Srinivas Institute of Management Studies

Pandeshwar, Mangalore – 575 001

BACKGROUND STUDY MATERIAL

PROGRAMMING IN 'C'

BCA I Semester

Compiled by

Prof.Panchajanyeswari M Achar

SIMS, Mangalore

2015-16

CONTENTS

UNIT I
CHAPTER 1
OVERVIEW OF C LANGUAGE

1.1 History of C:

C is an offspring of Basic Combined Programming Language BCPL called B developed in 1960's at Cambridge University. B language was modified and written by Dennis Ritchie and the language was called C. C language was developed and implemented at AT&T's Bell laboratory in the year 1972. Over the years C became very popular because it was reliable, simple and easy to use.

1.2 Importance of C Language:

- C is a robust language with a rich set of built in functions and operations that can be used to write any complex program.
- C compilers combine the features of assembly language and the capabilities of high level language. Therefore, it is well suited for writing system software and business packages.
- C supports a variety of data types and operators. Hence programs written in C are fast and efficient.
- C supports only 32 keywords and its strength lies in its built in functions.
- C language is portable i.e. C programs written in one system can be run on any system with little or no modifications.
- C language is best suited for structured programming. Thus, the user has to think of a problem in terms of functional modules or blocks. This modular structure makes program debugging, testing and maintenance easier.
- C has the ability to extend itself. A C program is basically a collection of functions that are supported by C library. We can continuously add our own functions to the C library.

1.3 Structure of a C Program:

- The documentation section consists of a set of comment lines. Lines beginning with /* and ending with */ are called comment lines. They are used to enhance readability of a program. Comment lines are ignored by the compiler and are not executable statements. The documentation section gives details of the program.
- The link section provides instructions to link the functions from system library. The statements in this section include or the files necessary to run the programs.

- The definition section is used to define all the symbolic constants in a program

- Global declaration section declares all the global variables in the program. A global variable is a variable that is used in more than one function in a program. These variables are declared outside so that they can be accessed by all the functions in the application.
- Every C program must have only one main() function. It indicates that the name of the function is main and that the program execution should begin here. The empty parenthesis () that follow main indicate that the function has no arguments. The actual execution of the program begins here.
- This section consists of 2 parts – declaration part and execution part. All the variables that are used in the program are declared in the declaration part. There must be at least one statement n the execution part. These 2 parts are enclosed between curly braces{}. Program execution begins at the opening brace { and the closing brace } indicates the logical end of the program. Ever statement in this ection are terminated by a semicolon(;).
- Every C program must have a main() section.
- The sub program section contains all the user defined functions that are called in the main function.

Documentation Section
Link Section
Definition Section
Global Declaration Section

main( ) Function Section
{
Declaration part
Execution part
}

Subprogram section Function 1 …… Function n

Sample Program 1:

```
/*Documentation Section*/
/*Sample C Program to find area of a circle*/
/*Link Section*/
#include<stdio.h>
/*Definition Section*/
#define PI 3.141592
/*main function section*/
void main()
{
        int rad;       /*declaration part*/
        float area;
        clrscr();
        rad=14;        /*execution part*/
        area=PI*rad*rad;
        printf("Area:%f",area);
        getch();
}
```

Output:
Area : 615.752014

Sample Program 2:

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

Output:
Hi Jahnavi

1.4 Rules for writing a C program:

- C program statements must be written in lower case letters. Only symbolic constants are written in uppercase letters.
- Every statement should end with a semicolon.
- Braces are used to group and mark together the beginning and end of functions.
- Proper indentation of braces and statements make the program easier to read and debug
- C is a case sensitive language.
- C is a free-form language. i.e. More than one statement can be written on one line separated by a semicolon.

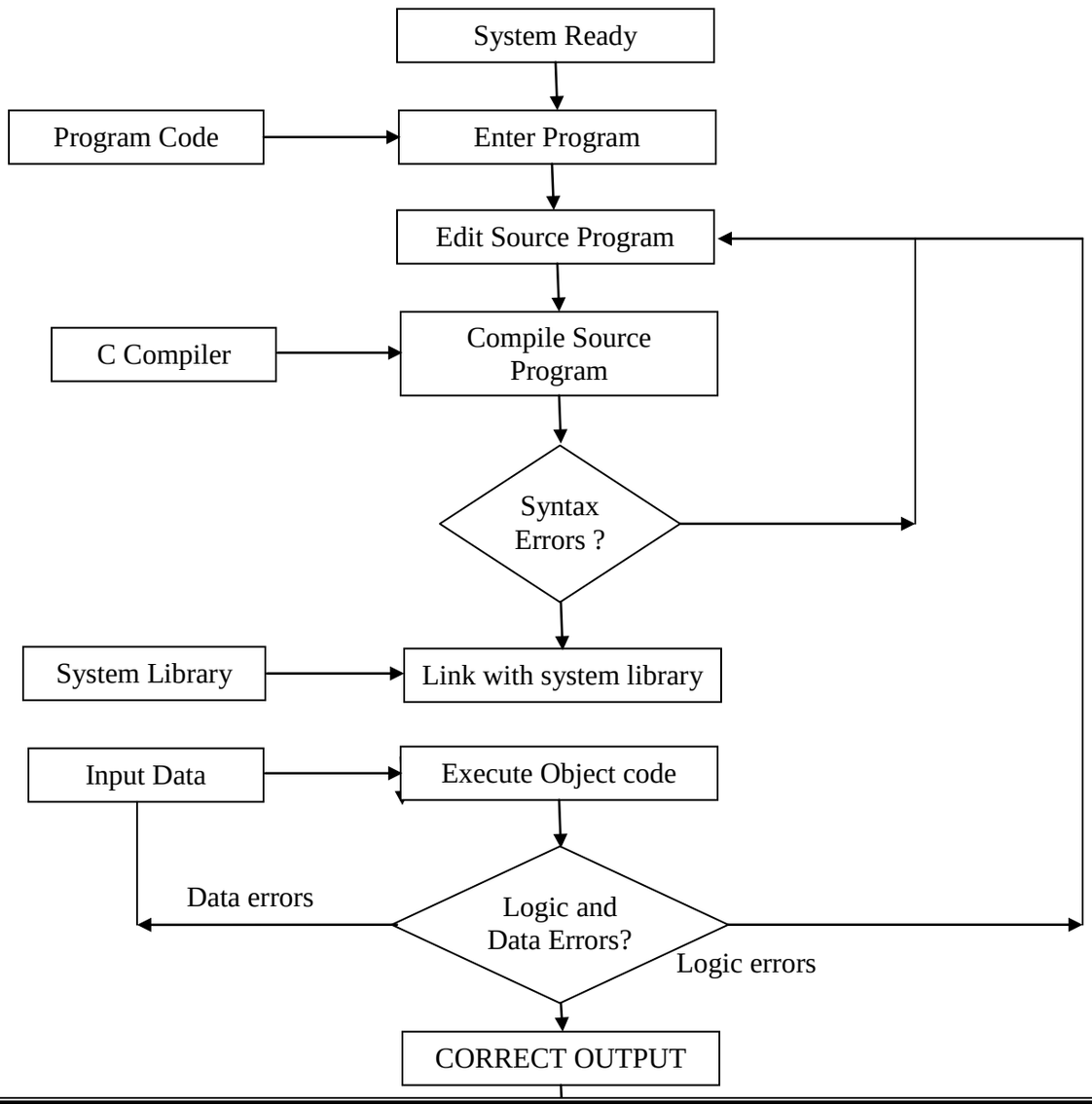
1.5 Executing a C Program:

The various steps involved in executing a C program are

1. Create the program

2. Compile the program
3. Link the program with the various functions available in the C library
4. Execute the program.

- The programs are first typed in an editor.
- Once editing is over, the program is compiled. The program is given to the compiler. The compiler checks for any syntax errors in the source code and generates object code if no errors are found. Otherwise errors are displayed to the user for debugging.
- During linking various system files are included in the object code from the system library. Once linking is over, executable object code is generated.
- The executable code is executed by giving input data to it. If no logical or data errors are found then the correct output is generated.



Assignment

1. Explain the structure of a C program.(2006, 2007,2010,2013)
2. List the features of C (2008,2011)
3. What are the rules to write a C program?

CHAPTER 2

CONSTANTS, VARIABLES AND DATA TYPES

2.1 Character Set
2.2 C tokens
2.3 Keywords and identifiers
2.4 Constants
2.5 Variable
2.6 Data types
2.7 Declaration of variables

2.8 Assigning values to variables
2.9 Defining symbolic constant

2.1 Character set:

The characters that are used to form statements and expressions in a programming language is known as character set. The characters set in C are grouped as follows.

1. Letter- uppercase A-Z and lowercase a-z
2. Digits- 0-9
3. Special characters
4. White spaces.

2.1.1 Trigraph characters:

- Certain characters that are not available on some keyboards can be used in C using trigraph characters.
- Each trigraph sequence consists of 3 characters i.e. two question marks followed by another character.
- For example if the keyboard doesn't support square brackets, we can use a trigraph sequence ??(and ??) for [and] respectively.

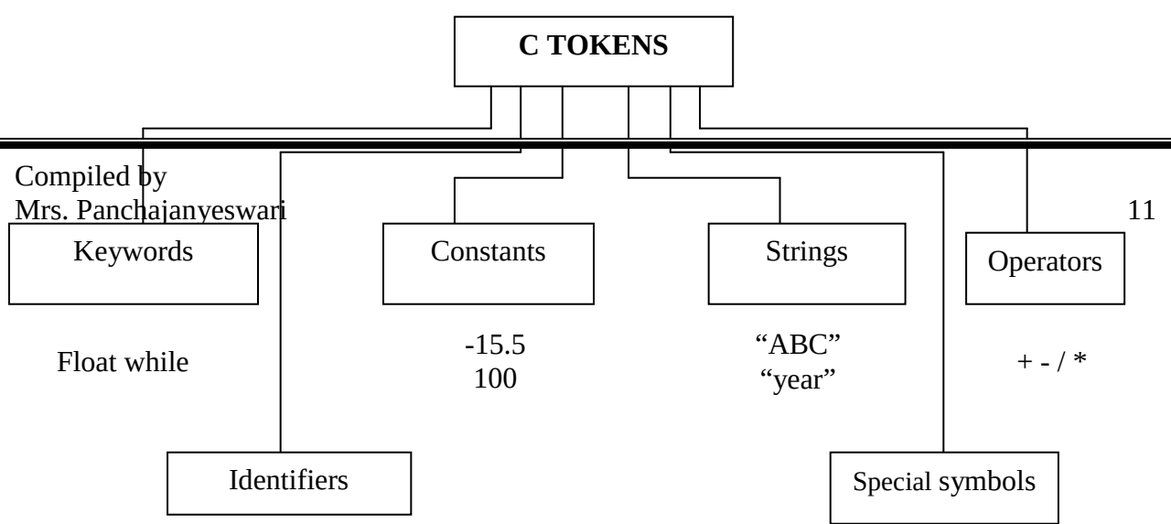
2.2 C Tokens:

The smallest individual units in a program are known as tokens. C supports 6 types of tokens. They are

- Keywords
- Identifiers
- Constants
- Strings
- Special symbols
- Operators

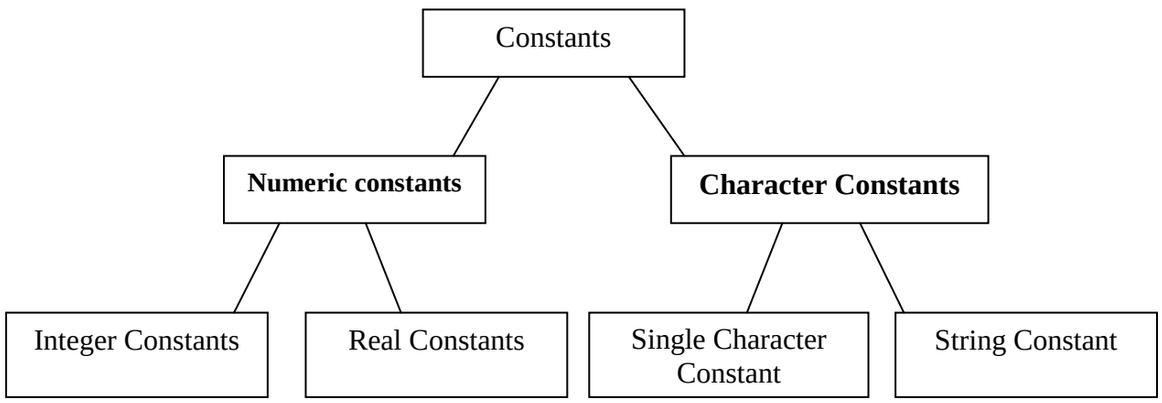
2.3 Keywords and identifiers:

- Every C word is classified either as a keyword or an identifier
- All keywords have a fixed predefined meaning that cannot be changed. Hence keywords are also known as reserve words.
- All keywords must be written in lower case.
- C supports a set of 32 keywords
- Identifiers are names given to variables, functions etc
- These are user defined names consisting of a sequence of letters and digits. But the first character in an identifier must be a letter.
- Blank spaces are not permitted in identifiers. An underscore(_) can be used to join 2 words in an identifier.



2.4 Constants:

A constant refers to a fixed value that does not change during the execution of a program. Integer constants refer to a sequence of digits preceded by an optional +/- sign. E.g. +123, -567, 0. Decimal integers consist of a set of digits 0-9. An octal integer consists of combination of digits from 0 to 7 with a leading 0. E.g. 032, 056,0435. A hexadecimal integer consists of 0-9 digits and alphabets A through F to represent numbers from 10-15. A hexadecimal integer is preceded by 0x or 0X. E.g. 0xF2, 0X35A, 0x56



2.4.1 Real constants:

- Numbers containing fractional part are called real or floating point constants.
- These can be represented either in decimal notation or exponential notation.
- In decimal notation a whole number is followed by a decimal point and a fractional part. E.g. 0.045, -0.25
- A real number can also be expressed in scientific or exponential notation as mantissa e exponent.

- The mantissa could be either a real number expressed in decimal notation or an integer. The exponent is an integer with an optional +/- sign. E.g. 0.65e4, 1.25e-5, 56.23E-1

2.4.2 Character constants:

- Character constants are either single character constants or string constants
- Single character constants consist of a single character in single quotes. E.g. ‘a’, ‘Q’, ‘5’
- String constants are a group of characters enclosed in double quotes. The characters may include letters, numbers or special characters. E.g.“abc”, “1956”, “Hello!”

2.4.3 Backslash character constants:

- C supports backslash character constants that are used in output function.
- A backslash character constant consists of a backslash followed by a character. These character combinations are also known as escape sequences

Constant	Meaning
‘\a’	Audible alert
‘\n’	New line
‘\t’	Horizontal tab space
‘\0’	Null character
‘\’’	Single quote

2.5 Variables:

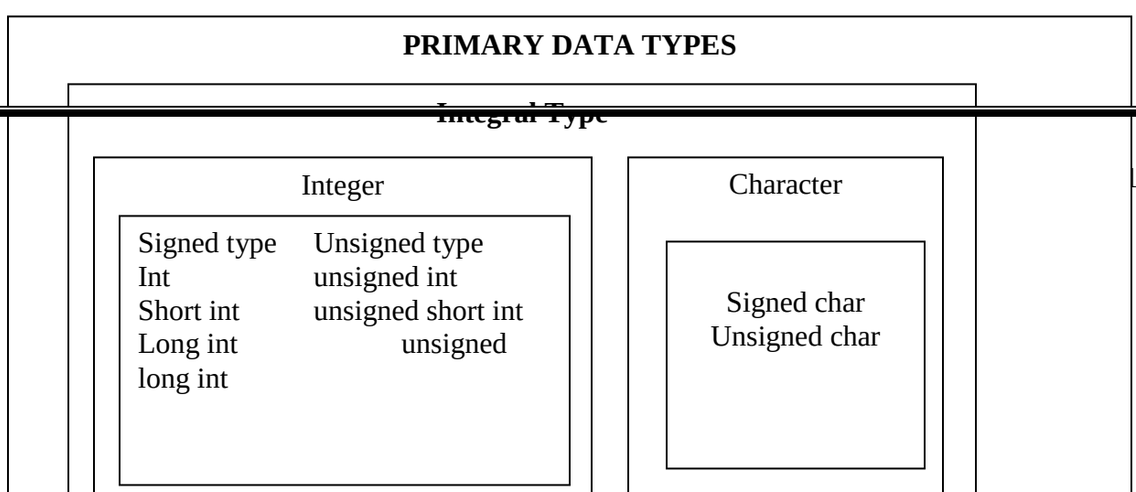
- A variable is a data name that may be used to store a data value
- A variable may take different values at different times during execution.
- The value of a variable changes during the execution of a program

Rules for naming a variable:

- Variable names should begin with a letter.
- Variable names should not be keywords.
- Blank spaces are not allowed.
- Uppercase and lowercase letters are significant because C is a case sensitive language.
- Length of a variable name should not exceed 8 characters

2.6 Data Types:

- C language is rich in its data types. A variety of data types allow the programmer to select the type according to the application.
- The 4 basic data types supported by C are
    1. Fundamental / Primary Data type
    2. User defined Data type
    3. Derived data type
    4. Empty data set



2.6.1 Primary Data Types:

- All C compilers support this data type. Primary data types can be classified as follows
    1. Integer
    2. Floating point
    3. Character

Data type	Range of Values
Char	-128 to +127
int	-32,768 to +32,767
float	3.4e-38 to 3.4e+38
Double	1.7e-308 to 1.7e+308

2.6.1.1 Integer Data Type:

- Integers are whole numbers with a range of values. Integers occupy one word of storage
- In a 16-bit machine, the range of integers is -32768 to +32767
- In order to provide some control over a range of numbers and storage space, C has 3 classes of storage for integers namely, short int, int and long int in both signed and unsigned format

- Short int is used for small range of values and requires half the amount of storage as a regular int number uses.
- Long int requires double the storage as an int value.

