DEPARTMENT OF INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING & TECHNOLOGY

LAB MANUAL
FOR

SUBJECT CODE: 15IT102L
SUBJECT: PROGRAM DESIGN AND DEVELOPMENT LABORATORY

PREPARED BY
P.SUBHA, G.MARAGATHAM
Assistant Professor (Ordinary Grade)

SRM University, SRM Nagar, Kattankulathur-603203
Kancheepuram District, Tamil Nadu

SYLLABUS

		L	T	P	C
15IT102L	**Program Design and Development**	**3**	**0**	**2**	**3**

PURPOSE

Knowledge of problem solving and programming concepts is essential for those who develop applications for users. Hence to provide the required knowledge, this course imparts basic knowledge in C programming along with the concepts of design and development of programs using C.

INSTRUCTIONAL OBJECTIVES

After successful completion of the course, the students should be able to,
O1 – Gain knowledge about problem solving in computers.
O2 – Understand the basic components and structure of a C program.
O3 – Develop proficiency in basic programming skills.

LIST OF EXERCISES (30 hours)

1. Programs to demonstrate the use of scanf( ) and printf( ) functions
2. Programs to evaluate arithmetic expressions
3. Programs using conditional statements
4. Programs using for-while - do…while
5. Programs on arrays
6. Programs to perform matrix addition and multiplication
7. Programs to implement functions
8. Programs to illustrate recursion
9. Programs to illustrate pointers

Reference Book (s):

R1. E.Balagurusamy,"*Programming in ANSI C"*, Tata McGrawHill, 6thEdition, 2011.
R2. P.B.KOTUR, "Computer Concepts and C Programming", Sapna Edition, Bangalore.
R3. Y.P. Kanetkar, "*Let us C*", BPB Publications, 8th Edition, 2008.
R4. Steve Oualline, "*Practical C Programming*", O'Reilly Publishers, 2011.
R5. Byron Gottfried, "*Programming with C*", Schaum's Outline Series, 2nd Edition, 2000.

Rubrics

Experiment Component	Max. Marks	Grading Rubrics		
Algorithm/ Flowchart	2	Algorithms are written clearly as pseudo codes, focusing and mentioning, every step to solve the given problem (demonstrates good clarity in understanding the problem statement). Appropriate use of proper symbols for different operations in flowchart. Correctness of logic in algorithm/flowchart is taken care off. **[2 marks]**	Algorithms are written as pseudo codes, but lacks clarity in identifying different independent steps, to solve the given problem (demonstrates less clarity in understanding the problem statement). Use of inappropriate symbols for few operations in flowchart. Correctness of logic in algorithm/flowchart is taken care off. **[1 mark]**	
Program	3	Completeness of code, consistent variable naming and formatting, well commented, uses existing skills in new ways/learns new skills to solve the experimental problem. **[3 marks]**	Completeness of code, inconsistent variable naming and formatting, lacks clarity in commenting, uses existing skills to solve the experimental problem. **[2 marks]**	Completeness of code, improper variable naming and unformatted, lacks comments, uses existing skills to partially solve the experimental problem. **[1 mark]**
Program Execution	3	Program is free of errors and output is well formatted. Demonstrates Excellent problem solving and creativity skills. **[3 marks]**	Program is free of errors and output is not properly formatted. Demonstrates a clear understanding of the concepts relevant to the experiment. **[2 marks]**	Program contains few logical errors and output is not formatted. Demonstrates partial understanding of the concepts relevant to the experiment. **[1 mark]**
Program Testing	2	Decision control logic, loop logic in program exhibits proper functional behavior and Output is obtained for varying sets of input data. **[2 marks]**	Decision control logic, loop logic in program lacks proper functional behavior and Output is obtained for only few sets of input data. **[1 mark]**	

Note: A student who does not secure a minimum of 4 marks should redo the experiment.

