BEHARA COLLEGE OF ENGINEERING & TECHNOLOGY

88th ward, G.V.M.C, NARAVA, VISAKHAPATNAM

COMPUTER PROGRAMMING LABORATORY MANUAL

LAB CODE : R23ES07 SCHEME:R23

(Common to All branches of Engineering)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

88th ward, G.V.M.C, NARAVA, VISAKHAPATNAM

VISION

To Excel in the emerging areas of Computer Science and Engineering by imparting quality education , relevant practices and inculcating human values to transform the students as potential resources to contribute innovatively to meet industrial needs and social expectations.

MISSION

M1: To provide strong fundamentals and value - based technical education for Computer Science applications through effective teaching learning methodologies.

M2: To transform lives of the students by nurturing ethical values, creativity and novelty to become Entrepreneurs and establish start-ups.

M3: To Impart high quality experiential learning to get expertise in modern software tools and to cater to the real time requirements of the industry

M4: To provide a conducive environment for faculty to engage in and train students in progressive and convergent research themes through collaborative linkages with industry and academia by establishing Centres of Excellence.

M5: To inculcate problem solving and team building skills and promote lifelong learning with a sense of societal and ethical responsibilities.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

PO1.ENGINEERING KNOWLEDGE: Apply the knowledge of mathematics, science, engineeringfundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2.PROBLEM ANALYSIS: Identify, formulate, review research literature, and analyze complexengineering problems reaching substantiated conclusions using first principles of mathematics,natural sciences, and engineering sciences.

PO3.DESIGN/DEVELOPMENT OF SOLUTIONS: Design solutions for complex engineering problems anddesign system components or processes that meet the specified needs with appropriateconsideration for the public health and safety, and the cultural, societal, and environmentalconsiderations.

PO4.CONDUCT INVESTIGATIONS OF COMPLEX PROBLEMS: Use research-based knowledge and researchmethods including design of experiments, analysis and interpretation of data, and synthesis of theinformation to provide valid conclusions.

PO5.MODERN TOOL USAGE: Create, select, and apply appropriate techniques, resources, and modernengineering and IT tools including prediction and modelling to complex engineering activitieswith an understanding of the limitations.

PO6. THE ENGINEER AND SOCIETY: Apply reasoning informed by the contextual knowledge to assesssocietal, health, safety, legal and cultural issues and the consequent responsibilities relevant tothe professional engineering practice.

PO7.ENVIRONMENT AND SUSTAINABILITY: Understand the impact of the professional engineeringsolutions in societal and environmental contexts, and demonstrate the knowledge of, and needfor sustainable development.

PO8.ETHICS: Apply ethical principles and commit to professional ethics and responsibilities andnorms of the engineering practice.

PO9.INDIVIDUAL AND TEAM WORK: Function effectively as an individual, and as a member or leaderin diverse teams, and in multidisciplinary settings.

PO10.COMMUNICATION: Communicate effectively on complex engineering activities with theengineering community and with society at large, such as, being able to comprehend and writeeffective reports and design documentation, make effective presentations, and give and receiveclear instructions.

PO11.PROJECT MANAGEMENT AND FINANCE: Demonstrate knowledge and understanding of theengineering and management principles and apply these to one's own work, as a member andleader in a team, to manage projects and in multidisciplinary environments.

PO12.LIFE-LONG LEARNING: Recognize the need for, and have the preparation and ability to engage inindependent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

The Computer Science & Engineering graduate will

PSO-1	**Professional Skills:**	The Computer Engineering Graduates are able to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics and networking for efficient design of computer based systems of varying complexity.
PSO-2	**Successful Career and Entrepreneurship:**	The Computer Engineering Graduates are able to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur and enthusiastic for higher studies/employability in the field of Computer Science & Engineering.

Program Educational Objectives (PEOs)

The Programme Educational Objectives of the B.Tech in Computer Science & Engineering programme are given below and are numbered from PEO1 to PEO4.