- In unsigned integers, all the bits in the storage are used to store the magnitude and are always positive

2.6.1.2 Floating point Data Type:

- All floating point numbers require 32 bits with 6 digits of precision
- They are defined in C using the keyword float
- For higher precision, double data type can be used.
- Double data type gives a precision of 14 bits and requires 64 bits

2.6.1.3 Character Data Type:

- A single character can be defined as a character data type using the keyword **char**
- Characters usually need 8 bits for storage

Primary Type Declaration:

Syntax for declaring variables:

Data type v1,v2,….vn

Here v1,v2…vn are the variable names.
Variable names must be separated by commas. Every declaration statement must end with a semi-colon

E.g. : int count;
float amt,tot;

2.6.2 User Defined Type:

- C supports a feature known as 'type definition' that allows user to define an identifier that would represent an existing data type.
- The user defined data type identifier can later be used to declare variables

Syntax:

typedef type identifier;

type refers to data type supported by C and identifier is a new name given to that data type.

- typedef cannot create a new data type. It assigns a new name given to the existing data type
- The main advantage of typedef is that it provides better readability of the program.
  E.g.: typedef int units;
  typedef float marks;

  units batch1,batch2;
  marks m1,m2,m3;
- Another type of user defined data type is enumerated data types
  Syntax: enum identifier {value1,value2,…,valuen};

- The values enclosed in braces are known as enumeration constants
- The identifier can be used to declare variables that can have only one of the values enclosed in braces.
  E.g.: enum day{Sun,Mon,….Sat};
- The compiler automatically assigns integer digits beginning with 0 to all the enumeration constants
- We can explicitly assign integer values to enumeration constant as follows
  enum day{Sun=2,Mon,…,Sat};

2.6.3 Derived Data Types:

- These data types are derived from existing data types according to the need of the application
- They can contain a group of elements that belong to the same type or different data types

E.g.: arrays, structures

2.6.4 Empty Data Set:

- This data set corresponds to null values and is denoted by the key word **void**.
- When void keyword is used in functions as a type specifier, it means that the function returns nothing. i.e. When a function doesn't return values corresponding to a particular data type void is used.

2.7 Declaration of a Variable:

- Variables must be declared before they are used in the program
- Declaring a variable does two things :
  1. It tells the compiler what the name of the variable is
  2. It specifies what type of data that variable will hold.

2.8 Assigning values to Variables:

- Values are assigned to variables using the assignment operator '='

Syntax: var-name=value;
E.g.: x=10;

- It is also possible to assign values to a variable during declaration. The process of assigning initial values to variables is known as initialization.

Syntax: datatype var-name=value;
E.g. : int x=10;
char x= 'y';

Declaring variable as a constant:

- Sometimes in programming, we would like variable values unchanged during program execution. This can be achieved by declaring the variable with a qualifier constant at the time of initialization.

Syntax: const datatype var-name=value;
const int x=40;

2.9 Defining Symbolic Constants:

- Sometimes constant values may appear repeatedly in a number of places in a program
- In such cases we encounter problems of modification and understanding
- If the value is to be altered the value must be modified at each and every place in the program.
- In order to such situations, C supports symbolic constants or constant identifiers
- They are not declared in the declaration section
- A symbolic constant is declared as follows:
  # define symbolic-constant value-of-constant
  E.g. # define PI 3.141592
- In the above statement #define is a preprocessor compiler directive. The compiler will substitute the value 3.141592 in all places where it encounters the symbol PI
- One statement must be written for every symbolic constant.

Assignment

2 mark Questions:

1. Define variable.(2005,2007, 2008,2011,2012)
2. Define constant (2007,2008)
3. What are trigraph characters? Give example.
4. What is a backslash constant? Give example. (2009, 2011)
5. Define keyword. Give example. (2005,2007, 2008)
6. What is a token in C? Give example
7. List the character set in C.
8. How do you declare a variable in C? Give example (2004,2009,2010)
9. How do you assign a value to a variable? Give example.
10. How do you declare a variable as a constant? Give example (2010,2009)
11. How do you define a symbolic constant in C? Give example
12. Differentiate between keyword and identifier(2007,2009,2011)
13. What is the purpose of typedef in C?
14. What is the purpose of enum keyword in C?

5 mark Questions:

1. Explain the primary data types in C. (2008,2009)
2. Explain the various types of constants in C.(2006,2007)
3. Explain about tokens available in C.(2008)
4. How do you declare a variable in C? What are the rules to form a variable?(2008)
5. Write a note on user defined data types in C.

CHAPTER 3

OPERATORS AND EXPRESSIONS

C supports a rich set of operators. An operator is a symbol that tells the computer to perform an arithmetic or logical function. Operators are used in programs to manipulate data and variables. Operators in C can be classified into a number of categories. They are

- ✓ Arithmetic Operators
- ✓ Relational Operators
- ✓ Logical Operators
- ✓ Assignment Operators
- ✓ Increment and Decrement Operators
- ✓ Conditional Operators
- ✓ Bitwise Operator
- ✓ Special Operators

3.1 Arithmetic Operators:

All the basic arithmetic operators are supported by C. These operators can manipulate with any built in data types supported by C. The various operators are tabulated as follows:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Division

3.1.1 Integer Arithmetic:

When both the operands in the expression are integers, then the operation is known as integer arithmetic.

For example if the values of a=14 and b=4 then

a+b=18 , a-b=10, a*b=56 , a/b=3, a%b=2

During modulo division, the sign of the result is the sign of the first operand.

E.g. -14%3=-2 , 14%-3=2

3.1.2 Real Arithmetic:

An arithmetic operation involving only real operands is called real arithmetic. A real value can assume value either in decimal or exponential notation. % operator cannot be used with real operands

E.g. If a=2.4 and b=1.2 then

a+b=3.6 , a-b=1.2, a*b=2.88 a/b=2.0

3.1.3 Mixed mode Arithmetic:

When one of the operands is real and the other is an integer, the expression is called mixed mode arithmetic expression. The result is always a real number

E.g. : 15/10.0=1.5 , 15/10=1.5

3.2 Relational Operators:

Comparisons can be done using relational operators. A relational expression contains a combination of arithmetic expressions, variables or constants along with relational operators. A relational expression can contain only two values i.e. true or false. When the expression is evaluated as true then the compiler assigns a non zero value and 0 otherwise. These expressions are used in decision statements to decide the course of action of a running program.

Syntax: ae1 relational operator ae2

where ae1 and ae2 are arithmetic expressions

Operators	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

3.3 Logical Operators:

An expression that combines two or more relational expressions is called a logical expression and the operators used to combine them are called logical operators.

Syntax: R1 operator R2

where R1 and R2 are relational expressions

Operator	Meaning
&&	Logical And
||	Logical Or
!	Logical Not

Relational Expression R1	Relational Expression R2	R1&&R2	R1||R2
0	0	0	0
0	Non zero	0	Non zero
Non zero	0	0	Non zero
Non zero	Non zero	Non zero	Non zero

3.4 Assignment Operators:

Assignment operators are used to assign the result of an expression to variable. The operator used for assignment in C is '='. C also supports short hand assignment operators like **v op = expression** where v- variable and op- operator. Here op= is known as a short had assignment operator.

v op=expression is equivalent to v=v op expression

E.g. The statement x+=y+1; is equivalent to x=x+y+1;

The advantage of short hand assignment operator is

a) Easy to read and write
b) Statements are more concise and efficient

3.5 Increment/Decrement Operators:

C has 2 very powerful operators that are not found in any other language. They are increment/decrement operators i.e. ++ and --. The ++ operator is used to increment value of a variable by 1. The -- operator is used to decrement value of a variable by 1. Both are unary operators and can be written as follows: ++m, m++ , m—and –m. Both m++ and ++m mean the same thing when they form statements independently. But, they behave differently when used in expression on the right hand side of assignment operator.

E.g. Let a=5; x=a++; Here, this statement can be broken into 3 statements as follows

a=5; x=a; a=a+1;

In the above example, the value of a is assigned to x and then its value is incremented by 1. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on the left. A postfix operator first assigns its value to the variable and then increments its value by 1.

E.g. m=10;

y=--m

Here the value of m is decremented by 1 and then assigned to y. Hence the value of y is 9.

3.6 Conditional Operator:

Conditional operators are also called ternary operators. Conditional expressions can be constructed in C using the operator pair '?:'.

Syntax: expr?expr1:expr2;

Here expr is evaluated first. If the value is true, then expr1 is evaluated and becomes the value of the expression. If the expression is false then, expr2 is evaluated and becomes the value of the expression.

E.g. a=5;

```
    b=10;
    x=(a>b)?a:b;
```

The output of the above example is as follows- The value of b i.e. 10 is assigned to x.

Program:

```
#include<stdio.h>
void main()
{
        int a,b,small;
        printf("Enter value for a");
        scanf("%d",&a);
        printf("Enter value for b");
        scanf("%d",&b);
        small=(a<b)?a:b;
        printf("%d is small",small);
        getch();
}
```

3.7 Bitwise Operators:

These operators are used to manipulate data at bit level. These operators are used for testing the bits or shifting the bits either to the left or right.

Operator	Meaning
&	Bitwise And
|	Bitwise Or
^	Bitwise exclusive Or
<<	Shift left
>>	Shift Right
~	One's complement

3.8 Special Operators:

C supports other special operators like pointer operator and member selection operator. Comma is also an operator use to link related expressions together. The sizeof operator is a compile time operator. It returns the number of bytes occupied by the operand depending on the data type. The operand could be a variable, constant or a data type qualifier.

E.g.: m=sizeof(x);
n=sizeof(double);

Sample Program:

```
#include<stdio.h>
void main()
{
        float x;
        printf("The size of variable x is %d bytes",sizeof(x));
        getch();
```

}

Output:
The size of variable x is 4 bytes

3.9 Arithmetic Expressions:

An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language. Expressions are evaluated using assignment statements of the form

variable=expression;

Expressions are always evaluated from left to right, using the rules of precedence of operators, if parenthesis are missing

High Precedence - * / %
Low Precedence - + -

Basically the evaluation procedure involves 2 left to right passes. During the first pass the high priority operators are applied as they are encountered and during the second pass, lower priority operators are applied as they are encountered.
Example: Consider the following expression x= a-b/3+c*2-1
If the values of a=9, b=12,c=3
Now x= 9-12/3+3*2-1
Pass 1: x=9-4+6-1
Pass 2: x=5+6-1=10

3.10 Precedence of operators:

If more than one operators are involved in an expression then, C language has predefined rule of priority of operators. This rule of priority of operators is called operator precedence.

In C, precedence of arithmetic operators(*,%,/,+,-) is higher than relational operators(==,!=,>,<,>=,<=) and precedence of relational operator is higher than logical operators(&&, || and !). Suppose an expression:
(a>b+c&&d)

This expression is equivalent to
((a>(b+c))&&d)
i.e. (b+c) executes first
then, (a>(b+c)) executes
then, (a> (b=c)) && d) executes

Associativity of operators
Associativity indicates in which order two operators of same precedence(priority) executes. Let us suppose an expression:
a==b!=c

Here, operators == and != have same precedence. The associativity of both == and != is left to right, i.e, the expression in left is executed first and execution take pale towards right. Thus, a==b!=c equivalent to :
(a==b)!=c
The table below shows all the operators in C with precedence and associativity.
Note: Precedence of operators decreases from top to bottom in the given table.

Summary of C operators with precedence and associativity

Operator	**Meaning of operator**	**Associativity**
() [] -> .	Functional call Array element reference Indirect member selection Direct member selection	Left to right
! ~ + - ++ -- & * sizeof (type)	Logical negation Bitwise(1 's) complement Unary plus Unary minus Increment Decrement Dereference Operator(Address) Pointer reference Returns the size of an object Type cast(conversion)	Right to left
* / %	Multiply Divide Remainder	Left to right
+ -	Binary plus(Addition) Binary minus(subtraction)	Left to right
<< >>	Left shift Right shift	Left to right
< <= > >=	Less than Less than or equal Greater than Greater than or equal	Left to right
== !=	Equal to Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional Operator	Left to right
=	Simple assignment	Right to left

Summary of C operators with precedence and associativity

Operator	**Meaning of operator**	**Associativity**
*= /= %= -= &= ^= |= <<= >>=	Assign product Assign quotient Assign remainder Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	
,	Separator of expressions	Left to right

3.11 Type Conversions in Expressions:

a) Automatic Type Conversions: If the operands are of different data types, the lower data type is automatically converted to a higher data type before the operation proceeds. The result is of a higher data type.

b) Type casting: The process of local conversion is known as casting a value.
Syntax: (typename) expression;
where typename is one of the standard data types in C.
The type of that variable will not be changed in other parts of the program.
Example: ratio=no_female/no_male;
If no_female and no_male are declared as int, the value of ratio will not be accurate as the decimal part will be lost. For the division to take place accurately, we need to typecast the value to floating point mode as follows:
ratio=(float)no_female/no_male;

3.12 Built-in Mathematical functions:

Mathematical functions such as cos, tan and sqrt are widely used in problem solving. All these functions are available in the <math.h> header file. It is necessary to include this header file in the program in order to use these functions. The list of mathematical functions available is given below:

Function	Meaning
ceil(x)	x rounded up to the nearest integer
exp(x)	e to the power of x
fabs(x)	Absolute value of x
floor(x)	x rounded down to the nearest integer
log(x)	Natural log of x, x>0
pow(x,y)	x raised to y (x^y)
sqrt(x)	Square root of x ; x>=0

Assignment:

2 mark questions

1. What is the difference between pre-increment and post-increment operators (2006)
2. If the value of a=5 then, explain a—and –a.

3. What is short hand assignment in C? Give example(2007,2009)
4. What is the purpose of ternary operator? Give example(2006,2010)
5. What do you mean by typecasting? Give example
6. List any four mathematical functions in C.
7. What is the purpose of size of operator?(2005,2006,2009,2010)
8. Why are bitwise operators used in C?
9. If a=10, b=20 evaluate the following statements in C
    a. a+=10;
    b. a+4/6*6/2;
    c. b+3/2 *2/3;
    d. a+=b;
    e. c=a++;
    f. d=--b;

5 mark Questions

1. Explain about arithmetic operators.(2005)
2. Write a note on bit-wise operators.
3. Explain about logical operators
4. Write a note on relational operators.(2007)
5. Explain about increment operators in C.(2008)
6. Explain about decrement operators in C.(2009)

UNIT II

CHAPTER 4

MANAGING INPUT/OUTPUT OPERATIONS

4.1 Reading and Writing a Character

4.1.1 Reading a Character:

The simplest of all input/output operations is to read and write a character on the standard input/output units. The C compiler treats all standard input/output units as separate files. Reading a single character from the keyboard is done using the getchar( ) function.

Syntax: var-name=getchar( );

Here var-name is a valid C variable declared as a character data type.

When this statement is encountered, the computer waits until a key is pressed and then assigns this character as a value to the getchar function.

E.g. char ch;

ch=getchar( );

In the above example, if 'Y' is given as input via keyboard, the getchar function accepts it and assigns this value to the variable ch.

4.1.2Writing a Character:

putchar( ) is the function used to display one character at a time on the terminal.

Syntax: putchar(var-name);

Here var-name is a valid C variable declared as a character data type.

E.g. ch= 'n';

putchar(ch);

The above statements display the character 'n' on the monitor.

4.2 Formatted Input:

Formatted input refers to an input data that is arranged in a particular format. The function used to accept formatted input data from the console is scanf.

Syntax:

scanf("format specifier", arg1,arg2,……argn);

Here format specifier specifies the field format in which the data is to be entered. arg1,arg2,……argn specify the address of the location where data is stored. The format specifier contains field specifier consisting of conversion character % , data type specifier and an optional number specifying the field width.

Inputting Integer Numbers:

The field specification for inputting an integer number is %wd where %-conversion character, w- integer number that specifies the field width and d- type specifier for integers

E.g.: scanf("%2d%5d",&n1,&n2);

Suppose the data line inputs numbers 15, 87649. Then,15 is assigned to n1 and 87649 is assigned to n2.

An input field can be skipped by specifying * in the place of field specifier. The compiler then, ignores the input value given at that place. In order to read long integers the type specifier is ld. In order to input real numbers the type specifier is f. In order to input double values the type specifier is lf. Character strings can be input using %wc or %ws. %wc is used to read a single character and %ws is used to read a group of characters.

4.3 Formatted output:

printf function is used to display the output on to the terminals.

Syntax:

printf("control string", arg1,arg2,….argn);

Here the control string contains the following:

- Message to be printed
- Format specifiers that define the output format of each data
- Escape sequences like \n,\t to format the output as desired.

Arguments arg1,arg2,….argn that follow are variables whose values are formatted and printed according to the specifications in the control string. The arguments must match in number, order and type with the format specifications. A simple format specifier is as follows:

%w.p type specifier

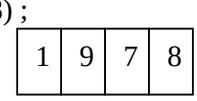
Here w is the integer value that specifies the total number of columns for output value. p is the integer number that specifies the number of digits to the right of decimal point(in case of real numbers) or the number of characters to be printed from a string.