LIST OF EXERCISES

1. Study of UNIX commands
2. Basic Programs in C
3. a) Swapping of two numbers using temporary variable
   b) Swapping of two numbers without using temporary variable

Decision making – Branching

4. Greatest of three numbers
5. Use of Switch...Case statements - Counting number of vowels and digits in a string

Decision making - Looping

6. Sum of digits
7. Generation of prime numbers
8. Generation of Fibonacci Series

Functions

9. Factorial of a given number using Function
10. Functions with Static data type

Arrays

11. Matrix Addition
12. a) Matrix Transpose
    b) Matrix Multiplication

Strings

13. Palindrome checking
14. Removal of duplicate string from the given sentence

Structures

15. Sorting of items using array of structures
16. User-defined data type using structure

Pointers

17. Basic Programs on pointers
18. Display sum of array using pointers
19. Sorting of names using array of pointers
20. Using pointers compute mean and standard deviation

B.TECH - INFORMATION TECHNOLOGY **I Year**

15IT102L **PROGRAM DESIGN & DEVELOPMENT USING C**

INDEX

EX. No. : 1 **STUDY OF UNIX COMMANDS**

AIM:

To study and practice the command features of UNIX environment for program development.

UNIX Commands Overview:

I. Directory command

1.mkdir:

- **Description:**
  To create an directory
- **Syntax:**
  mkdir <dirname>

2. rmdir:

- **Description:**
  To delete a directory
- **Syntax:**
  rmdir <dir name>

3. cd:

- **Description:**
  To change from current to another directory.
- **Syntax:**
  cd ---------- Move to source directory
  cd ----------moves to the subdirectory
  cd absolute_address---------move to the particular directory.

II. FILE MANIPULATION COMMANDS:

- **cat:**

- **Description:**
  To create and view the content of a file
- **Syntax:**
  i. cat file ----------view the contents of file
  ii. cat> filename ----------creation of new file

- **cp:**

- **Description:**
  To copy the content of file(s) to another file (over write the content)
- **Syntax:**
  cp source_file destination_file

- **MV:**

- **Description:**
  To move the contets of an file to another file(similar to rename).
- **Syntax:**
  mv source_file destination_file

- ***rm:***

- **Description:**
  To delete the file
- **Syntax:**
  rm file

- **Sort:**

- **Description:**
  To sort the contents of a file
- **Syntax:**
  sort < file>

- **Head:**

- **Description:**
  To display the first n line of a file
- **Syntax:**
  head –n <file>

- **tail:**

- **Description:**
  To display the last “n” lines of a file
- **Syntax:**
  t***ail*** –n <file>

- **pg:**

- **Description:**
  To display one full screen content of a file at a time
- **Syntax:**
  pg <file>

- **lg:**

- **Description:**
  To print the content of a file using a printer.
- **Syntax:**
  l***g*** -n <file>

- **word count**:

- **Description:**
  To display the line, word and character count
- **Syntax:**
  wc <file name>

- **Compare:**

- **Description:**
  To compare the content of files
- **Syntax:**
  cmp <file1> <file2>

- **Difference**:

- **Description:**
  To display the difference between files
- **Syntax:**
  diff <file1> <file2>

- **Grep:**

- **Description:**
  To globally search, the regular expression.
- **Syntax:**
  grep search_text <filename>

III . Information commands

- ***who:***

  Description:
  To display the list of user currently logged into the system.
  Syntax:
  Who<filename>

- ***who am i:***

  Description:
  To display the system numbers, month, date and time.
  Syntax:
  Who am i

- ***present work directory:***

  Description:
  To display the present working directory form the root directory absolute address.

Syntax:

Pwd

- ***date:***

Description:

To display the system date

Syntax:

Date

- ***cal:***

Description:

To display the calendar of any year and month

Syntax:

cal month year

- ***read:***

Description:

To get the value of the variable

Syntax:

read <variable name>

- ***echo***

Description:

To display the ensuring text in the screen.

