);
        printf ("Factorial value = %d", Result);
}
int fact (int n)
{
        int f;
        if (n == 1)
                return (1);
        else
                f = n * fact (n – 1);
        return (f);
}
```

Output:

Enter any number: 4
Factorial value = 24

Example Program 2 : Write a program using recursive function to generate the Fibonacci series

```
#include<stdio.h>
int fib(int val);
void main ()
{
        int i, n;
        printf("Enter the number of terms:");
        scanf ("%d", &n);
        printf("\n Fibonacci sequence for %d terms are :", n);
        for (i = 0 , i < n, i++)
                printf("%d  ", fib(i));
}
int fib (int j)
{
        if (j == 0)
        {
                return(0);
```

```
        }
        If (j == 1)
        {
                return(1);
        }
        return ( fib(j – 1) + fib (j – 2));
    }
```

Output

```
Enter the number of terms: 6
Fibonacci sequence for 6 terms are: 0  1  1  2  3  5
```