4.3.1 Output of Integers:

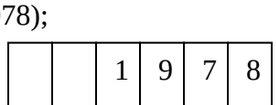
The format specification is %wd, where w is the minimum field width required by the output. If the number of digits in the number is greater than the specified field width, the number will be printed in full. d specifies that the value to be printed is an integer value. The number is always by default right justified in the given field width. The number can be made left justified

by placing a minus (-) sign after %. We can also pad the leading spaces by 0 by placing a 0 before a field specifier.
Examples:

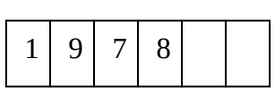
printf("%d",1978) ;
Output:



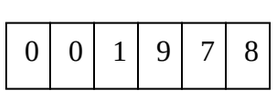
printf("%6d",1978);
Output:



printf("%-6d",1978);
Output:

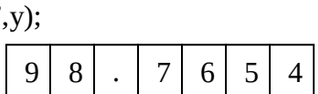


printf("%06d",1978);
Output:

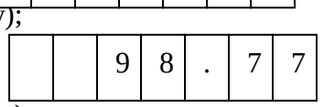


4.3.2 Output of Real Numbers:

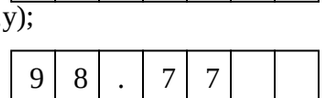
The output of a real number in decimal notation can be displayed using %w.pf where w is the mnimum number of positions to display the value. p is the number of digits to be displayed after the decimal point. In order to display a real number in exponential notation we use %w.pe. The default precision is 6 places. Padding of leading blank spaces with 0 or left justification of the output is possible using 0 or – before the field specifier w respectively.
Example: To illustrate the output of a number y=98.7654
printf("%7.4f",y);



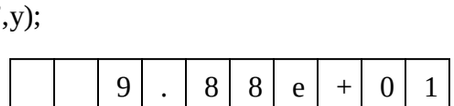
printf("%7.2f",y);



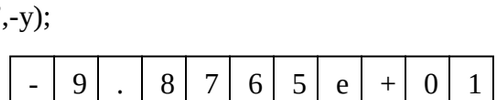
printf("%-7.2f",y);



printf("%10.2e",y);



printf("%11.4e",-y);



4.3.3 Printing of strings:

The format specification of strings is similar to that of real numbers. It is of the form %w.ps where w-field width and p instructs that only the first p characters should be displayed. The display is by default right justified.

Example:

To illustrate the display of the string city= "CHENNAI"

printf("%s",city);

C	H	E	N	N	A	I

printf("%10s",city);

			C	H	E	N	N	A	I

printf("%10.4s",city);

						C	H	E	N

printf("%-10s",city);

C	H	E	N	N	A	I			

Printing of Single Character:

A single character can be displayed using %wc. The characters are displayed in a right justified manner in the field of w columns. The display can be made left justified by placing a minus (-) sign before the integer before the integer w.

Assignment

2 mark Questions:

1. What is purpose of getchar() function in C? (2006,2008,2011)
2. What is purpose of putchar() function in C?(2005,2008)
3. Give the syntax of scanf with example. (2005,2008,2011)
4. Give the syntax of printf with example.(2007)

5 mark Questions:

1. Explain how to get formatted output for real numbers in C with examples.(2011)
2. Explain how to get formatted output for strings in C with examples.(2012)
3. Explain how to get formatted output for integer numbers in C with examples.(2005)

CHAPTER 5

DECISION MAKING AND BRANCHING

5.1 Simple if statement
5.2 If..else statement
5.3 Nested if statement
5.4 Else-if ladder
5.5 Switch statement
5.6 Conditional statement (?: operator)
5.7 Goto statement
Assignment questions

Introduction:

The program statements are executed sequentially. But, sometimes we need to change the order of execution of statements depending on certain conditions. C language possesses decision making capabilities and supports statements called decision making statements. The decision making statements supported by C are as follows:

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

The if statement:

The if statement is a powerful decision making statement and is used to control the flow of execution of statements. It is basically a two-way decision statement used in conjunction with an expression. The computer evaluates the value of the expression first and depending on the value of the expression it transfers control to a particular statement. The if statement can be implemented in different forms depending on the complexity of the conditions to be tested. The different forms of if statements are:

- Simple if

- If…else statement
- Nested if…else statement
- Elseif ladder

5.1 Simple if statement:

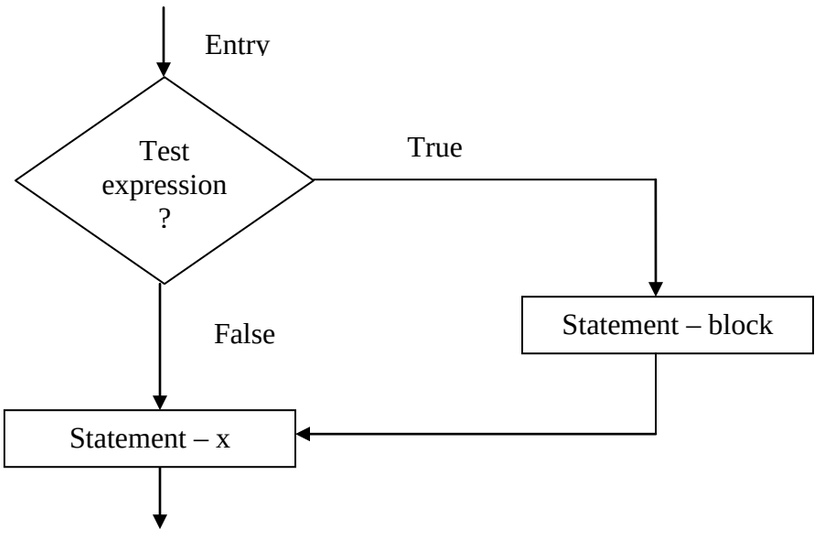
The general form of simple if statement is as follows

```
if(test expression)
{
        statement block;
}
        statement x;
```

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 control jumps to statement x. However, if the condition is true both the statement block as well as statement x are executed in a sequence.

Example:

```
if(category==sports)
 {
    mark=mark+bonus;
 }
  printf("%f " ,mark);
```



5.2 if..else statement:

The if else statement is an extension of the simple if statement. The general format of it is as follows:

```
if(test-expression)
  {
     True-block statements;
  }
else
  {
     False-block statements;
  }
 Statement x;
```

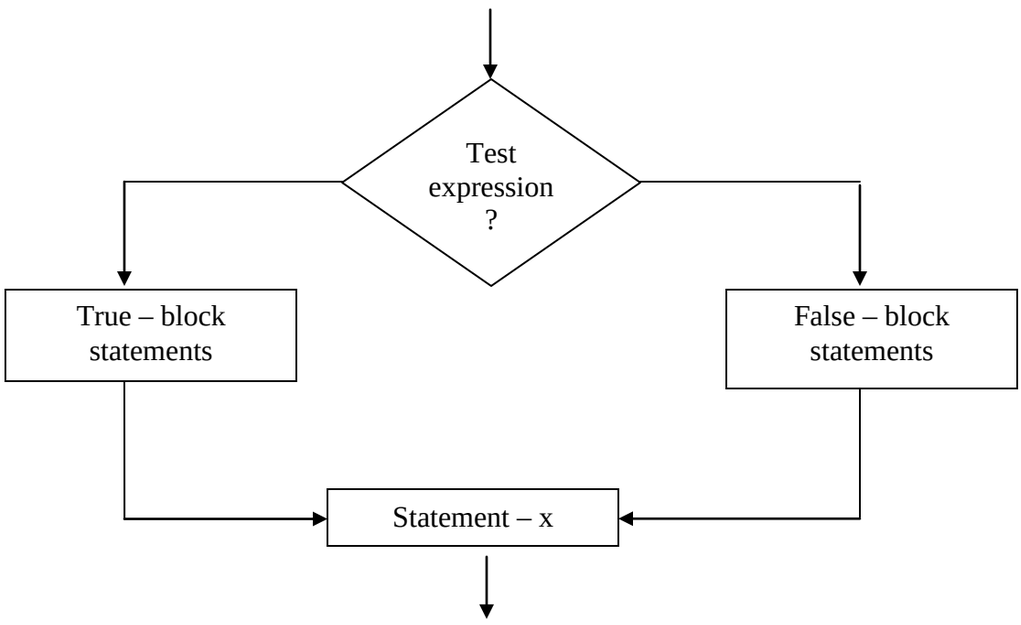
If the test expression is true, then the true block statements are executed otherwise the false block statements are executed. Either of the two statements, i.e. true block or false block statements only are executed, but not both. In both the cases, the control transfers to statement x after executing the block.

Example 1:

```
if(code==1)
  boy=boy+1;
else
  girl=girl+1;
```

Example 2:

```
if(a>b)
printf("a is greater than b");
else
    printf("b is greater than a");
```



Sample Program:

```
#include<stdio.h>
void main()
{
        int n;
        printf("Enter Number");
        scanf("%d",&n);
        if(n%2==0)
         printf("Even Number");
        else
         printf("Odd number");
        getch();
}
```

Output:

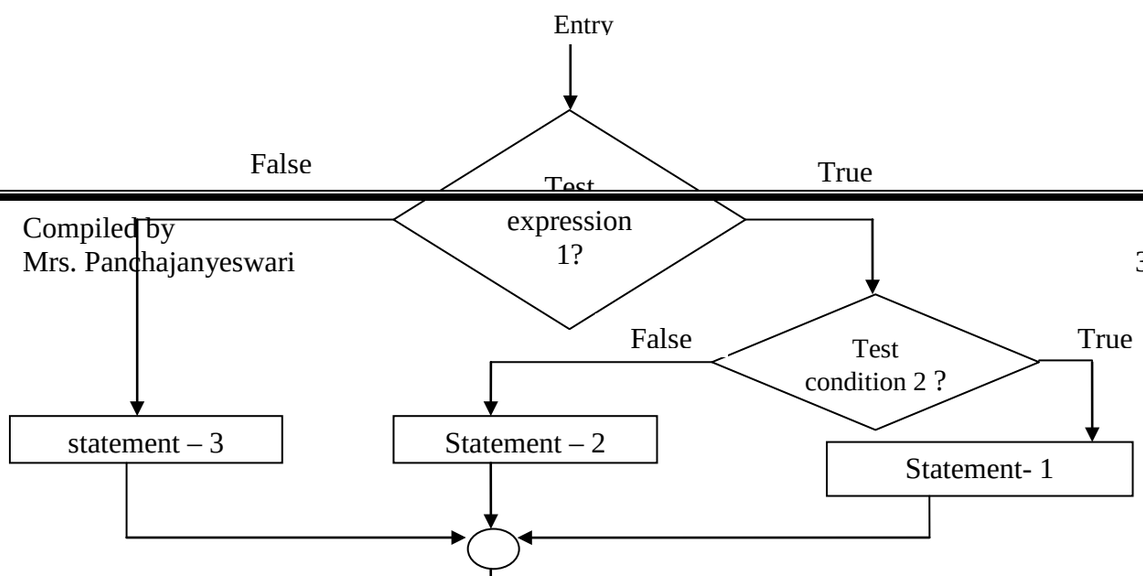
Enter Number 5
Odd number
Enter Number 8
Even Number

5.3 Nested if..else statement:

When a series of decisions are involved, we may have to use more than one if..else statement one within the other. This form of if..else statement is known as nested if..else statement. The general format of this statement is as follows:

```
if(test condition1)
{
   if (test condition2)
      statement 1;
  else
    statement 2;
}
else
   statement 3;
statement x;
```

If condition 1 is initially false, statement 3 will be executed; otherwise it continues to evaluate condition 2. If condition 2 is true then statement 1 is executed; otherwise statement 2 will be executed and then the control transfers to statement x.



Example 1:

```
if(sex==”F”)
{
     if(bal>50000)
    bonus=0.05*bal;
     else
       bonus=0.02*bal;
  }
else
   bonus=0.01*bal;
   bal=bal+bonus;
```

Example 2:

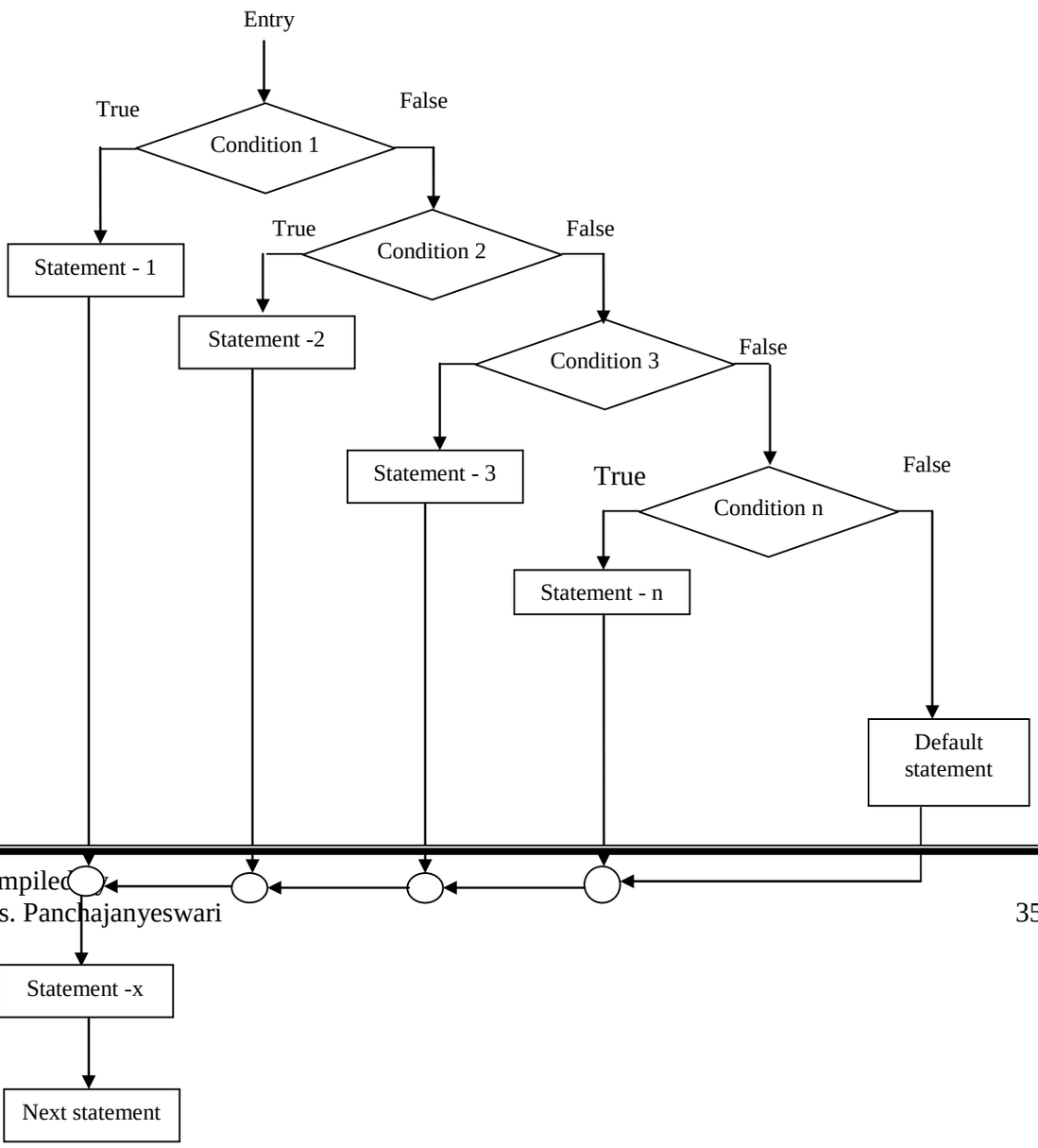
```
if (i > 2)
{
if (i < 4)
{
  printf ("i is three");
}
}
```

Example 3:

```
#include<stdio.h>
void main()
{
        int a,b,c;
        clrscr();
        printf("Enter value for a:");
        scanf("%d",&a);
```

```
        printf("Enter value for b:");
        scanf("%d",&b);
        printf("Enter value for c:");
        scanf("%d",&c);
        if(a>b)
        {
         if(a>c)
          printf("%d is Largest",a);
         else
          printf("%d is largest",c);
        }
        else if(b>c)
          printf("%d is Largest",b);
        else
          printf("%d is Largest",c);
        getch();
}
```

5.4 The Else-if Ladder:



A multi path decision is a chain of if statements in which each else is an if statement. This construct is known as an else-if ladder. The general format of else-if ladder is as follows:

```
  if(condition 1)
     statement 1;
elseif(condition 2)
     statement 2;
elseif(condition 3)
     statement 3;
:
:
  elseif(condition n)
     statement n;
else
   default statement;
statement x;
```

Here the conditions are evaluated from top. As soon as a true condition is encountered, the statements associated with it are executed and the control is transferred to statement x. If all the conditions are false, then the final else containing the default statement will be executed.