Syntax:

echo $<variable name>

VI .Listing command and its options

- **ls** : To display the list of files in the current working directory
- **ls -t:** List in order, at first modification time
- **ls -l:** List files, aling with the defail like mode,number of links, files, size, owner of the file, modification date and time.
- **ls -a:** List all files including hidden files.
- **ls -1**:To show single entry per line.

V. I/O redirection commands:

- > - Output to be stored in the directed file and not on the user terminal.
- < - Input to be taken from a file
- >> - Output to be appended to the end of file content and not to be over written.
- | - Pass output of one command from left of symbol to another command to the right of the symbol.

VI. Security and protection commands:

Following are the symbolic representation of three different roles:

u is for user,

g is for group,

and **o** is for others.

Following are the symbolic representation of three different permissions:

r *is for read permission,*

w *is for write permission,*

x *is for execute permission*

Chmod - To change the protection mode of the file.

1. Add single permission to a file/directory

Changing permission to a single set. + symbol means adding permission. For example, do the following to give execute permission for the user irrespective of anything else:

$ chmod u+x filename

-

2. Add multiple permission to a file/directory

Use comma to separate the multiple permission sets as shown below.

$ chmod u+r,g+x filename

3. Remove permission from a file/directory

Following example removes read and write permission for the user.

$ chmod u-rx filename

4. Change permission for all roles on a file/directory

Following example assigns execute privilege to user, group and others (basically anybody can execute this file).

-

$ chmod a+x filename

- **Chown -** To change the owner of a file.
-
- **Syntax:**
- Chown owner_user filename
-
- **Chgrp -** To change the group.

Example:

To change the group name of a file or directory

```
$ ls -l
-rw-r--r-- 1 john john 210 2011-01-04 16:01 /home/john/sample.txt

$ chgrp user /home/john/sample.txt

$ ls -l
-rw-r--r-- 1 john user 210 2011-01-04 16:01 /home/john/sample.txt
```

VII. Process and argument commands:

ps - Prints the status of various processes.
kill - Stop or terminate a process.
wait - Wait for background program.
sleep - Suspends execution for an interval.

VIII. Editor commands:

i.*Vi editor:*

Description:
To create and view files especially of shell program and C programs file type.

Syntax:
Vi filename

- ***Cursor movements*:**
  h,j,k,l left, down, up and right.

- ***Character Input modes:***
  a-append after cursor
  A-append at end of cursor
  i-insert before cursor
  I -Insert before first non-blank
  o-Add lines after current lines
  O-Add lines before current lines

- ***Delete and change:***
  dd-delete line
  cc-change line

- ***Control commands:***
  :w-save the file
  :wq-save file and quit
  :q- quit

ii**. ed**

n-print line
a-append text
d-delete line
h-explain last error
H-automatically explain all error
w-save editing changes

q-quit the editor.

C/C++ PROGRAM COMPILATION

We begin by outlining the basic processes you need to go through in order to compile your C (or C++) programs. We then proceed to formally describe the C compilation model and also how C supports additional libraries.

Creating, Compiling and Running Your Program

The stages of developing your C program are as follows:

Creating the program

Create a file containing the complete program, such as the above example. You can use any ordinary editor with which you are familiar to create the file. One such editor is ***textedit*** available on most UNIX systems another one is ***vi*** editor.

The filename must by convention end ``.c'' (full stop, lower case c),

e.g. myprog.c or ***progtest.c***. The contents must obey C syntax. For example, they might be as in the above example, starting with the line `/* Sample ....` (or a blank line preceding it) and ending with the line `} /* end of program */` (or a blank line following it).

Compilation

There are many C compilers around. The `cc` being the default Sun compiler. The GNU C compiler `gcc` is popular and available for many platforms. PC users may also be familiar with the Borland `bcc`compiler.