PEO-1	To provide the graduates with solid foundation in computer science and engineering along with fundamentals of Mathematics and Sciences in formulating, analyzing, designing, modelling, programming and implementation with global competence and helps the graduates for life-long learning.
PEO-2	To Promote collaborative learning and team work spirit through multi - disciplinary projects and diverse professional activities and prepare graduates with recent technological developments related to core subjects like programming, databases, design of compilers and Network Security aspects and future technologies so as to contribute effectively for Research & Development by participating in professional activities like publishing and seeking copy rights.
PEO-3	To train graduates to choose a decent career option either in high degree of employability /Entrepreneur or, in higher education by empowering students with sustainable progress, ability to handle critical situations and training to excel in competitive examinations.

PEO-4	To exhibit professionalism, ethical attitude, communication, managerial skills, teamwork, and social responsibility in their profession and adapt to current trends by engaging in continuous learning.

GENERAL INSTRUCTIONS

Do's

1. Please leave footwear outside the laboratory at the designated place.
2. Please keep your belongings such as bags in the designated place.
3. Students must present a valid ID card before entering the computer lab.
4. Maintain Discipline in the laboratory.
5. Work on the designated computers only.
6. Do keep your computer and workspace clean and organized.
7. Do save your work frequently to avoid data loss.
8. Do follow ethical guidelines and respect copyright when using digital resources.
9. Students must inform the in-charge Lecturer of any observed hardware or software failures.
10. Turn off the respective systems and arrange the chairs before you leaving the laboratory.

Don'ts

1. Don't touch the computer with a pen or pencil.
2. Don't eat or drink near your computer.
3. Don't touch wires or cables when the computer is on.
4. Do not install, uninstall or alter any software on the computer.
5. Students are not allowed to use personal Pen Drives, CDs, DVDs etc., in a Computer Lab. Only prescribed official Pen Drives, CDs, DVDs etc. will be used in the Computer Lab to avoid VIRUS in Computers.
6. The use of cell phones is prohibited in the computer lab.
7. Don't forget to turn off your computer before leaving.

COMPUTER PROGRAMMING

OBJECTIVES AND OUTCOMES

(Common to all branches of Engineering)

Course Objectives:

The course aims to give students hands – on experience and train them on the concepts of the C-programming language.

Course Outcomes:

CO1: Read, understand, and trace the execution of programs written in C language.

CO2: Select the right control structure for solving the problem.

CO3: Develop C programs which utilize memory efficiently using programming constructs like pointers.

CO4: Develop, Debug and Execute programs to demonstrate the applications of arrays, functions, basic concepts of pointers in C.

BEHARA
COLLEGE OF ENGINEERING AND TECHNOLOGY
Approved by AICTE NEW DELHI & Affiliated to JNTU-GV, Vizianagaram
88th Division, Narava, GVMC, Visakhapatnam, Andhra Pradesh 530027, India
e-mail: beharaenggclg@gmail.com www.bcet.in

DEPARTMENT OF CSE

INDEX

UNIT NO	WEEK NO	NAME OF THE PROGRAM	PAGE NO
UNIT-1	WEEK-1	Lab1: Familiarization with programming environment i) Basic Linux environment and its editors like Vi, Vim & Emacs etc. ii) Exposure to Turbo C, gcc iii) Writing simple programs using printf(), scanf()	1-11
	WEEK-2	Lab 2: Converting algorithms/flow charts into C Source code. Developing the algorithms/flowcharts for the following sample programs i) Sum and average of 3 numbers ii) Conversion of Fahrenheit to Celsius and vice versa iii) Simple interest calculation	12-15
	WEEK-3	Lab 3: Simple computational problems using arithmetic expressions. i) Finding the square root of a given number ii) Finding compound interest iii) Area of a triangle using heron's formulae iv) Distance travelled by an object	16-19
UNIT-2	WEEK-4	Lab4: Simple computational problems using the operator' precedence and associativity i) Evaluate the following expressions. a. A+B*C+(D*E) + F*G b. A/B*C-B+A*D/3 c. A+++B---A d. J= (i++) + (++i) ii) Find the maximum of three numbers using conditional operator iii) Take marks of 5 subjects in integers, and find the total, average in float	20-23
	WEEK-5	Lab 5: Problems involving if-then-else structures. i) Write a C program to find the max and min of four numbers using if-else. ii) Write a C program to generate electricity bill. iii) Find the roots of the quadratic equation. iv) Write a C program to simulate a calculator using switch case. v) Write a C program to find the given year is a leap year or not.	24-29