Example:

```
  if(marks>79)
    grade= " Distinction";
else if(marks>59)
    grade= " First";
  else if(marks>49)
    grade= " Second";
  else if(marks>39)
    grade= " Third";
  else
    grade= "Fail";
 printf("%s",grade);
```

5.5 The switch statement:

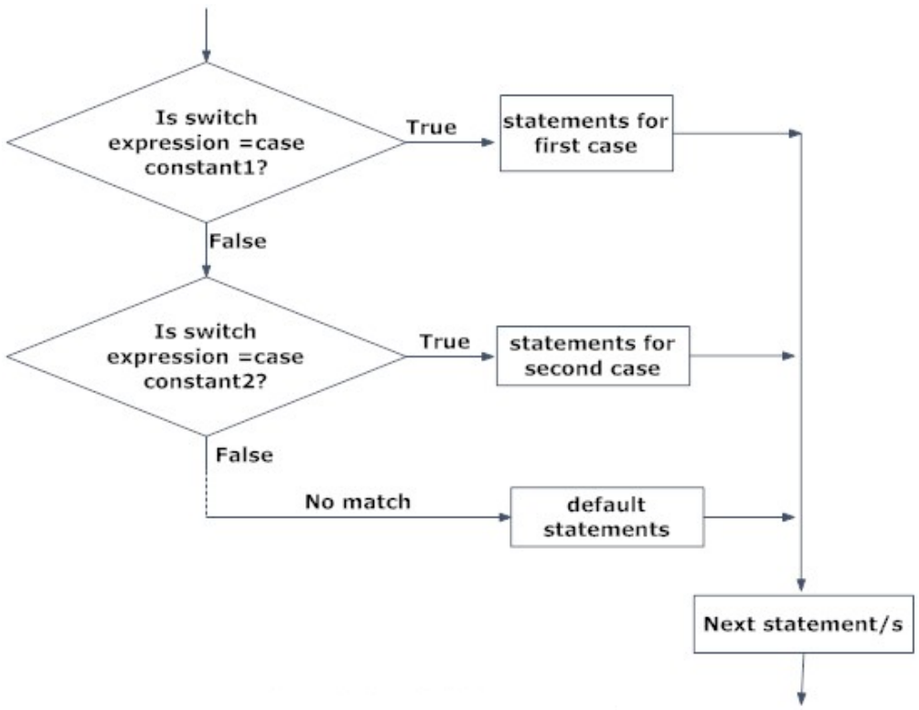
C language supports a built-in multi way decision statement known as switch statement. The switch statement tests the value of a given variable against a list of case values and when a match is found, the block of statements associated with that case are executed. The general form of switch statement is as follows:

```
switch(expression)
{
  case value 1:
                    block 1;
                    break;

  case value 2:
                    block 2;
                    break;
:
:
 case value n:
                    block n;
                    break;

default:
                    block x;
                    break;
}
```

Flowchart for switch statement:



The expression can either be an integer expression or a character expression. Value1, value 2,….value n are constants known as case labels. Each of these values should be unique within a switch statement. Case labels must end with a colon(:). When the switch statement is executed, the value of the expression is successively compared against the values value1, value 2,….value n. If a match is found, the block of statements that follow the case are executed. The break statement in each block signifies the end of that case and causes exit from the switch statement. The default case is an optional one. The statements in this case are executed when no case labels match with the value of that expression.

Example:

```
index=marks/10;
 switch(index)
 {
   case 10:
   case 9:
   case 8:
                grade='A';
                 break;
   case 7:
    case 6:
                grade='B';
                 break;
    case 5:
                grade='C';
```

```
                        break;
            case 4:
                        grade=’D’;
                        break;
            default:
                        grade=’E’;
                        break;
          }
          printf(“%c”,grade);
```

Program:

```
#include<stdio.h>
void main()
{
        char ch;
        printf("Enter a character in CAPITAL:");
        scanf("%c",&ch);
        switch(ch)
        {
          case 'V':printf("Color is Violet");
                   break;
          case 'I':printf("Color is Indigo");
                   break;
          case 'B':printf("Color is Blue");
                   break;
          case 'G':printf("Color is Green");
                   break;
          case 'Y':printf("Color is Yellow");
                   break;
          case 'O':printf("Color is Orange");
                   break;
          case 'R':printf("Color is Red");
                   break;
          default :printf("Color not in Rainbow");
        }
        getch();
}
```

Output:

```
Enter a character in CAPITAL:B
Color is Blue
Enter a character in CAPITAL:E
Color not in Rainbow
```

Statement	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.

5.6 Conditional Operator(?:) :

This operator is also called ternary operator. This operator is used for making two way decisions. It is a combination of ? and : symbol. The general form is as follows:

Condition ? expression 1 : expression 2;

Here, the condition is evaluated first. If the result is non-zero, expression 1 is evaluated and returned as the value of the expression. Otherwise, expression 2 is evaluated and returned
Example;

flag=(x>0)?1:0;

Sample Program:

```
#include<stdio.h>
void main()
{
        int a,b,small;
        clrscr();f
        printf("Enter value for a");
        scanf("%d",&a);
        printf("Enter value for b");
        scanf("%d",&b);
                small=(a<b)?a:b;
        printf("%d is small",small);
        getch();
}
```

Output:
Enter value for a 5
Enter value for b 4
4 is small

5.7 Goto statement:

Goto statements are used to branch unconditionally from one point to another in a program. The goto statement requires a label in order to identify the place where the branch is to be made. A label can be any valid user defined name and must be followed by a : . The label is placed immediately before the statement where the control is to be transferred. The label can appear anywhere in the program either before or after the goto label. The goto statement can be used to transfer control outside the loop when peculiar conditions are encountered. Goto statements are generally avoided in structured programming.

Syntax:

```
goto label;
   ----------
   ----------
label:
        statement ;
```

Assignment:

2 mark Questions:

1. Give the syntax of simple if with example.(2007,2010)
2. What is the purpose of goto statement?(2008)
3. What is ternary operator? Give example.(2006.2008,2009)
4. Give the syntax of switch statement.(2011)

5 mark Questions:

1. Explain the various forms if statement with example.(2006,2007,2009)
2. Explain the working of switch statement with an example.(2005,2008,2011)

CHAPTER 6

DECISION MAKING AND LOOPING

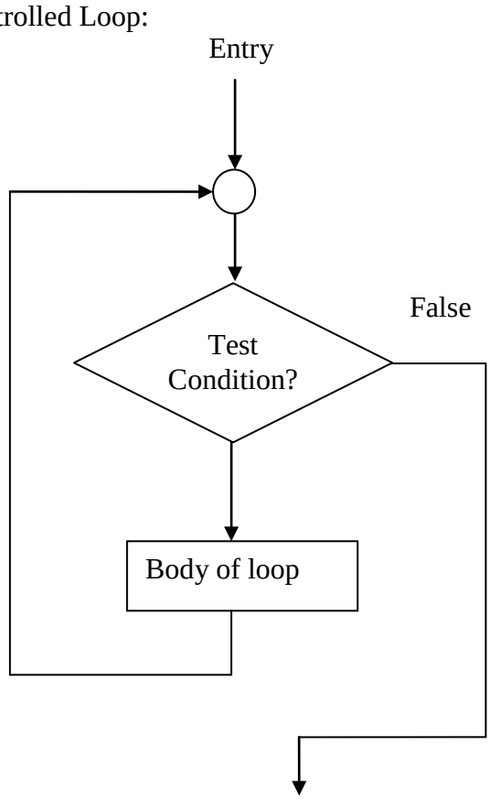
6.1 while statement
6.2 Do-while statement
6.3 For statement
6.4 Break, continue and exit statement
6.5 Jumps in loops
Assignment questions

Introduction

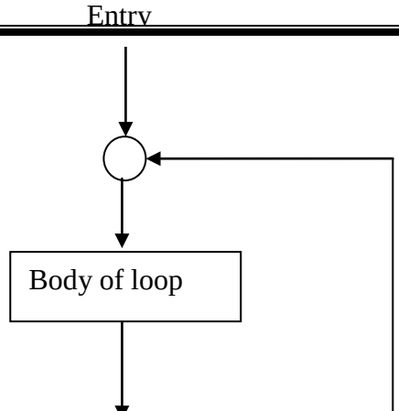
Looping is executing a sequence of statements until some conditions to terminate the loop are satisfied. A program loop consists of 2 segments i.e. body of the loop and control statements. The control statements test certain condition and then direct repeated execution of statements contained in the body of the loop. A looping process includes the following steps

1. Setting and initializing of a counter
2. Execution of statements in a loop
3. Test for specified condition for executing the loop
4. Incrementing the counter

a) Entry Controlled Loop:



b) Exit Controlled Loop:



C language supports 3 loop constructs to perform looping operations. They are as follows

- While statement
- Do…while statement
- For statement

6.1 The while statement:

The while statement is a entry-controlled loop. Here, the test condition is evaluated first and if the condition is true the body of the loop is executed. After executing the statements contained in the body of the loop, the test condition is again evaluated to check if it is true. This process is repeated until the test condition becomes false; and the control is transferred out of the loop. The general form of while statement is

```
while(test condition)
{
   Body of the loop;
}
```

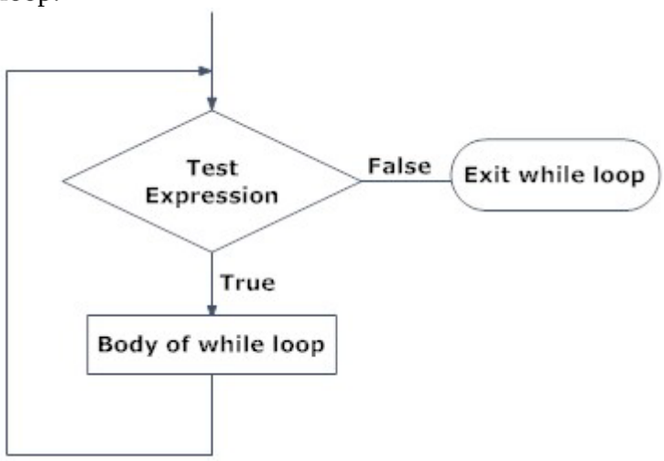
Here, the body of the loop can contain more than one statement. If the condition is initially false, the body of the loop will not be executed at all.

Example:

```
i=1;
while(i<=10)
```

```
        {
          printf(“%d”,i);
        }
```

Flowchart for while loop:

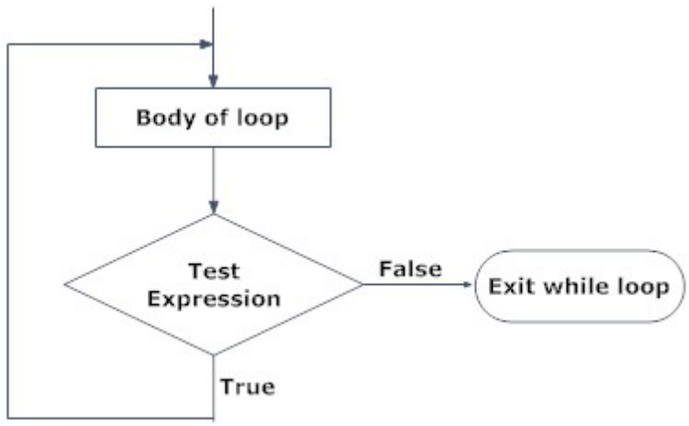



```
/*PROGRAM TO FIND THE SUM OF INDIVIDUAL DIGITS */
#include<stdio.h>
void main()
{
        int n,m,rem,rev,sum;
        clrscr();
        printf("Enter any number:");
        scanf("%d",&n);
        m=n;
        sum=0;
        rev=0;
        while(n!=0)
        {
         rem=n%10;
         sum=sum+rem;
         rev=(rev*10)+rem;
         n=n/10;
        }
        printf("The sum of digits is %d\n",sum);
        getch();
}
```

6.2 The do…while statement:

This loop is an exit controlled loop because the condition is checked after executing the statements in the loop. Here the statements are executed first and then, checks the condition. The body of the loop is executed at least once even if the condition is initially false. On reaching the do statement the body of the loop is executed. The test condition in the while statement is evaluated. If the condition is true, execution of the loop is repeated. This process continues as long as the condition specified in the while statement is true. Once the condition becomes false, the loop is terminated and the control goes to the statement after the while statement.

Flow chart for do..while loop:



The general form of do while statement is as follows:

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

Example:

```
do
{
  printf("%d",i);
i++;
}
while(i<=10);
```

```
/*To find X power of Y using do-while statement*/
#include<stdio.h>
void main()
{
    int x,y,p=1,i;
    printf("Enter value for x:");
    scanf("%d",&x);
    printf("Enter value for y:");
```

```
        scanf("%d",&y);
        i=1;
        do
        {
         p=p*x;
         i++;
        }while(i<=y);
        printf("%d power %d is %d",x,y,p);
        getch();
}
```

6.3 For statement:

This statement is also an entry-controlled loop that provides a concise loop control structure. The general form of for loop is as follows:

```
        for(initialization; test condition; increment/decrement)
        {
         Body of the loop;
        }
```

The execution of this loop is as follows:

- Initialization of the control variable is done using the assignment operator. More than one assignment can be made here, but, they must be separated by a comma operator.
- The value of the control variable is tested in the test condition. The test condition is any relational expression that determines when the loop will be terminated. The body of the loop executes as long as the condition is true.
- After the body of the loop is executed the control is transferred back to the for statement. Now the control variable is either incremented or decremented and its new value is tested to check if it satisfies the loop condition. This process continues till the value of the control variable fails to satisfy the test condition

Example :

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

```
/*Program to find the factorial of a number*/
#include<stdio.h>
void main()
{
        int n,fact=1;
        printf("Enter any number:");
        scanf("%d",&n);
        for(i=n;i>=1;i--)
         fact=fact*i;
        printf("Factorial of %d is %d",n,fact);
        getch();
}
```

Output:
Enter any number: 4
Factorial of 4 is 24

Loop Type	**Description**
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body
nested loops	You can use one or more loop inside any another while, for or do..while loop.

6.4 Break, Continue and Exit statements

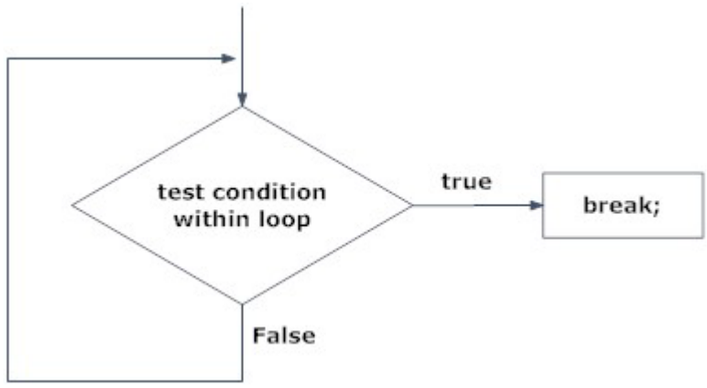
6.4.1 Break statement:

In order to exit out of a loop, we use the break statement. When a break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement following the loop. When a break statement is encountered in a nested loop then, it will only exit from the loop containing the break statement i.e. a break statement is used to exit from a single loop only.

Syntax:

break;

Flowchart for break statement:



6.4.2 Continue statement:

The continue statement causes the loop to be continued with the next iteration after skipping any statements. The continue statement tells the compiler to skip the following statements and continue with the next iteration.

Syntax:

continue;

Whenever the continue statement is encountered, the other statements are ignored by the compiler and the control directly passes to the test condition to execute the next iteration. In case of for loop, the increment section is executed before the test condition is evaluated.

The **difference** between break and continue statements is that the break statement causes the loop to be terminated where as the continue statement continues to execute the loop with the next iteration.

6.4.3 Exit function:

We can jump out of a program using the exit function. Due to some reason, if we wish to break out of a program and return to the operating system, we can use the exit() function. The exit() function takes an integer value as its argument. Normally, zero is used to indicate normal termination and a non-zero is used to indicate termination due to some error or abnormal condition. The use of exit() function requires the inclusion of the header file <stdlib.h>

Control Statement	**Description**
break statement	Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
goto statement	Transfers control to the labeled statement. Though it is not advised

	to use goto statement in your program.

6.5 Jumps in Loops:

Loops perform a set of operations repeatedly until the control variable fails to satisfy a test condition. The number of times a loop executes is decided in advance and the test condition is written to achieve this. Sometimes, when executing a loop it is desirable to skip a part of the loop or leave the loop as soon as certain conditions occur. C permits a jump from one statement to another within the loop as well as a jump out of a loop.

An early exit from a loop can be accomplished using the break statement or the goto statement. These statements can be used within a while, do-while or for loops. When a break statement is encountered inside a loop, the loop is immediately exited and the program continues with a statement immediately following the loop. When the loops are nested, the break statement would only exit from the loop containing it, i.e. break statement will exit only a single loop.

Assignment

2 mark Questions:

1. What is the purpose of break statement in C? (2005, 2007,2008,2011)
2. What is the purpose of continue statement in C? (2006,2009)
3. Give the difference between break and continue statements.(2010)
4. Give the syntax of while loop.
5. Give the syntax of do..while loop.
6. Give the syntax of for loop.

5 mark Questions:

1. Explain the working of while loop with syntax and example (2006,2007,2009)
2. Explain the working of do..while loop with syntax and example(2006,2009)
3. Explain the working of for loop with syntax and example(2005,2008)
4. Write a C program to find the sum of n natural numbers
5. Write a C program to find factorial of a number(2007,2009,2011)
6. Write a C program to find power of a number
7. Write a C program to find the sum of digits of a number(2007,2009,2010)
8. Write a C program to check whether a given number is armstrong or not
9. Write a C program to check whether a given number is prime or not
10. Write a C program to reverse a number (2010,2012)

CHAPTER 7

ARRAYS

Introduction:

Definition: An array is a group of related data items that share a common name. Each element in an array can be accessed by an index number or a subscript written in square brackets.