There are also equivalent C++ compilers which are usually denoted by cc (***note*** upper case CC. For example Sun provides CC and GNU `GCC`. The GNU compiler is also denoted by `g++`

To compile your program simply invoke the command `cc`. The command must be followed by the name of the (C) program you wish to compile.

Thus, the basic compilation command is:

```
cc program.c
```

where ***program.c*** is the name of the file.

If there are obvious **errors** in your program (such as mistypings, misspelling one of the key words or omitting a semi-colon), the compiler will detect and report them.

There may, of course, still be **logical errors** that the compiler cannot detect. You may be telling the computer to do the wrong operations.

When the compiler has successfully digested your program, the compiled version, or executable, is left in a file called ***a.out*** or if the compiler option ***-o*** is used : the file listed after the ***-o***.

It is more convenient to use a ***-o*** and filename in the compilation as in

```
cc -o program program.c
```

which puts the compiled program into the file program (or any file you name following the "-o" argument) **instead** of putting it in the file a.out .

Running the program

The next stage is to actually run your executable program. To run an executable in UNIX, you simply type ***./a.out***

This executes your program, printing any results to the screen. At this stage there may be **run-time errors**, such as division by zero, or it may become evident that the program has produced incorrect output.

If so, you must return to edit your program source, and recompile it, and run it again.

The C Compilation Model

Key features of the C Compilation model (Fig. 1.1).

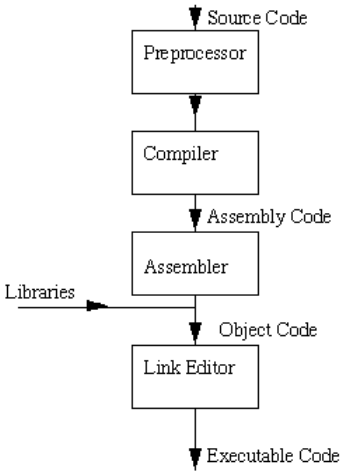

Fig. 1.1 The C Compilation Model

The Preprocessor

The Preprocessor accepts source code as input and is responsible for

- removing comments
- Interpreting special ***preprocessor directives*** denoted by `#`.

For example

- `#include` -- includes contents of a named file. Files usually called ***header*** files. ***e.g***
  - `#include <math.h>` -- standard library maths file.
  - `#include <stdio.h>` -- standard library I/O file
- `#define` -- defines a symbolic name or constant. Macro substitution.
  - `#define MAX_ARRAY_SIZE 100`

C Compiler

The C compiler translates source to assembly code. The source code is received from the preprocessor.

Assembler

The assembler creates object code. On a UNIX system you may see files with a `.o` suffix (`.OBJ` on MSDOS) to indicate object code files.

Link Editor

If a source file references library functions or functions defined in other source files the ***link editor*** combines these functions (with `main()`) to create an executable file. External Variable references resolved here also.

Some Useful Compiler Options

Now that we have a basic understanding of the compilation model we can now introduce some useful and sometimes essential common compiler options.

`-c`

Suppress the linking process and produce a `.o` file for each source file listed. Several can be subsequently linked by the `cc` command, for example:

```
cc file1.o file2.o ..... -o executable
```

`-llibrary`

Link with object libraries. This option must follow the source file arguments. The object libraries are archived and can be standard, third party or user created librarie. Probably the most commonly used library is the math library (`math.h`). You must link in this library explicitly if you wish to use the maths functions (**note:** do not forget to `#include <math.h>` header file), for example:

```
cc calc.c -o calc -lm
```

`-Ldirectory`

Add directory to the list of directories containing object-library routines. The linker always looks for standard and other system libraries in `/lib` and `/usr/lib`. If you want to link in libraries that you have created or installed yourself (unless you have certain privileges and get the libraries installed in `/usr/lib`) you **will** have to specify where you files are stored, for example:

```
cc prog.c -L/home/myname/mylibs mylib.a
```

`-Ipathname`

Add pathname to the list of directories in which to search for #include files with relative filenames (not beginning with slash /).