	WEEK-6	Lab 6: Iterative problems e.g., the sum of series i) Find the factorial of given number using any loop. ii) Find the given number is a prime or not. iii) Compute sine and cos series iv) Checking a number palindrome v) Construct a pyramid of numbers.	30-36
UNIT-3	**WEEK-7**	Lab 7:1D Array manipulation, linear search i) Find the min and max of a 1-D integer array. ii) Perform linear search on1D array. iii) The reverse of a 1D integer array iv) Find 2’s complement of the given binary number. v) Eliminate duplicate elements in an array.	37-47
	WEEK-8	Lab 8: Matrix problems, String operations, Bubble sort i) Addition of two matrices ii) Multiplication two matrices iii) Sort array elements using bubble sort iv) Concatenate two strings without built-in functions v) Reverse a string using built-in and without built-in string functions	48-56
UNIT-4	**WEEK-9**	Lab 9: Pointers and structures, memory dereference. i) Write a C program to find the sum of a 1D array using malloc() ii) Write a C program to find the total, average of n students using structures iii) Enter n students data using calloc() and display failed students list iv) Read student name and marks from the command line and display the student details along with the total. v) Write a C program to implement realloc()	57-65
	WEEK-10	Lab10 : Bitfields, linked lists Read and print a date using dd/mm/yyyy format using bit-fields and differentiate the same without using bit- fields i) Create and display a singly linked list using self-referential structure. ii) Demonstrate the differences between structures and unions using a C program. iii) Write a C program to shift/rotate using bitfields. iv) Write a C program to copy one structure variable to another structure of the same type.	66-75

UNIT-5	**WEEK-11**	Lab 11: Simple functions using call by value, solving differential equations using Eulers theorem. i) Write a C function to calculate NCR value. ii) Write a C function to find the length of a string. iii) Write a C function to transpose of a matrix. iv) Write a C function to demonstrate numerical integration of differential equations using Euler's method	76-82
	WEEK-12	Lab 12: Recursive functions i) Write a recursive function to generate Fibonacci series. ii) Write a recursive function to find the lcm of two numbers. iii) Write a recursive function to find the factorial of a number. iv) Write a C Program to implement Ackermann function using recursion. v) Write a recursive function to find the sum of series.	83-89
	WEEK-13	Lab 13: Simple functions using Call by reference, Dangling pointers. i) Write a C program to swap two numbers using call by reference. ii) Demonstrate Dangling pointer problem using a C program. iii) Write a C program to copy one string into another using pointer. iv) Write a C program to find no of lowercase, uppercase, digits and other characters using pointers.	90-96
	WEEK-14	Lab 14: File operations i) Write a C program to write and read text into a file. ii) Write a C program to write and read text into a binary file using fread() and fwrite() iii) Copy the contents of one file to another file. iv) Write a C program to merge two files into the third file using command-line arguments. v) Find no. of lines, words and characters in a file vi) Write a C program to print last n characters of a given file.	97-107

UNIT I
WEEK 1

Objective: Getting familiar with the programming environment on the computer and writing the first program.
Suggested Experiments/Activities:
Tutorial 1: Problem-solving using Computers.

Lab1: Familiarization with programming environment

i) Basic Linux environment and its editors like Vi, Vim & Emacs etc.

Linux Text Editors

Linux text editors can be used for editing text files, writing codes, updating user instruction files, and more. A Linux system supports multiple text editors. There are two types of text editors in Linux, which are given below:

- Command-line text editors such as Vi, nano, pico, and more.
- GUI text editors such as gedit (for Gnome), Kwrite, and more.

A text editor plays an important role while coding. So, it is important to select the best text editor. A text editor should not only be simple but also functional and should be good to work with.