The complete set of values is referred to as an array and the individual values are called elements. Arrays can be declared as any data type. Arrays can be of 2 types. They are

- One dimensional array
- Multidimensional array

Arrays are most useful when they have a large number of elements: that is, in cases where it would be completely impractical to have a different name for every storage space in the memory. It is then highly beneficial to move over to arrays for storing information for two reasons:

- The storage spaces in arrays have indices. These numbers can often be related to variables in a problem and so there is a logical connection to be made between an array an a program.
- In C, arrays can be initialized very easily indeed. It is far easier to initialize an array than it is to initialize twenty or so variables.

One Dimensional Array:

A list of items can be given one variable name using only one subscript. Such a variable is called a single-subscripted variable or a single dimensional array or a one dimension array. The subscripts of an array can only be integer constants.

7.1 Declaration and Initialization of Arrays:

7.1.1 Declaration of Arrays:

Arrays like any other variable should be declared before they are used. Arrays are declared as follows:

type array-name[size];

Here, type specifies the data type of the elements contained in the array. The size indicates the maximum number of elements that can be stored in an array.

Example:

int age[5];

Here, the name of array is age. The size of array is 5,i.e., there are 5 items(elements) of array age. All element in an array are of the same type (int, in this case).

When a variable is declared as an array the computer reserves the specified number of maximum storage for that variable. C language treats strings as array of characters. The size in character string represents the maximum number of characters it can hold.

Example:

char name[10];

name= “RAMA”

The variable name is stored in the array as follows:

‘R’	‘A’	‘M’	‘A’	‘\0’

7.1.2 Initialization of Arrays:

Arrays can be initialized in the same way as ordinary variables. The general format for initializing array variables is as follows:

static type array-name={list of values};

The values in the list are separated by commas. The size of the array need not be specified. In such cases the compiler allocates enough space for all the initialized elements.

E.g. int mark[5]={34,24,56,27,29}

char name[]={‘R’,‘O’,‘B’,‘E’, ‘R’, ‘T’};

There are 2 limitations when initializing arrays. They are:

- Selected items cannot be initialized
- No short cut to initialize a large array

Example:

int age[5]={2, 4, 34, 3,4}

Initialization of one-dimensional array

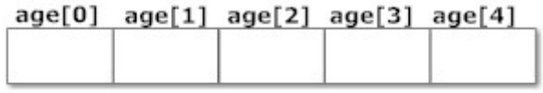
Accessing one dimensional array:

In C programming, arrays can be accessed and treated like variables in C.
For example:

```
scanf("%d",&age[2]);
/* statement to insert value in the third element of array age[]. */

scanf("%d",&age[i]);
/* Statement to insert value in (i+1)th element of array age[]. */
/* Because, the first element of array is age[0], second is age[1], ith is
age[i-1] and (i+1)th is age[i].
/*Statement to print the first element in the array*/
printf("%d",age[0]);
```

Size of array defines the number of elements in an array. Each element of array can be accessed and used by user according to the need of program. For example:



Array elements

Note that, the first element is numbered 0 and so on.
Here, the size of array age is 5 times the size of int because there are 5 elements.
Suppose, the starting addres of age[0] is 2120d and the size of int be 4 bytes. Then, the next address (address of a[1]) will be 2124d, address of a[2] will be 2128d and so on.

7.3 Accessing Two Dimensional Arrays:

In order to store a table of values comprising of rows and columns, C supports 2 dimensional arrays. Two dimensional arrays are declared as follows:

type array-name [row size][column size];

Example:

float numbers[8][7];

Here, row size and column size are some constant. (The sizes of the two dimensions do not have to be the same.) This is called a two dimensional array because it has two indices, or two labels in square brackets. It has (SIZE * SIZE) or size-squared elements in it, which form an imaginary grid, like a chess board, in which every square is a variable or storage area.
Example: If A is a 2x6 array, it can be represented as follows. Here the first number indicates the row and the second represents the column. In C, each is written within square brackets.

	col 1	col 2	col 3	col 4	col 5	col 6
row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]

Figure: Multidimensional Arrays

Each dimension in the array is indexed from 0 to maximum size minus one. The first index is used to select the row and the second index is used to select the column within that row.

Initializing a 2-D Array:

2-D arrays can be initialized in the following ways:

a) static int table[2][3]={0,0,0,1,1,1};

b) static int table[2][3]={{0,0,0},{1,1,1}};

c) static int table[2][3]={
{0,0,0},
{1,1,1}
};

If the values are missing in the initializer, they are automatically set to zero. If all the elements in the array are to be set to zero then the following method is used

static int table[3][5]={{0},{0},{0}};

Multidimensional array:

C allows arrays of 3 or more dimensions. The exact limit on the number of dimensions depends on the compiler. The general form of multidimensional array declaration is

type array-name [s1][s2]….[sm];

where si is the size of the ith dimension.

Example:

int survey[3][5][12];

The above declaration can be used to store the amount of rainfall in past 3 years, in five cities during 12 months.

7.4 Sample Programs in Arrays

7.4.1 C program to implement Linear Search

```
/*PROGRAM TO FIND A NUMBER USING LINEAR SEARCH*/
#include<stdio.h>
void main()
{
        int a[10],n,i,key,flag=0;
        clrscr();
        printf("Enter Range:");
        scanf("%d",&n);
        printf("Enter %d elements\n",n);
        for(i=1;i<=n;i++)
         scanf("%d",&a[i]);
        printf("Enter the element to be searched:");
```

```
        scanf("%d",&key);
        flag=0;
        for(i=1;i<=n;i++)
          if(key==a[i])
          {
           flag=1;
           break;
          }
        if(flag==1)
          printf("Element is found at position %d",i);
        else
         printf("Element not found");
        getch();
}
```

7.4.2 C program to sort an array of numbers in descending order

```
/*PROGRAM TO ARRANGE NUMBERS USING BUBBLE SORT*/
#include<stdio.h>
void main()
{
        int a[20],i,j,n,temp;
        clrscr();
        printf("Enter range");
        scanf("%d",&n);
        printf("Enter the elements:\n");
        for(i=1;i<=n;i++)
         scanf("%d",&a[i]);
        for(i=1;i<n;i++)
         for(j=1;j<=n-i;j++)
          if(a[j]<a[j+1])
          {
            temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
          }
        printf("The sorted elements are:\n");
        for(i=1;i<=n;i++)
         printf("%d\t",a[i]);
        getch();
}
```

7.4.3 C program to add two matrices

```
/*Addition of two matrices*/
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[5][5],b[5][5],c[5][5],m,n,p,q,i,j;
        clrscr();
        printf("Enter the number of rows for 1st matrix\n");
        scanf("%d",&m);
        printf("Enter the number of columns for 1st matrix\n");
        scanf("%d",&n);
        printf("Enter the number of rows for 2nd matrix\n");
        scanf("%d",&p);
        printf("Enter the number of columns for 2nd matrix\n");
        scanf("%d",&q);
        if(m!=p || n!=q)
         printf("Matrix addition not possible");
        else
        {
         printf("Enter the elements of the 1st matrix:");
         for(i=1;i<=m;i++)
          for(j=1;j<=n;j++)
           scanf("%d",&a[i][j]);
         printf("Enter the elements of the 2nd matrix:");
         for(i=1;i<=m;i++)
          for(j=1;j<=n;j++)
           scanf("%d",&b[i][j]);
         for(i=1;i<=m;i++)
          for(j=1;j<=n;j++)
           c[i][j]=a[i][j]+b[i][j];
         printf("The resultant matrix is \n");
         for(i=1;i<=m;i++)
         {
          for(j=1;j<=n;j++)
           printf("%d ",c[i][j]);
          printf("\n");
         }
        }
         getch();
}
```

7.4.4 C Program to multiply two matrices

```
/*Multiplication of two matrices*/
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
        int a[5][5],b[5][5],c[5][5],m,n,p,q,i,j,k;
        clrscr();
        printf("Enter the number of rows for 1st matrix\n");
        scanf("%d",&m);
        printf("Enter the number of columns for 1st matrix\n");
        scanf("%d",&n);
        printf("Enter the number of rows for 2nd matrix\n");
        scanf("%d",&p);
        printf("Enter the number of columns for 2nd matrix\n");
        scanf("%d",&q);
        if(n!=p)
         printf("Matrix Multiplication not possible");
        else
        {
         printf("Enter the elements of the 1st matrix:");
         for(i=1;i<=m;i++)
          for(j=1;j<=n;j++)
           scanf("%d",&a[i][j]);
         printf("Enter the elements of the 2nd matrix:");
         for(j=1;j<=p;j++)
          for(k=1;k<=q;k++)
           scanf("%d",&b[j][k]);

         for(i=1;i<=m;i++)
          for(k=1;k<=q;k++)
          {
           c[i][k]=0;
           for(j=1;j<=n;j++)
           c[i][k]=c[i][k]+(a[i][j]*b[j][k]);
          }
         printf("The resultant matrix is \n");
         for(i=1;i<=m;i++)
         {
          for(k=1;k<=q;k++)
           printf("%d ",c[i][k]);
          printf("\n");
         }
        }
         getch();
}
```

7.4.5 C Program to find the transpose of a given matrix

```
/*To find the transpose of a matix*/
#include<stdio.h>
```

```
#include<conio.h>
void main()
{
        int a[5][5],m,n,i,j;
        clrscr();
        printf("Enter the number of rows\n");
        scanf("%d",&m);
        printf("Enter the number of columns\n");
        scanf("%d",&n);
        printf("Enter the elements of the matrix:");
        for(i=1;i<=m;i++)
         for(j=1;j<=n;j++)
          scanf("%d",&a[i][j]);
        printf("The given matrix is :\n");
        for(i=1;i<=m;i++)
        {
         for(j=1;j<=n;j++)
          printf("%d ",a[i][j]);
         printf("\n");
        }
        printf("The transpose of a matrix is \n");
        for(i=1;i<=n;i++)
        {
         for(j=1;j<=m;j++)
          printf("%d ",a[j][i]);
         printf("\n");
        }
        getch();
}
```

Assignment

2 mark Questions:

1. Define an array.
2. How do you declare a one dimensional array in C? Give example (2009,2010)
3. How do you declare a two dimensional array in C? Give example(2006,2008)
4. How do you initialize a one dimensional array in C? Give example(2007,2010)
5. How do you initialize a two dimensional array in C? Give example(2007)

5 mark Questions:

1. How do you work with one dimensional array in C? Give example(2010)
2. How do you work with two dimensional array in C? Give example(2011)
3. Write a C program to find the largest element in an array.(2012)
4. Write a C program to sort a list of elements in ascending order.(2006,2009,2010)
5. Write a C program to find the sum of diagonal elements in an array

UNIT III

CHAPTER 8

HANDLING OF CHARACTER STRINGS

Introduction

A string is an array of characters. Any group of characters enclosed in double quotes is a string constant.

E.g.: "Hello"

The common operations performed on strings are

1. Reading/Writing
2. Combining strings
3. Copying one string to another
4. Comparing two strings
5. Extracting a portion of strings

8.1 Declaring and initializing strings:

A string variable is any valid C variable name and is always declared as an array. The general form of declaration is

```
char  string-name[size];
```

where size determines the number of characters in the string-name.

When a character string is assigned to a character array, the compiler automatically supplies a null character (‘\0’) at the end of the string. Hence the size should always be equal to the maximum number of characters plus one. C permits a character array to be initialized in either of the following forms

```
static char city[9]= “NEW YORK”;
static char city[9]= {‘N’, ‘E’, ‘W’, ‘ ‘, ‘Y’, ‘O’, ‘R’, ‘K’ ‘\0’};
static char name[]= { ‘G’, ‘O’, ‘O’, ‘D’, ‘\0’};
```

When we initialize a character array by listing elements, a null character must be specified explicitly. C also permits to initialize character strings without specifying the number of elements.

8.2 Reading Strings from Terminals:

Words can be read using the scanf function with %s as the format specifier. But the problem with scanf statement is that it terminates its input when it encounters a white space.

For example,

```
char city[15];
scanf(“%s”, city);
```

If New York is given as input at the terminal then, only New will be assigned to the variable city.

Reading a Line of Text:

It is not possible to read more than one word because scanf terminates as soon as it encounters a white space in the input. Hence, getchar() function is used to repeatedly read successive single characters from the terminal and place them into an array. In this case, reading is terminated when a new line character is entered and a null character is appended at the end of the string.

Example:

```
i=0;
while((ch=getchar())!='\n')
  line[i++]=ch;
line[i]='\0';
```

8.3 Writing Strings to the Screen:

The format %s can be used to display an array of characters that is terminated by a null character.

For example: Consider the following statement

```
printf(“%s”, name);
```

The above statement can be used to display the entire contents of the array, name.

8.3.1 Writing a Line of Text:

putchar() function is used to write a single character on to the screen. This helps us to print a line on the screen. putchar() takes one argument i.e. the character that is to be printed on the screen.

Example:

```
i=0;
do
{
  ch=line[i++];
```

```
    putchar(ch);
}while(ch!='\n');
```

8.4 Arithmetic Operations on Strings:

C allows us to manipulate characters the same way we do with numbers. Whenever a character constant or a character variable is used in an expression, it is automatically converted into an integer value by the system. The integer value depends on the local character set of the system.

For example, if the system uses ASCII representation then,

```
x='a';
printf("%d",x);
```

The above code will display he number 97 on the screen. 97 is the ASCII equivalent on the letter 'a'.

It is also possible to perform arithmetic calculations on character constants and variables. Consider this statement: x='z'-1;

It is a valid C statement. The ASCII code for z is 122. Hence the above statement will assign the value 121 to the variable x.

Character constants can also be used in relational expressions.

For example: This expression (ch>= 'A' && ch<= 'Z') would test if the character contained in ch is an upper case letter.

We can also convert character digit to its equivalent integer value using the following statement.

```
x=ch-digit – '0';
```

Here x is a variable and ch-digit is a character digit.

E.g. Let us assume that ch-digit contains the digit '7', then

```
x= ASCII value of 7 – ASCII value of 0;
 = 55-48 = 7
```

C library also supports a function that converts a string of digits to its equivalent integer value. This is done using the function atoi. The general format of this function is

```
x= atoi(string);
```

where x is an integer variable and string is a character array containing a string of digits.

```
Example:    num= "1947";
            year=atoi(num);
```

The atoi function converts the string "1947" to its numeric equivalent 1947 and assigns it to year.

8.5 Putting Strings together (Concatenation):

The process of combining 2 strings together is known as concatenation. In C, we cannot directly combine two strings. The characters from string1 and string2 should be copied to string3 one after another. But, the size of string3 should be large enough to accommodate both string1 and string2.

8.6 Comparison of two strings:

Strings cannot be compared directly. We have to compare the two strings character by character. The comparison is done till there is a mismatch or one of the strings terminates to a null character, whichever occurs first.

8.7 Built-in String Functions:

C library supports a large number of built-in string functions in a header file called string.h. The most important string handling functions are tabulated as follows:

Function	Purpose
strcat()	Concatenates two strings
strcmp()	Compares two strings
strcpy()	Copies one string to another
strlen()	Finds length of a string

8.7.1 strcat() Function:

This function is used to concatenate or join 2 strings together. It takes the following form

strcat(string1,string2);

where string1 and string2 are character arrays. When this function is executed string2 is appended to string1, It does so by removing the null character at the end of string1 and placing string2 from there. The contents of string2 remain unchanged.

E.g.:

word1:

V	E	R	Y		‘\0’				

word2:

G	O	O	D		‘\0’		

strcat(word1,word2);

The execution of the above statement will result in

word1

V	E	R	Y		G	O	O	D	‘\0’

```
#include<stdio.h>
#include<string.h>
void main()
{
char str1[31]="Sri";str2[32]="nivas";
printf("\nFirst String is %s\n",str1);

printf("Second String is%s\n",str2);
strcat(str1,"")
strcat(str1,str2);
printf("Resultant String %s\n",str1);
```

```
}
```

Output:
First String is Sri
Second String is nivas
Resultant String Sri nivas

8.7.2 strcmp() Function:

This function compares two strings identified by arguments and has value 0 if they are equal. If they are not, it has the numeric difference between the first non-matching characters in the string. It takes the following form

strcmp(string1,string2);

where string1 and string2 may be string variables or string constants.
E.g.: strcmp(“their”, “there”);

The above statement will return a value -9 which is a numeric difference between ASCII “i” and ASCII “r”. If the value is negative, it means that string1 is alphabetically above string2.

Program:

```
/*Use of Strcmp Function*/
#include<stdio.h>
#include<string.h>
void main(void)
{
char str1[31],str2[31];
int value;
printf("\n Enter string 1");
gets(str1);
printf("Enter Second String");
gets(str2);
value=strcmp(str1,str2);
if(value>0)
printf("string %s comes after %s in dictionary order\n"str1,str1);
else if(value<0)
printf("string %s comes before %s in dictionary order\n"str1,str2);
else
printf("Both Strings are same\n");
}
```

Output:
Enter string 1 Mangalore
Enter Second String Bangalore
Mangalore comes after Bangalore in dictionary order

8.7.3 strcpy() Function:

This function works almost like a string-assignment operator. The general format of this function is

strcpy(string1,string2);

The above statement assigns the content of string2 to string1. String2 may be character array or a string constant.

E.g.:

strcpy(city, “DELHI”);

This statement will assign the string “DELHI” to string variable city.

If city1= “DELHI” and city2= “CHENNAI”. The statement strcpy(city1,city2); will assign the contents of string variable city2 to city1. The size of array city1 should be large enough to contain the contents of city2.