BY default, The preprocessor first searches for #include files in the directory containing source file, then in directories named with -I options (if any), and finally, in /usr/include. So to include header files stored in `/home/myname/myheaders` you would do:

```
cc prog.c -I/home/myname/myheaders
```

Note: System library header files are stored in a special place (`/usr/include`) and are not affected by the `-I option`.

`-g`

invoke debugging option. This instructs the compiler to produce additional symbol table information that is used by a variety of debugging utilities.

`-D`

define symbols either as identifiers (`-D`***identifer***) or as values (`-D`***symbol=value***) in a similar fashion as the `#define` preprocessor command.

Using Libraries

C is an extremely small language. Many of the functions of other languages are not included in C. ***e.g.*** No built in I/O, string handling or maths functions.

What use is C then?

C provides functionality through a rich set function libraries. As a result most C implementations include ***standard*** libraries of functions for many facilities (I/O ***etc.***). For many practical purposes these may be regarded as being part of C. But they may vary from machine to machine. (***cf*** Borland C for a PC to UNIX C).

A programmer can also develop his or her own function libraries and also include special purpose third party libraries (***e.g.*** NAG, PHIGS).

All libraries (except standard I/O) need to be explicitly linked in with the `-l` and, possibly, `-L` compiler options described above.

UNIX Library Functions

The UNIX system provides a large number of C functions as libraries. Some of these implement frequently used operations, while others are very specialised in their application.

Do Not Reinvent Wheels: It is wise for programmers to check whether a library function is available to perform a task before writing their own version. This will reduce program development time. The library functions have been tested, so they are more likely to be correct than any function which the programmer might write. This will save time when debugging the program.

Lint -- A C program verifier

C compiler is pretty vague in many aspects of checking program correctness, particularly in type checking. Careful use of prototyping of functions can assist modern C compilers in this task. However, There is still no guarantee that once you have successfully compiled your program that it will run correctly.

The UNIX utility `lint` can assist in checking for a multitude of programming errors. Check out the online manual pages (`man lint`) for complete details of lint. It is well worth the effort as it can help save many hours debugging your C code.

To run lint simply enter the command: `lint myprog.c`.

Lint is particularly good at checking type checking of variable and function assignments, efficiency, unused variables and function identifiers, unreachable code and possibly memory leaks.

EX. No. : 2 BASIC C PROGRAMS

SAMPLE 1: Write a program to print Simple Interest.

```
/* program to find simple interest */
#include<stdio.h>
int main( )
{
        float p,n,r,si;
        clrscr();
        printf("Enter principle, Number of years and Rate of Interest :\n");
        scanf(%f%f%f",&p,&n,&r);
        si=(p*n*r/100);
        printf("The Simple Interest is : %2.2f: ",si);
        return 0;
}
```

OUTPUT:-

```
Enter principle, Number of years and Rate of Interest:
5000    5    10
The Simple Interest is: 2500.00
```

SAMPLE 2: Write a program to generate Sales slip.

```
/********** SALES SLIP **********/
#include <stdio.h>
int main( )
{
        float purchase, tax_amt, total, rate=0.125;
        clrscr();
        printf("\n Amount of purchase: ");
        scanf("%f",&purchase);
        tax_amt = purchase * rate;
        total = purchase + tax_amt;
        printf("\n Purchase is  : %0.2f ",purchase);
        printf("\n Tax           :  %0.2f",tax_amt);
        printf("\n Total         : %0.2f",total);
        return 0;
}
```

OUTPUT 1:

```
Amount of purchase: 500
Purchase is: 500.00
 Tax          :   62.50
 Total        : 562.50
```

OUTPUT 2:

```
Amount of purchase: 1773
Purchase is: 1773.00
 Tax          :   221.62
 Total        : 1994.62
```

RESULT:

Thus the command options in the UNIX environment have been studied and basic C programs are compiled and executed adopting UNIX commands.

EX. No. : 3 SWAPPING OF TWO NUMBERS

AIM

To write a C program to swap two numbers using temporary variable and without using temporary variable.

ALGORITHM

1. Start
2. Read two numbers A, B
3. Copy A to TEMP
4. Copy B to A
5. Copy TEMP to B
6. Print A, B
7. Without Temporary Variable, Calculate A=A+B, B=A-B, A=A-B
8. Display A, B
9. Stop

RESULT

Thus swapping of two numbers using temporary variable and without using temporary variable was done and successfully executed.

EX. No. : 4 GREATEST OF THREE NUMBERS

AIM:

To write a C program to find the largest of three numbers using conditional and IF statement.

ALGORITHM

1. Start
2. Read three numbers A, B, C
3. Compare whether A is greater than B and C using conditional operator
4. If it is equal then print “A is greater”
5. If step 3 is false check whether B is greater than A and C using conditional operator.
6. If it is true print “B is greater”
7. If step 5 is false print “C is greater”
8. Stop

RESULT

Thus the largest of three numbers using conditional and if statement was successfully executed.

EX. No. : 5 COUNTING NUMBER OF VOWELS AND DIGITS IN A STRING

AIM

To write a C program to count the number of vowels and digits in a given string using switch case statement.

ALGORITHM

1. Start
2. Read a string CH
3. Loop until CH is equal to new line
4. Check whether the entered character is vowel or a digit using switch case
5. If it is vowel increment VOWEL
6. If it is digit increment DIGIT
7. End loop
8. Print VOWEL, DIGIT
9. Stop

RESULT

Thus a C program to count the number of vowels and digits using switch case was compiled and executed successfully.

EX. No. : 6 **SUM OF DIGITS**

AIM:

To write a C program to result the sum of digits of given number.

ALGORITHM:

1. Start the program.
2. Initialize s=0
3. Read the given number in variable N
4. Repeat while(n!=0)
5. Set, x=n%10
6. Assign, s=s+x
7. Set, n=n/10
8. Print sum
9. Stop

RESULT:

Thus the C program to result sum of digits of given number was compiled and executed successfully.

EX. No. : 7 GENERATION OF PRIME NUMBERS

AIM:

To write a C program to generate all prime numbers in a given range.

ALGORITHM:

1: Start the program.

2: Get the range of prime numbers N

3: Check if N= =1 if equal go to step 4

4: Print number is not valid to generate prime.

5: By using while loop check (i<=N).

6: Assign c=0 check the process j<=i using for loop.

5: Check for condition (i%j==0) then c=c+1.

6: Check for the condition whether c=2, then display the value.

7: Stop

RESULT:

Thus the C program to generate prime numbers in the given range was compiled and executed successfully.

EX. No. : 8 **GENERATION OF FIBONACCI SERIES**

AIM:

To write a c program to generate the Fibonacci series.

ALGORITHM:

1. Start
2. Initialize FIB1 = 0, FIB2 = 1, FIB3, NUMBERS = 2, COUNTER = 2
3. Read NUMBERS
4. If NUMBERS<3 stop the program
5. Print FIB2
6. Loop until COUNTER<=NUMBERS
7. Increment COUNTER by 1
8. FIB3=FIB1+FIB2
9. Print FIB3
10. Copy FIB2 to FIB1
11. Copy FIB3 to FIB2
12. End loop
13. Stop

RESULT

Thus the Fibonacci series for a given range was compiled and executed successfully.

EX. No. : 9

FACTORIAL OF A GIVEN NUMBER USING FUNCTION

AIM

To write a C program to result factorial of given number using function.