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[31]="Sri";str1[32]="nivas";
printf("\nFirst String is %s\n",str1);

printf("Second String is%s=\n",str2);
strcpy(str1,str2);
printf("Resultant String %s\n",str1);
}
```

Output:

```
First String is Sri
Second String is nivas
Resultant String nivas
```

8.7.4 strlen() Function:

This function counts and returns the number of characters in the specified string. The general format of this function is

n=strlen(string);

where n is an integer variable which receives the value of the length of the string. The argument in this function can be a string constant also. The counting ends when the first null character is en countered.

EXAMPLE :

n=strlen(“MANGALORE”);

The above statement counts the number of characters in the given string constant and is assigned to n i.e. 9 is assigned to n.

city= “DELHI”;

strlen(city);

The above statement assigns 5 to the variable m.

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[31];
int len;
printf("\nEnter any String");
gets(str);
len=strlen(str);
printf("\nNumber of Character in%s=%d\n",str,len);
}
```

Output:
Enter any String
Jahnavi
Number of Character in Jahnavi=7

8.8 Table of Strings:

A list of names in a class can be treated as a table of strings and a two dimensional character array can be used to store the entire list. For example, a character array student[30][15] can be used to store a list of 30 names, each of length not more than 15 characters.

Consider the following character array declaration:

```
static char city[][]
                    {
                        "Chennai",
                        "Madgaon",
                        "Mysore",
                        "Bombay",
                        "Kolkatta"
                    };
```

The above declaration is stored as follows:

C	h	e	n	n	a	i		
M	a	d	g	a	o	n		
M	y	s	o	r	e			
B	o	m	b	a	y			
K	o	l	k	a	t	t	a	

In order to access the ith city in the list, we write city[i-1]. Hence city[0] denotes "Chennai", city[1] denotes "Madgaon" and so on. This shows that once an array is declared as a two-dimensional, it can be used like a one dimensional array for further manipulations. The table can be treated as a column of strings.

Assignment
2 mark Questions:

1. What is the purpose of atoi function in C? (2006,2008,2009)
2. How do you declare strings in C? (2007)
3. How do you initialize strings in C?
4. How do you read a line of text in C? (2010)
5. How do you print a line of text in C?(2007)

5 mark Questions:

1. Explain the built in string functions with syntax and example. (2006,2008,2009,2010)
2. Write a C program to reverse a given string without using built-in functions(2008)
3. Write a C program to find the length of a given string without using built-in functions(2009)
4. Write a C program to concatenate two strings without using built-in functions(2007)
5. Write a C program to compare two strings without using built-in functions(2006)
6. Write a C program to count the number of words in a given string(2013)

CHAPTER 9

USER DEFINED FUNCTIONS

9.1 Need for user defined functions
9.2 Declaring functions
9.3 Defining and calling functions
9.4 Function return values and their types
9.5 Categories of functions
9.6 Recursion
9.7 Function with arrays
9.8 Scope, visibility and lifetime of variables
Assignment questions

Introduction:

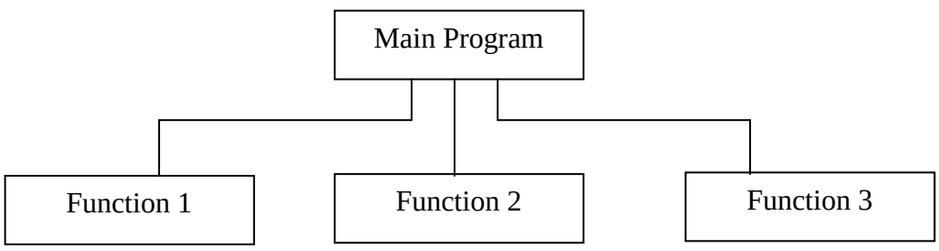
A function is a set of statements that perform a particular task. C functions can be classified into two categories, namely, library functions and user-defined functions. Library functions are those that are predefined and are already present in the various libraries. The examples for library functions are printf, scanf, strlen,strcpy etc. User defined functions are written by the user depending on the need of the application. **main** is an example of user defined function.

It is always possible to write code for any program using a single main function. In this case, the program becomes too large and complex and as a result the task of debugging and maintenance becomes difficult. If a program is divided into functional units, then each part can be independently coded and later combined into a single unit. These subprograms are called 'functions' are easier to understand, debug and test.

9.1 Need for User Defined Functions:

- It facilitates top-down modular programming. In this approach, the high level logic of the overall problem is solved first while the details of each lower level function is addressed later.
- The length of the source program can be reduced by using functions at appropriate places.
- It becomes easy to isolate and locate errors i.e. debugging becomes easier.
- Saves time and memory space.
- A function once written can be used by other programs.

Top-down modular programming



A Multi-Function Program:

A function is a self contained block of code that performs a particular task. The inner details of operation performed by the function are invisible to the rest of the program. All the program knows is what goes in and what comes out. Every C program can be written as a set of such functions.

Consider the following example:

```
printline()
{
   i=1;
   for(i=1;i<=30;i++)
     printf("_");
    printf("\n");
}
```

This function will print a line of 30 character length. This function can be used in the program as follows:

```
main()
{
  printline();
```

```
    printf("This illustrates C functions\n" );
    printline();
}
printline()
{
    i=1;
    for(i=1;i<=30;i++)
      printf("_");
     printf("\n");
}
```

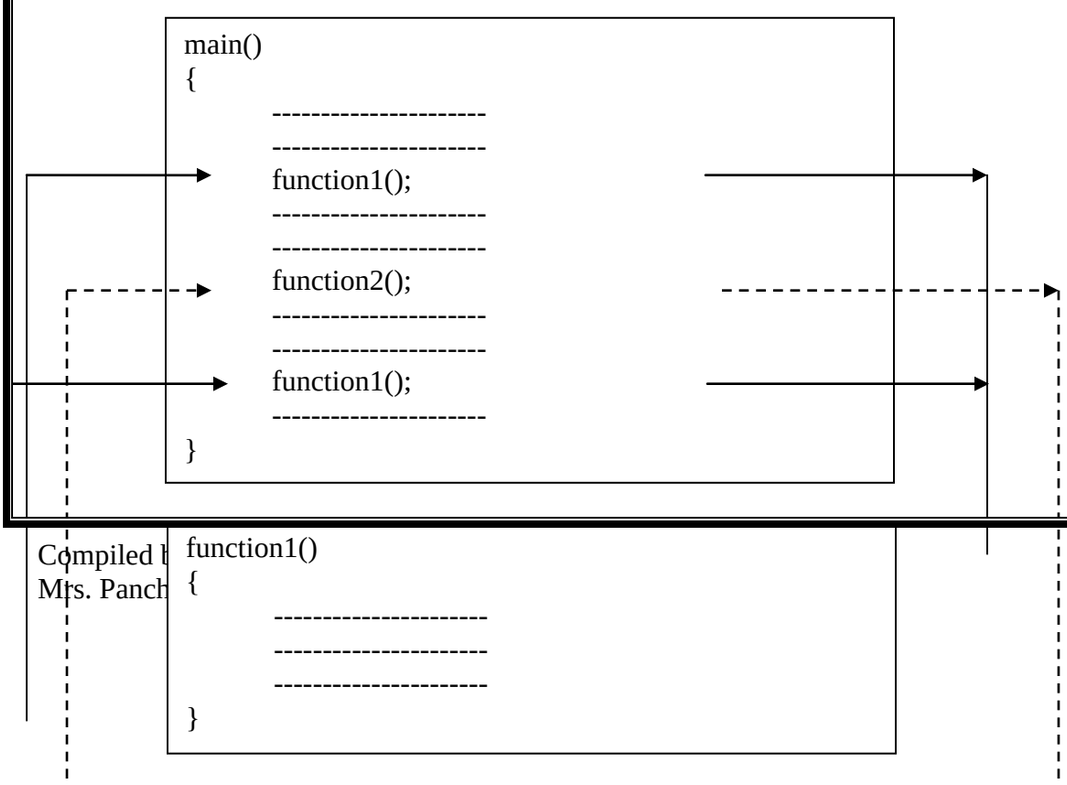
The above program will print the following output:

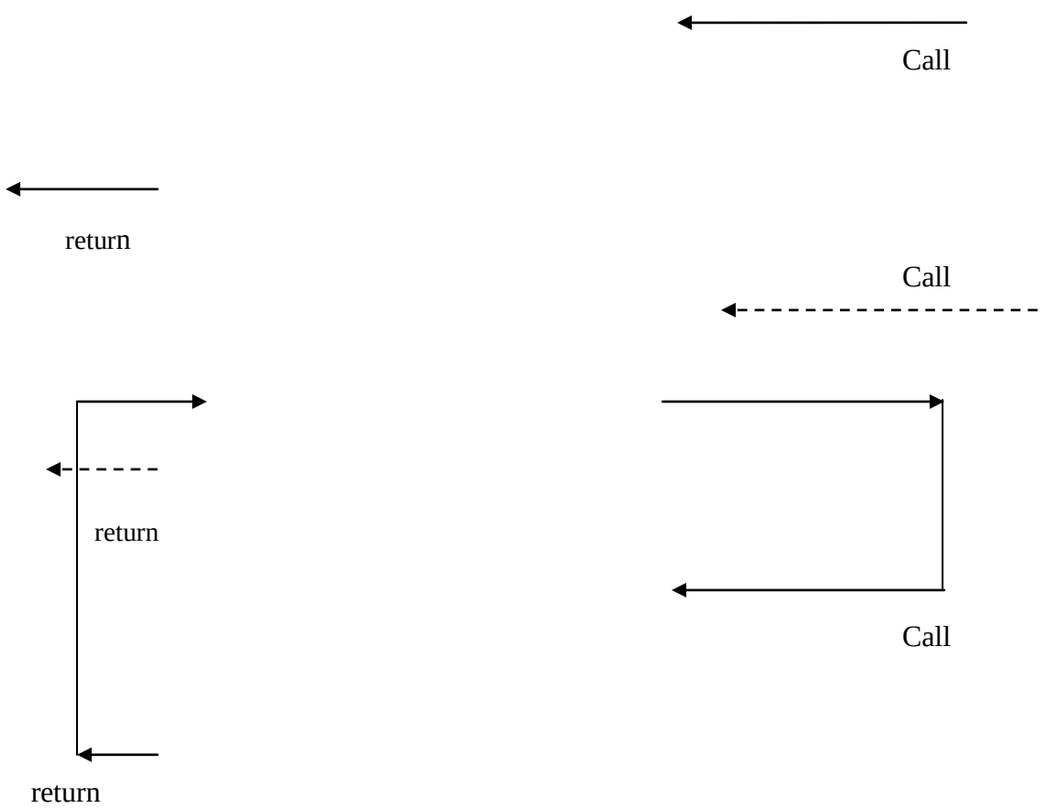
```
_____________________
This illustrates C functions
_____________________
```

The program execution always begins at main(). Here both main and printline are 2 user defined functions. At the very first statement, printline function is encountered. So, the control transfers to the function printline. After the printline function is executed the control is transferred back to the very next statement after the calling function in main. Now the execution continues at the point where the function call was executed.

Any function can call any other function. A function can also call itself. A called function can also call another function. A function can be called more than once. A called function can be placed either before or after the calling function.

Flow of control in a multi function program:





9.3 Declaring and calling Functions:

9.3.1 Declaring a C Function:

All functions in C have the following form:

```
function-name(argument list)
argument declaration;
{
  Local variable declarations;
  Executable statements;
        :
        :
  return(expression);
}
```

Some of the above elements are optional. The argument list and its associated argument declaration are optional. The declaration of local variables is required only when any local variables are used in the function. A function can have any number of executable statements.

Sometimes a function can have no executable statements at all. The return statement is the mechanism for returning a value to the calling function. This is also optional. Its absence indicates that no value is being returned to the calling function.

Function name:

A function name must follow the same rules as the formation as any other variable names. But care must be taken not to duplicate library routine names or any other system commands.

Argument List:

The argument list contains valid variable names separated by commas. The list must be surrounded by parentheses. No semicolon follows the closing parentheses. The argument variables receive values from the calling function, thus providing a means of data communication from the calling function to the called function. All argument variables must be declared for their types after the function header and before the opening brace of the function body.

Example:

```
power(x,n)
float x;
int n;
{
  -------
  -------
}
```

9.3.2 Calling a Function:

A function can be called by simply using the name of the function in the statement.

Example:

```
main()
{
  int p;
  p=mul(10,5);
  printf("%d",p);
}
```

When the compiler encounters a function call, the control is transferred to the function mul(x,y). This function is executed line by line and the value is returned when the return statement is encountered. This value is assigned to p.

When a function call is made, the statement is ended by a semicolon. A function that returns a value can be used in expression like any other variable. A function that doesn't return any value cannot be used in expressions. However a function cannot be used in the right side of an assignment statement.

9.4 Function return values and their types:

A function may or may not send back a value to the calling function. If it does then it is done through the return statement. We can pass any number of values to the called function; the called function can return only return one value per call. The return statement can take the following form:

```
return;
   Or
return(expression);
```

In the first form, the function does not return any value. It only acts as a closing brace for that function. When a return is encountered, the control immediately passed to the calling function.

Example:
```
if(error)
   return;
```

The second form of return with an expression returns the value of the expression.

Example:
```
mul(x,y)
int x.y;
{
  int p;
  p=x*y;
  return(p);
}
```

In the above example, the value of p i.e. the product of two variables x ad y is returned to the calling function. The last two statements can be combined into one statement as follows:

```
return(x*y);
```

A function can have more than one return statement that will return a value based on certain conditions.

Example:
```
if(x<=0)
  return(0);
else
  return(1);
```

9.5 Categories of Functions:

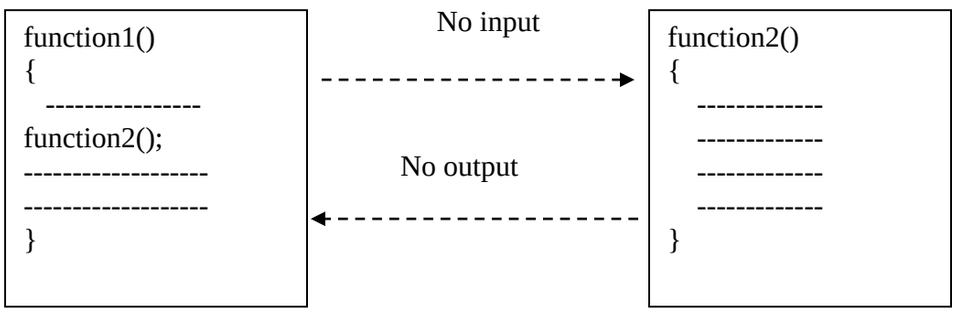
A function depending on whether arguments are present or not and whether a value is returned or not may belong to one of the following categories:

1. Functions with no arguments and no return values
2. Functions with arguments and no return values
3. Functions with arguments and return values

9.5.1 Functions with no arguments and no return values:

When a function has no arguments, it does not receive any data from the calling function. Similarly when there are no return values, the calling function doesn't receive any data from the

called function. There is no data transfer between the calling function and the called function. There is only transfer of control between the two functions.
No data communication between the functions.

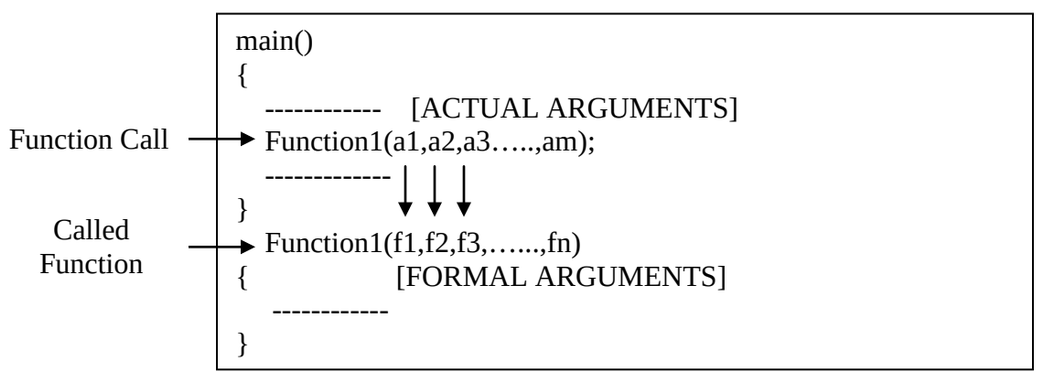


9.5.2 Functions with arguments but no return values:

Here the calling function will pass data to the called function with the help of arguments. Arguments are of two types:

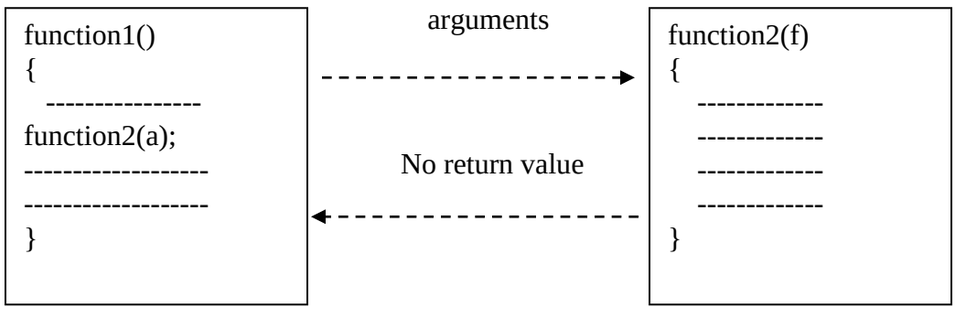
- Actual arguments : These are arguments that are listed in the function call.
- Formal Arguments: These are arguments that are listed in the function definition.