ALGORITHM

1. Start
2. Read the number in variable a
3. Call the declared function fact_func() with actual parameter 'a'
4. Receive the return value from function in variable n
5. Print n
6. Stop

Function fact_func()

1. Get the value in formal parameter f
2. Initialize fact = 1
3. Repeat step 3 and 4 until f>=1
4. fact=fact*f
5. decrement the variable f by 1
6. return fact value after condition fails in step3

RESULT

Thus the C program to result factorial of given number using function was compiled and executed successfully.

EX. No. : 10 FUNCTIONS WITH STATIC DATA TYPE

AIM

To write a C program using functions with static data type.

ALGORITHM

7. Start
8. Call function f1()
9. Print “after first call”
10. Call function f1()
11. Print “after second call”
12. Call function f1()
13. Print “after third call”
14. Stop

Function f1()

7. Initialize k= 0 as static and j=10
8. Print the value of k and j
9. Calculate k=k+10

RESULT

Thus the C program using functions with static data type was compiled and executed successfully.

EX. No. : 11 MATRIX ADDITION

AIM:

To write a C program to perform the Matrix Addition.

ALGORITHM

Addition

1. Start
2. Read the order of matrix M,N
3. Read the value of two matrices A,B using loop
4. Set loop i=0 to m and j=0 to N
5. Calculate c[i][j]=a[i][j]+b[i][j]
6. End loop
7. Print c[i][j]
8. Stop

RESULT

Thus the C program to perform the matrix addition was compiled and successfully executed.

EX. No. : 12 MATRIX OPERATIONS

AIM:

To write a C program to perform the Matrix Transpose and Multiplication operations

ALGORITHM

Muliplication

1. Start
2. Read the order of matrix M,N
3. Read the value of two matrices A,B
4. Initialize c[i][j]=0
5. Set loop i=0 to M j=0 to N and k=0 to N
6. Calculate c[i][j]=c[i][j]+a[i][k]*b[k][j]
7. End loop
8. Print c[i][j]
9. Stop

Transpose

1. Start
2. Read the order of matrix M,N
3. Read the value of two matrices A,B
4. Set loop i=0 to M and j=0 to N
5. Assign t[i][j]=a[j][i]
6. End loop
7. Print t[i][j]
8. Stop

RESULT

Thus the C program to perform the matrix operations was compiled and successfully executed.

EX. No. : 13 PALINDROME CHECKING

AIM

To write a C program to perform given string is palindrome or not a palindrome.

ALGORITHM

1. Start
2. Read the given string in variable s1
3. Calculate the length of string **s1** using strlen function and store it in variable cnt
4. Store the reversed string in another string r1 using loop
5. Until s1[cnt] reaches NULL character, store s1[cnt] = r1[index]
6. Assign, r1[index]=NULL character
7. Compare strings s1 and r1, strcmp returns 0 when the strings are equal
8. Or a negative integer when s1 is less than s2,
9. Or a positive integer if s1 is greater than s2,
10. If compare result is zero, print string s1 is palindrome or not a palindrome
11. Stop

RESULT

Thus the C program to check given string is palindrome or not a palindrome was compiled and executed successfully.

EX. No. : 14 REMOVING DUPLICATE STRING FROM A GIVEN SENTENCE

AIM:

To write a C program to remove the duplicate string from the given sentence.

ALGORITHM

1. Start
2. Read the sentence in bin variable from the user
3. Separate the words and store it in temp variable which is two dimension array
4. Compare the consecutive words in temp upto the word count
5. If it is not equal print the word else do not print that word
6. Repeat 4 and 5 until all the words in temp are compared
7. Stop

RESULT

Thus a C program to remove the duplicate string from the given sentence was compiled and executed successfully.