The actual and formal arguments should match in number type and order. The values of the actual arguments are assigned to formal arguments on a one to one basis, starting with the first argument. Whenever a function call is made only a copy of the values of actual arguments is passed into the called function. In case, the actual arguments are more than the formal arguments, the extra ones are discarded. On the other hand, if the actual arguments are less than the formal arguments then, the unmatched formal arguments are initialized to some garbage values. Hence, care should be taken to ensure that the arguments match in number and type.



The calling function will validate the data and then send it to the called function. But the called function will execute all the statements and will not return a value to the called function. There is only one way communication between the calling function and the called function.

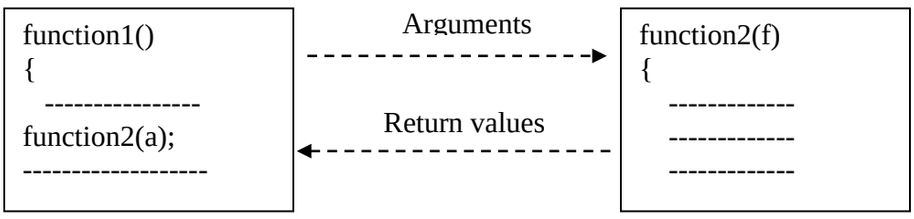
One way communication between the functions:



9.5.3 Functions with arguments with Return values:

Here there is two way communication between the calling function and the called function. The calling function sends data as a set of arguments to the called function. The called function in turn returns a value with the help of return statement.

Two way communication between functions:



Handling non-integer functions:

C functions by default return a value of type int when no type is specified explicitly. In order to explicitly specify what value it should return we should use a type specifier in the function header. The type specifier tells the compiler what type of data the function returns. The called function must also be declared at the start of the calling function like any other variable. The general format of function definition is:

```
type-specifier function-name(argument list)
{
   Function statements;
}
```

Functions returning nothing:

We can use functions that do not return any value. They simply execute a set of statements. Such functions need not be declared in main. Even if they are declared, they are declared with a qualifier void. This states explicitly that the function does not return any value. The general format of this declaration is as follows:

void function-name();

Nesting of Functions:

C language supports nesting of functions. Calling of another function within another function is called as nesting of functions.

9.6 Recursion:

When a function calls itself repeatedly until a terminating condition is satisfied, then the concept is called recursion. Recursive functions can be effectively used to solve problems where the solution is expressed in terms of successively applying the same solution to the subset of the problem. When a recursive function is written, it must be ensured that terminating condition is present. Otherwise, the process is repeated indefinitely.

Example:

```
factorial(n)
int n;
{
 int fact;
 if(n==1)
  return(1);
 else
  fact=n*factorial(n-1);
 return(fact);
}
```

Assume that n=3

fact=n*factorial(n-1);

Since n is not equal to 1 the statements in the else part are executed.

fact=3*factorial(2);

This process continues until n becomes 1.

fact=3*2*1=6

9.7 Functions with arrays:

It is possible to pass the value of an array to a function just like the case of simple variables. To pass an array to the function, it is sufficient to list the name of the array, without any subscripts, and the size of the array as arguments.

Sample Program:

```
/*To Find the largest of n numbers*/
#include<stdio.h>
void main()
{
 int largest(int a[],int n);
```

```
  int a[10],n,i;
  clrscr();
  printf("Enter Range\n");
  scanf("%d",&n);
  printf("Enter %d numbers\n",n);
  for(i=1;i<=n;i++)
   scanf("%d",&a[i]);
  printf("The largest number is %d",largest(a,n));
}

int largest(a,n)
int a[],n;
{
 int i,big;
 big=a[1];
 for(i=2;i<=n;i++)
  if(big<a[i])
   big=a[i];
 return(big);
}
```

OUTPUT:
Enter Range
6
Enter 6 numbers
4
3
6
8
7
9
The largest number is 9

9.8 Storage Classes in C:

The lifetime and scope of a C variable depends on the storage class it may assume. The **scope** of a variable determines over what parts of the program a variable is actually available for use. **Longevity** refers to the period during which a variable can retain a given value during the execution of a program. C supports four storage classes. They are

- Automatic variables
- External variables
- Static variables
- Register variables

9.8.1 Automatic variables:

Automatic variables are declared inside a function in which they are used. They are created when the function is called and destroyed automatically when the function is exited, and hence the name. Automatic variables are therefore local to the function. They are also referred to as local or internal variables. Any variable declared inside a function is by default an automatic variable.

One important feature of these variables is that their value cannot be changed by other functions. This assures that the same variable can be used in different functions within the same program. Although the automatic variables are active only the function that declares it, it remains live throughout the program,

The declaration of automatic variables can take any one of the following forms:

```
main()
{
  int num;
  …….
  …….
}
(Or)
main()
{
  auto int num;
  …….
  …….
}
```

9.8.2 External Variables:

Variables that are both alive active throughout the entire program are known as external variables. They are also called global variables. These variables can be accessed by any function in the program. They are declared before the function definition. Once a variable is declared as global, any function can use it and change its value. Then, the subsequent functions can refer to the new value of the variable. Hence, only variables that are shared between two functions only need to be declared as global. A variable that is declared as a global variable should be preceded by a keyword ‘extern’. The declaration of this variable can be done in either of the two methods.

```
int number;
float len=7.5;
function1()
{
 ------
 ------
}
```

(or)

```
function()
{
  extern int number;
  extern float len=7.5;
  -----
  -----
}
```

Global variable is visible only from the point of declaration to the end of the program.

Example:

```
main()
{
  y=5;
  -------
  -------
}
int y;
function1()
{
   y=y+1;
}
```

In the above example, the value of y is not declared in function main. Hence the compiler issues an error message. Here the value of y after the execution of function1 will be only 1 as the undefined global variable will be initialized to zero by default.

9.8.3 External Declaration:

The above problem can be solved by declaring the variable as **extern**. The above program can be modified as follows:

```
main()
{
  extern int y=5; /*external declaration*/
  -------
  -------
}

function1()
{
  extern int y; /*external declaration*/
  y=y+1;
}
 int  y;/*declaration*/
```

Here although the variable y is encountered before is definition, the external declaration informs that the variable has been defined else where in the program. This declaration will not allocate any memory for the variable.

9.8.4 Static Variables:

A static variable is declared using the keyword static. A static variable may be an internal or external type of variable depending on the place of declaration. Internal static variables are those that are declared inside the function. The scope of internal static variables extends up to the end of the function in which they are defined. Hence, internal static variables are similar to auto variables, except that they remain alive throughout the program. Internal static variables can be used to retain values between function calls. A static variable is initialized only once, when they are compiled.

An external static variable is declared outside of all functions and is available to all the functions in the program. The difference between static external variable and a simple external variable is that the static external variable is available only within the file where it is defined while simple extern variable can be accessed by other files.

9.8.5 Register Variables:

We can tell the compiler to store a variable in one of the machine's registers, instead of in the memory. Accessing variables that are stored in register are faster than storing them in memory. The declaration of such variables is done with the help of a keyword 'register' as follows

register int count;

Only a few variables can be stored as register variables. C will automatically convert register variables into non-register variables once the limit is reached.

2 mark Questions:

1. Define a user defined function.
2. Define recursion.(2006,2008,2009)
3. What is actual parameter? Give example
4. What is formal parameter? Give example

5 mark Questions:

1. Explain the concept of user defined functions with an example(2007,2008,2010)
2. Explain the structure of user defined function in C(2007)
3. Explain the storage classes in C(2005,2008,2010)
4. Write a recursive program to find the factorial of a given number.(2006,2009)
5. Write a recursive program to find the power of a number (x^y) (2007,2010)
6. Write a function in C to find the smallest of n numbers

UNIT IV

CHAPTER 10

STRUCTURES AND UNIONS

10.1 Structure definition
10.2 Giving values to members
10.3 Structure initialization
10.4 Comparison of structure variables
10.5 Array of structures
10.6 Arrays within structures

C supports a constructed data type known as structure, which is a method of packaging data of different types. A structure is a convenient tool for handling a group of items of different data types. Structures help to organize complex data in a more meaningful way. For e.g. it can be used to represent a set of attributes, such as student name, roll number and marks.

10.1 Structure Definition:

A structure definition creates a format that may be used to declare structure variables. The general format of structure definition is

```
struct tag-name
{
        Data type member1;
        Data type member2;
        …….
};
```

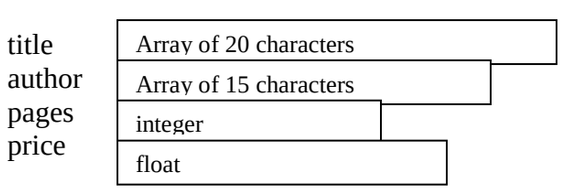
The keyword struct declares a structure to hold the details of the member fields. These fields are also known as structure elements or members. Each member may belong to a different data types. The name of the structure is also known as structure tag. The tag name can be used to declare variables that have the tag's structure.

Example:

```
struct book_bank
{
        char title[20];
        char author[15];
        int pages;
        float price;
};
```

The above format describes only a format called a template to represent information

struct book_bank



We can declare structure variables using the tag names as follows:

struct book_bank book1, book2,book3;
Here book1, book2, book3 are variables of type struct book_bank

10.2 Giving values to members:

The link between a member and a structure variable is established using the member operator or dot operator or period operator(.).
For e.g. book1.price is a variable representing the price of book1 and can be used like any other variables

Book1.price=120.75;

10.3 Structure Initialization:

A structure variable can be initialized like any other variable name. A keyword static is used to initialize it inside a function
E.g.:

```
struct st-record
{
   int weight;
    float height;
} student1={45,  170.5};
main()
{
  Static struct st-record student2={50,  150.5};
 --------
 --------
}
```

10.4 Comparison of structure variables:

Two variables of the same structure type can be compared in the same way as ordinary variables

10.5 Array of structures:

When a structure is to be applied to a group of people or items, arrays of structures can be used.

Example:

```
struct stud
{
        char name[20];
        int reg-no;
        int mark1,mark2,mark3;
        int total;
        float avg;
```

```
};
struct stud student[10];
```

Here student is an array of 10 elements of type stud.

10.6 Arrays within structures:

C permits the use of arrays within structures. A single or a multidimensional array can be used inside the structure

Example:

```
struct stud
{
        char name[20];
        int reg-no;
        int mark[3];
        int total;
        float avg;
};
struct stud student[10];
```

10.7 Structures within structures:

Nesting of structures means a structure within a structure. Nesting of structures is permitted in C

Example;

```
struct employee
{
        int empno;
        char name[20];
        char dept[20];
        struct
        {
          int da;
          int hra;
          int ta;
        } allowance;
} employee;
```

10.8 Structures and Functions:

Structures are usually passed to function by sending a copy of the structure to the function during a function call.

The general format of a function call is

```
function-name(struct-var-name);
```

The called function takes the following form

```
data-type function-name(st-name)
struct st-name;
{
--------
```

```
--------
}
```

10.9 Unions:

Unions follow the same syntax as structures. The major distinction between them is in terms of storage. In structures each member has its own storage location, whereas in a union all the members use the same storage location. This implies that although a union can contain many members of different data types it can handle only one member at a time. Unions are declared as follows

```
union item
{
  int m;
  float x;
  char c;
} code;
```

10.10 Size of structures

We can use the unary operator **sizeof** to tell the size of a structure variable. The expression sizeof(struct x) will evaluate the number of bytes required to hold all the members of the structure x.

10.11 Bit fields

A bit field is a set of adjacent bits whose size can be from 1 to 16 bits in length. A word can be divided into a number of bit fields. The name and size of bit fields can be defined using a structure. The general form of bit field definition is as follows

```
struct tag-name
{
        data-type name1 : bit length;
        data-type name1 : bit length;
        :
        data-type nameN : bit length;
}
```

Example:

```
struct person
{
        unsigned  gender: 1;
        unsigned  age: 7;
        unsigned  m_status: 1;
        unsigned  children: 3;
}emp;
```

The above example defines a variable named emp with four fields. The range of values each field could have is as follows:

Bit field	Bit length	Range of values

gender	1	0 or1
Age	7	0-127(2^7-1)
m_status	1	0 or 1
Children	3	0-7(2^3-1)

Assignment

2 mark Questions:

1. Define structure.
2. Define union.
3. Differentiate between structure and union (2007,2008,2009)
4. What is the purpose of bit fields in C? (2012)
5. How do you declare a structure in C? Give example.(2005,2006,2009)
6. How do you declare a union in C? Give example.(2008,2010)
7. How do you define bit fields in C? Give example.(2012)

5 mark Questions:

1. Explain the concept of defining structures in C with examples.
2. How do you work with array of structures in C? Give examples. (2006)
3. Explain the concept of structures within structures with examples. (2007)

CHAPTER 11

POINTERS

Introduction:

Pointers are an important concept in C. There are a number of advantages of using pointers. They are

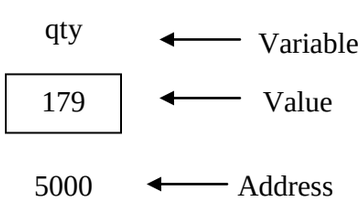
- It enables us to access a variable that is defined outside a function.
- More efficient in handling arrays
- Reduce the length and complexity of a program
- Increase the execution speed

When a variable is declared, the system allocates memory location suitable to hold the variable value. The memory location is called the address.

Consider the following statement:

```
int qty;
qty=179;
```

Representation of a variable in memory



Definition: A pointer is a variable that represents the location of a data item such as variable or array element.

11.3 Declaring and initializing a pointer:

The declaration of a pointer variable takes the following form:

```
data type *pt-name;
```

This informs the compiler that

- The asterisk(*) tells that pt-name is a pointer variable
- pt-name needs memory location

- pt-name points to a variable of type data type

Example:

```
int  *p;
```

Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement as follows:

```
p=&quantity;
```

The above statement causes p to point to quantity. p now contains the address of quantity. This is known as pointer initialization

11.4 Accessing a variable through pointers:

Once a pointer is assigned the address of a variable, the value of the variable can be accessed using a unary operator (*) asterisk. It is also known as indirection operator.
Consider the following statements:

```
int qty,*p,n;
qty=150;
p=&qty;
n=*p;
```

Here n gets the value 150

11.5 Pointers expressions:

Pointer variables can be used in expressions just like ordinary variables. All the arithmetic operators available in C can be applied to them.

11.6 Pointers and scale factors:

Pointers can be incremented like p1=p2+1; p2=p1+2;. Even expressions like p1++; will cause the pointer p1 to point to the next value of its type. For example, if p1 is an integer pointer with an initial value, say 2800, then after the operation p1=p1+1; the value of p1 will be 2802. Hence, when we increment a pointer, its value is increased by the length of data type that it points to. This length is called scale factor.

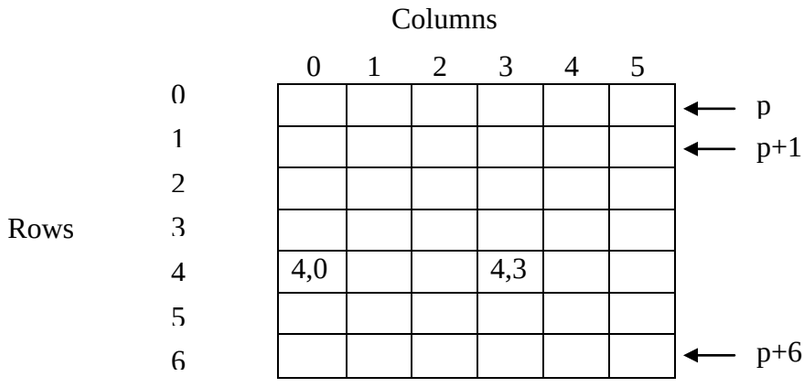
11.7 Pointers and arrays:

When an array is declared, the compiler allocates the base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations. The base address is the location of the first element in the array.
Example:

```
int a[10];
int *p;
p=a;
```

The above statement p=a makes p to point to the array a. i.e. p will point to the base address of a.
The address of an element in an array is calculated as follows:
Address of ith element= base address + (i x scale factor)
where, scale factor is the amount of memory required to store a single value of a data type.
The values in an array can be accessed as *(p+i)
Two dimensional arrays contain two subscripts rather than a single subscripts.

Pointers to 2D arrays



In the above figure,

- p points to the first row.
- (p+i) is a pointer to the i^{th} row.
- *(p+i) is a pointer to the first element in the i^{th} row.
- *(p+i)+j is a pointer to the j^{th} element in the i^{th} row.
- *(*(p+i)+j) is a value stored in the i^{th} row and j^{th} column

11.8 Pointers as function parameters:

Parameters can be passed to functions in two ways. They are

1. Call by value
2. Call by reference

11.8.1 Call by value

When the actual value of the variable is passed as an argument to a function then it is known as pass by value. The value of the variable is copied from the actual parameter to the formal parameter. The function which is called by values does not change the value of the variable in the function call.

Example:

```
main()
{
        int x;
        x=20;
        change(x);
        printf("%d",x);
}
change(p)
int p;
```

```
{
        p=p+10;
}
```

When this code is executed, the value 20 is passed as a function argument, the output is 20. The value is only copied when it goes to the function change. The change in the function does not affect the function main. Hence the output is only 20.

11.8.2 Call by reference:

When we pass address of a variable to a function, the parameters receiving the addresses should be pointers. The process of calling a function using pointers to pass addresses of variables is known as call by reference. The function which is called by reference can change the value of the variable in the function call.

Example:

```
main()
{
        int x;
        x=20;
        change(&x);
        printf("%d",x);
}
change(p)
int *p;
{
        *p=*p+10;
}
```

When this code is executed, the address of x is passed as a function argument. p is a pointer that contains the address of the variable x. Inside the function, 10 is added to the value at address p. Hence the output is only 30.