EX. No. : 15 ARRAY OF STRUCTURES

AIM

To write a C program to create an array of structures for a list of items with the following details

Item_Code	Item_Name
102	Paste-Colgate
102	Paste-Pepsodent
102	Paste-Close-up
101	Soap-Cinthol
101	Soap-Lux
101	Soap-Hammam
101	Soap-Dove

Arrange the set of items in ascending order of its Item_Code and descending order of its Item_Name as given below

Item_Code	Item_Name
101	Soap-Lux
101	Soap-Hamam
101	Soap-Dove
101	Soap-Cinthol
102	Paste-Pepsodent
102	Paste-Colgate
102	Paste-Close-up

ALGORITHM

1. Start
2. Define structure item with the members

   Item_code : integer

   Item_name : character

   end item
3. Read the given details in item_code and item_name
4. Set loop i=0 to 6 and j=1 to 6
5. Check if a[i].item_code<a[j].item_code then assign
6. Temp=a[i].item_code

7. A[i].item_code=a[j].item_code
8. A[j].item_code=temp
9. End loop i,j
10. Set loop i=0 to 6 and j=1 to 6
11. Check if a[i].item_name is less then a[j].item_code
12. Then assign a[i].item_name to temp
13. Assign a[j].item_name to a[i].item_name
14. Assign temp to a[j].item_name
15. End loop
16. Print item_code and item_name using loop
17. Stop

RESULT

Thus the list of items using array of structures was successfully compiled and executed.

EX. No. : 16 USER-DEFINED DATA TYPE USING STRUCTURE

AIM

To write a C program to create a user defined data type using structure to accept a list of N numbers of students with their names and roll numbers and arrange them in an alphabetical order and write the details in the file.

ALGORITHM

1. Start
2. Create a user-defined structure tag with the members
   name: character
   rno: integer
3. Open the file details .txt in writing mode
4. Read no for the number of student details
5. Read all the student details in name, rno
6. Before sorting print the student details and same time write it in the file details.txt
7. Using function sort(), sort the names upto no count
8. After sorting print the student details and same time write it in the file
9. Stop

RESULT

Thus the C program to accept a list of N numbers of students with their names and roll numbers are arranged in an alphabetical order and successfully written to the file using user defined data type using structure.

EX. No. : 17 BASIC PROGRAMS ON POINTERS

AIM

To write a C program to demo on Pointer variables.

ALGORITHM

1. Start
2. Initialize the variables.
3. Assign the address for dereferencing the variable to pointer variable after pointer declaration.
4. Print out the values based on address referencing and dereferencing operator.
5. Stop

RESULT

Thus the C program to perform basic understanding of pointers was compiled and executed successfully.

EX. No. : 18 DISPLAY SUM OF ARRAY USING POINTERS

AIM

To write a C program to Display sum of array using pointers

ALGORITHM

1. Start
2. Read n elements from the user and store it in the array.
3. store the address of the array into the pointer.
4. Set loop i=0 to i=n
5. Fetch the value from the location pointer by pointer variable.
6. Using De-referencing pointer can get the value at address.
7. Perform the addition
8. Repeat until n
9. Print the sum value
10. Stop

RESULT

Thus the C program to compute sum of array using pointers was compiled and executed successfully.

EX. No. : 19 SORTING OF NAMES USING ARRAY OF POINTERS

AIM

To write a C program to sort the given names using array of pointers.

ALGORITHM

1. Start
2. Read n for number of names to be sorted
3. Read the names
4. Set loop i=0 to n and j=1 to n
5. Compare the first name with second name if is greater than 0
6. Interchange the first name and the second name using third variable
7. Repeat until all the names are sorted
8. Print the sorted names
9. Stop

RESULT

Thus the C program to sort the given names using array of pointers was compiled and executed successfully.

EX. No. : 20 USING POINTERS COMPUTE MEAN AND STANDARD DEVIATION

AIM

To write a C program to compute mean and standard deviation

ALGORITHM

1. Declare an integer array , x[50];
2. Initialize the temporary variables to compute the mean value and standard deviation values.
3. read the input array.
4. compute the mean and standard deviation on the stored values using * operators.
5. Print the computed values, mean, standard deviation.
6. Stop the execution of the program.

RESULT

Thus the C program to compute mean and standard deviation was compiled and executed successfully.