Assignment

2 mark Questions:

1. Define a pointer variable. (2006,2007,2009)
2. How do you declare pointers in C? Give example (2008)
3. How can you initialize pointers? Give example. (2009,2012)
4. How do you define array as a pointer? Give example(2006,2008,2010)
5. What do you mean by call by value? (2007,2009)
6. What do you mean by call by reference?(2008,2010)

5 mark Questions:

1. Explain the concept of working with pointers in C. (2005,2010)
2. Explain how pointers can be used in a one dimensional array with examples.
3. Explain how pointers can be used in a two dimensional array with examples.
4. Explain call by value with example.(2007,2008,2010)
5. Explain call by reference with example.(2006,2009)

CHAPTER 12

THE PREPROCESSOR

Introduction:

The preprocessor is a program that processes source code before it passes through the compiler. It operates under the control of preprocessor command lines or directives. Preprocessor directives are placed in the source program before the main line. Before the source code passes through the compiler, to is examined by the preprocessor for any preprocessor directive. If there are any, appropriate actions are taken and then the source program is handed over to the compiler.

Preprocessor directives follow special syntax that is different from normal C syntax. They all begin with the symbol # and do not require a semi-colon at the end of the statement.

Preprocessor directives:

Directive	Function
#define	Defines a macro substitution
#undef	Undefines a macro
#include	Specifies the files to be included
#ifdef	Test for macro definition
#endif	Specifies the end of #if
#if	Test a compile time condition
#else	Specifies alternatives when #if test fails

These directives can be divided into three categories. They are

1. Macro substitution directives
2. File inclusion directives
3. Compiler control directives

12.1 Macro substitution directives:

Macro substitution is a process where an identifier in a program is replaced by a predefined string composed of one or more tokens. The preprocessor accomplishes this task using the #define statement. This statement is known as macro definition. The general form of macro definition is

#define identifier string

If this statement is included at the beginning of the program the preprocessor replaces every occurrence of the identifier in the source code by the string. The different forms of macro substitution are

a. Simple macro substitution
b. Argumented macro substitution
c. Nested macro substitution

12.1.1 Simple macro substitution:

Simple string replacement is commonly used to define constants.

Examples:

```
#define PI 3.141592
#define MAX 10
```

12.1.2 Macros with arguments:

The preprocessor permits us to define more complex and useful form of replacements. It takes the following form

```
#define identifier(f1,f2,..fn) string
```

Example:

```
#define CUBE(x)       (x*x*x)
```

Inside the main program, whenever the following statement is encountered

```
volume=CUBE(s);
```

the preprocessor will expand the statement to

```
volume=(s * s * s);
```

12.1.3 Nesting of macros:

We can also use one macro in the definition of another macro. Macro definitions can be nested.

Example:

```
#define M 5
#define N M+2
```

The preprocessor expands each #define macro, until no more macros appear.

12.2 File inclusion:

An external file containing functions or macro definitions can be included as a part of a program so that we need not rewrite those functions or macro definitions. This can be achieved as follows:

```
#include<filename>
```

Here, the preprocessor inserts the entire contents of the filename into the source code.

Example:

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

12.3 Compiler Control Directives:

These are the directives which can be used for conditional compilation. Here, we can make the compiler skip a part of source code by inserting the preprocessing commands #ifdef and #endif. The general form of it is

```
#ifdef macroname
    Statement 1;
    Statement 2;
    Statement 3;
#endif
```

If the macro name is defined the block of code will be executed as usual, otherwise not. It could be used in the following cases

a. To comment out obsolete lines of code.
b. To make the programs portable
c. To check multiple declaration of files

12.4 Command Line arguments:

It is possible to accept command line arguments. Command-line arguments are given after the name of a program in command-line operating systems like DOS or Linux, and are passed in to the program from the operating system. To use command line arguments in your program, you must first understand the full declaration of the main function, which previously has accepted no arguments. In fact, main can actually accept two arguments: one argument is number of command line arguments, and the other argument is a full list of all of the command line arguments.

Syntax:

```
int main ( int argc, char *argv[] )
```

The integer, argc is the **arg**ument **c**ount. It is the number of arguments passed into the program from the command line, including the name of the program.

The array of character pointers is the listing of all the arguments. argv[0] is the name of the program, or an empty string if the name is not available. After that, every element number less than argc is a command line argument. You can use each argv element just like a string, or use argv as a two dimensional array. argv,[argc] is a null pointer.

Assignment

2 mark Questions:

1. Define preprocessor.(2012)
2. Define a macro.
3. What do you mean by command line arguments?(2012)
4. What are the rules to define a preprocessor directive?
5. What do you mean by nesting of macros? Give example.

5 mark Questions:

1. Explain the concept of macro substitution in C?(2012)
2. Explain file inclusion with example?
3. Write a note on command line arguments.

CHAPTER 13

FILE MANAGEMENT IN C

13.1 Introduction:

We use printf and scanf functions to write and read data from I/O devices. If the data is small then, it is fine. But, most of the real life problems deal with voluminous data. In such cases, entering data through the console becomes very difficult.

It is necessary to have a flexible approach where in data is stored on secondary storage devices like disks and read data from it whenever necessary without destroying data. A file is place on a disk where a group of related data is stored. The basic operations on files are

- Naming a file
- Opening a file
- Reading data from a file
- Writing data on to a file
- Closing a file

13.2 Defining and opening a file:

A file name is a string of characters that make up a valid file name for the operating system. A file name may contain 2 parts, a primary name and an extension that follows the period.

A file name is defined using the data structure FILE in the library of the standard I/O function definitions. A FILE is a defined data type

The general format of defining a file is

```
FILE *fp;
fp=fopen("filename", "mode");
```

The first statement here declares the variable fp as a pointer to the data type FILE. The second statement opens the file named filename and assigns it to the identifier fp. It also specifies the

mode in which the file should be opened. It specifies the purpose of opening the file. A file can be opened in any one of the following modes

r-read mode
w-write mode
a- append mode

- When a file is opened in write mode, a file is created if the file does not exists. If it already exists, the contents are deleted
- When a file is opened in the append mode, the contents remain intact if the file already exists. If not, a new file is created
- When a file is opened in read mode, it is opened with the contents safe, if is already exists. Otherwise an error occurs

13.3 Closing a file:

A file must be closed when all the operations associated with it are completed. This ensures that all the information associated with the files are flushed out of the buffer and all links to the file are broken. The function to close a file is fclose. The general format is

fclose(file-pointer);

13.4 I/O operations in files:

getc: This function is used to read a single character from a file. The general format of this function is

var=getc(file-pointer);

Example:

c=getc(fp);

putc: This function is used to write a single character onto a file. The general format is

putc(var, file-pointer);

Example:

putc(c,fp);

getw: This function is used to read a single integer value from a file. The general format is

var=getw(file-pointer);

Example:

num=getw(fp);

putw: This function is used to write a single integer value onto a file.The general format is

putw(var, file-pointer);

Example:

putw(num,fp);

fscanf: This function is used to read mixed data type values from a file. The general format is

fscanf(file-pointer, " control string" , list);

Example:

fscanf(fp1, "%s %d " , item,&qty);

fprintf: This function is used to write mixed data type values onto a file. The general format is

fprintf(file-pointer, " control string" , list);

Example:

```
fprintf(fp1, "%s%d", name,qty);
```

13.5 Error Handling during I/O operations:

It is possible that an error may occur during an I/O operation on a file. Typical error situations include

1. Trying to read beyond the end-of-file mark
2. Device overflow
3. Trying to use a file that has not been open
4. Opening a file with invalid name
5. Trying to perform an operation on a file, when a file is opened for another type of operation.

If we fail to check such errors it may lead to abnormal termination of the program or incorrect output. To handle such situations we have two status-inquiry library functions: **feof** and **ferror** that help in detecting I/O errors in files.

The **feof** function can be used to test for an end-of-file condition. It takes file pointer as an argument and returns a non-zero integer value if all the data from specified file have been read, and returns zero otherwise. If fp is a pointer to a file that has just been opened for reading, then, the statement

```
if( feof(fp))
  printf("End of data\n");
```

would display the message "End of data" on reaching the end of file condition

The **ferror** function reports the status of the file indicated. It takes file pointer as an argument and returns a non-zero integer value if an error has been detected up to that point, during processing. It returns zero otherwise. The statement

```
if( ferror(fp)!=0)
  printf("An error has occurred\n");
```

would display the message if the reading is not successful.

We can also check whether a file is opened or not. When a file is opened using fopen function, a file pointer is returned. If the file cannot be opened for some reason, then the function returns a NULL pointer.

Example:

```
if(fp == NULL)
printf("File Cannot be opened\n");
```

Assignment

2 mark Questions:

1. Define a file.
2. How do you declare a file in C?
3. How do you open a file in C?
4. List the various operations performed on files. (2006,2009)
5. How do you close a file? Give example.
6. Why do you use feof function?
7. What is the purpose of ferror function?

5 mark Questions:

1. Explain the concept of opening a file in C? (2008,2009)
2. How do you perform I/O operations on files in C? Explain with syntax and example(2007,2009, 2011)

CHAPTER 14
Advanced Topics in C

14.1 Variable-Length Argument Lists

- It is possible to have functions that take variable number of arguments
- The best example is the printf function

– printf must receive a string as its first argument, but can receive any number of additional arguments
– The prototype for printf is specified as
int printf (const char *, ...);

- The ellipsis in the function (...) indicates that the function receives a variable number of arguments of any type
- The ellipsis cannot be placed in the middle of the parameter argument list (it must be at the end of the list)

14.2 Program Parameters

- Defined by specifying parameters with the function main
- Useful in specifying command-line parameters for programs
- Possible to send any number of parameters to the program through the use of just two parameters

1. argc
– Number of arguments (argument count)
– Declared as int
2. *argv[]
– Entire command line (argument vector)
– Declared as char * (character array)
– argv[0] points to the command itself
– argv[1] points to the first argument (program parameter)
– argv[2] points to the second argument (program parameter)
.
.
.

Note:
∗ Each array element argv[i] is a pointer to a character
∗ Name of character constant and string arrays are also pointers to characters
∗ An array of pointers can be regarded as a pointer to the first array element, which itself is a pointer
∗ The name argv is a pointer to a pointer and can also be written as **argv

Example:
Program to compute factorial

```
/*****************************************************************************
*/
/* main.c */
```

```
#include <stdio.h>
int fact ( int ); /* Function prototype */
int main ( int argc, char *argv[] )
{
int x; /* Variable to hold parameter */
if ( argc != 2 )
{
printf ( "Usage: %s <number>\n", argv[0] );
exit ( 1 );
}
x = atoi ( argv[1] );
printf ( "Factorial of %d is %d\n", x, fact(x) );Advanced Topics 2
return ( 0 );
}
/*************************************************************************
*/
/* fact.c */
int fact ( int x )
{
return ( x ? ( x * fact ( x - 1 ) ) : 1 );
}
/*************************************************************************/
```

14.3 External variables and program structure

- External to the module in which they are used
- Allows for permanent allocation of memory
- External variable has only one definition but can have several declarations

– Declaration specifies the attributes of a variable
– Definition specifies the attributes of a variable as well as allocates memory
definition = declaration + memory allocation

- External variables are defined outside the functions (contrast with automatic variables)
- The keyword extern in function declarations can be omitted without causing any problems

Example:

```
/*************************************************************************/
#include <stdio.h>
extern int counter; /* Declaration of counter */
extern void inc_counter ( void );
main()
{
int index;
for ( index = 0; index < 10; index++ )
inc_counter();
printf ( "Counter is: %d\n", counter );
```

```
}
/*************************************************************************
*/
int counter; /* Definition of counter */
void inc_counter ( void );
{
counter++;
}
/*************************************************************************/
```

14.4 Pointers to functions

- Functions stored in memory much the same way as data

– Functions have addresses that can be assigned to pointers
– Notation for these pointers is cryptic and we have to be extremely careful with the parentheses and the unary operator *

- If p is a pointer to a function that has no parameters, a call to that function can be written as (*p)()
- The parentheses in (*p) cannot be omitted since in *p(), the parentheses have higher precedence than * and the function is read as *(p()) which is incorrect and denotes the object pointed to by the value returned by p, assuming that p is a function returning a pointer
- The parentheses surrounding *p also occur in declarations and prototypes

Example

```
/* pfunc.c : Illustrating a pointer to a function */
#include <stdio.h>
#include <ctype.h>
#include <math.h>
main()

{
char ch;
double (*p)( double );
printf ("Enter c, s, or t to select one of the functions cos, sin, tan: ");
switch ( ch = tolower ( getchar () ) )
{
case ’c’: p = cos; break;
case ’s’: p = sin; break;
case ’t’: p = tan; break;
default : printf ( "Wrong character\n" ); exit ( 1 );
}
printf ( "\nArgument Function value\n" );
printf ( "%8.3f %12.8f\n", 0.1, (*p)(0.1) );
printf ( "%8.3f %12.8f\n", 0.3, (*p)(0.3) );
printf ( "%8.3f %12.8f\n", 1.6, (*p)(1.6) );
```

```
}
```

14.5 Passing Function names as Parameters to other functions

- Useful in passing the function name as a parameter to other functions
  - In numerical analysis, a general integration routine is called with the interval boundaries and the function to be integrated as arguments

Example:

Compute the following series

sum= f(1)+f(1/2)+f(1/3)+…..+f(1/n)

```
/*************************************************************************/
/* main.h */
#include <stdio.h>
double fn_sum ( int, double (*)() ); /* Function to be summed */
double square ( double );
double cube ( double );
/*************************************************************************
*/
/* main.c */
/* func_arg: Passing a function name as an argument */
#include "main.h"
main()
{
printf ( "Sum of squares: %f\n", fn_sum( 5, square ) );
printf ( "Sum of cubes : %f\n", fn_sum( 5, cube ) );
}
/*************************************************************************/
/* fn_sum.c */
double fn_sum ( int n, double (*fn)() )
{
double s = 0;
int i;
for ( i =1; i <= n; s += (*fn)(1.0/i++) );
return ( s );
}
/*************************************************************************/
/* square.c */
double square ( double x )
{
return ( x * x );
}
/*************************************************************************/
/* cube.c */
double cube ( double x )
```

```
{
return ( x * x * x );
}
/*************************************************************************/
```

14.6 In-memory format conversion

- Allows the conversion from internal binary format to display (decimal) format without putting it onto screen or file
- Resulting character sequence appears as a string variable
- Achieved by the function sprintf
- The following two statements are equivalent, using the declaration int i = 123; char str[8];

```
sprintf ( str, "%4d", 2*i );
strcpy ( str, " 246" );
```

- The target string variable should have enough space to accommodate the output string
- The target variable (str in above example) could be a pointer in the middle of character array (string)
- sscanf function

– Same as scanf with in-memory scanning of string
– Better substitute for scanf when used in conjunction with fgets

Credit Based First Semester B.C.A. Degree Examination, Oct./Nov. 2014
(New Syllabus) (2012-13 Batch Onwards)
PROGRAMMING IN C

Time: 3 Hours Max. Marks: 80

Note: Answer any ten questions from Part A and any one full question from each Unit of Part B.

PART -A (2x10=20)

1. a) What is the difference between character constant and string constant? Give example for each.
b) What is a variable? How do you declare a variable?
c) Determine the value of each of the following assume a = 6, b = -2 and c = - 7 :
i) X = (a > b) ? a + 3 : b + 3 * c .
ii) Y = pow (sqrt (9), 3)
d) What is the use of break and continue statements in a program?
e) What do you mean by scope and life time of a variable?
f) What is an entry controlled loop? Give example.
g) Differentiate array and structure.
h) Differentiate i ++ and ++ i with example.
i) List the categories of user defined functions.
j) What is a pointer? How is it declared?
k) What is a union? How does it differ from structure?
l) What is a file? List an advantage of a file.

PART-B

UNIT-I

2. a) What are identifiers? Write the naming rules of identifiers.
b) List and explain arithmetic and relational operators in C with example.
c) Explain the formatted output function with its syntax and example. (5+6+4)

3. a) Give the basic structure of C program and explain each section.
b) List and explain the fundamental data types in C?
c) What is type conversion? Explain different types of type conversion.
(5+5+5)

UNIT -II

4. a) What is a loop? Explain any two looping statements with its syntax and example.
b) Explain the switch statement with syntax and example.

c) Write a program to reverse a given number and check for palindrome. (5+5+5)

5. a) Explain different forms of if statement with suitable examples.
b) Explain break and continue with suitable example code.
c) Write a program to N elements in the descending order. (5+5+5)

UNIT –III

6. a) Explain any four string handling functions with syntax and example.
b) Explain any two categories of user-defined functions with examples.
c) What is Recursion? Write a program to find the factorial of a number using recursion. (5+5+5)

7. a) Explain the following in C with suitable examples.
i) Register Variables
ii)Local Variables
b) What is a function? Write the general form of User-defined function. Give an example.
c) State the purposes of gets(), getchar() and puts() with syntax and example. (4+5+6)

UNIT –IV

8. a) a) Explain the following:
i) Array of structures
ii) Nested Structures

b) Explain pointer expression with example.
c) Explain the different modes of opening a file with example. (5+6+4)

9. a) What are preprocessor directives? Explain any two types of preprocessor directives.
b) Explain the usage of Bit fields in structure.
c) Distinguish between the following:
i) printf() and fprintf()
ii) feof() and ferror()
iii) getc() and getchar() (5+4+6)