A text editor with IDE features is considered as a good text editor.

1.Vi/VIM editor

Vim editor is one of the most used and powerful command-line based editor of the Linux system. By default, it is supported by most Linux distros. It has enhanced functionalities of the old Unix Vi editor. It is a user-friendly editor and provides the same environment for all the Linux distros. It is also termed as programmer's editor because most programmers prefer Vi editor.

Vi editor has some special features such as Vi modes and syntax highlighting that makes it powerful than other text editors. Generally, it has two modes:

Command Mode: The command mode allows us to perform actions on files. By default, it starts in command mode. In this mode, all types of words are considered as commands. We can execute commands in this mode.

Insert Mode: The insert mode allows to insert text on files. To switch from command mode to insert mode, press the Esc key to exit from active mode and 'i' key.

The command to work with visual editor is

```
$ vi filename.c
```

It can function in different modes.

- Insert mode (I/i or A/a)
- Execute command mode(shift :)
- Escape mode(escape)

Insert mode: In this mode we can enter the data or modify content of a file . By pressing (I/I or A/a) from escape mode we can come to insert mode.

Execute command mode: This is the mode from where we can apply the command mode commands such as

: wq save and quit

: q! quit without saving

In addition to these commands it can be applied on selected test such as copying, deleting lines etc. From escape mode by pressing we can enter into execute command mode.

Escape mode: By default immediately after opening a file through vi editor the file will be in escape mode. We can switch from one mode to another mode.

2. Emacs editor

emacs is a screen editor. Unlike vi, emacs is not an insertion mode editor, meaning that any character typed in emacs is automatically inserted into the file, unless it includes a command prefix.

Commands in emacs are either control characters (hold down the <Ctrl> key while typing another character) or are prefixed by one of a set of reserved characters: <Esc> or <Ctrl>-X. The <Esc> key can be typed by itself (because it really is a character) and then followed by another character; the <Ctrl> key must be held down while the next character is being typed. The conventions for describing these characters (since it takes too long to type out the whole thing) are ESC means <Esc> and C- means <Ctrl>. One other distinction between emacs and vi is that emacs allows you to edit several files at once. The window for emacs can be divided into several windows, each of which contains a view into a buffer. Each buffer typically corresponds to a different file. Many of the commands listed below are for reading files into new buffers and moving between buffers.

To use emacs on a file, type $emacs filename

ii) Exposure to Turbo C, gcc

The most popular IDE for C language for beginners is Turbo C/C++. In this tutorial, we will cover the installation steps for Turbo C and will also share a few other IDE(integrated development environment) that you can use for C programming.

IDE

An IDE or **Integrated Development environment** is a software, which has a code editor, along with the compiler of the programming language too, hence you do not need to install anything else. An IDE has all the required features to manage your source code, like File handling, Edit options, etc. while it also has the support for different programming language environments too, for which it has in-built compilers/interpreters installed.

Many IDEs allow installing 3rd party plugins to further enhance the features of the IDE.

Turbo C is an IDE used for writing code in the C language. And you can also, compile and run your code using Turbo C itself. **Turbo C++ is the upgraded version** of Turbo C, which can also be used for C language programming.

The C++ compiler supports the C language syntax as the C++ language is built on top of the C language.

There are other IDEs too that you can use for C programming, like:

1. Dev-C++ (Download from here: Download Dev-C++)
2. Eclipse (Download Eclipse IDE)

Other IDEs for C Programming

You can use the **Dev-C++** IDE or the **Eclipse IDE** for C programming. We have shared the download link for both in the introduction of IDE.

If you want a more modern C/C++ IDE for writing C programs then you should try Dev-C++ as it is not as complex as the Eclipse IDE, while has many features which the basic Turbo C/C++ IDE lacks.

Here are some of the most popular options:

GCC (GNU Compiler Collection)

GCC is a free, open-source compiler collection that includes a C compiler. On Windows, you can install GCC through the MinGW (Minimalist GNU for Windows) project. On macOS and Linux, GCC is often included as part of the operating system's standard package repository.

Visual Studio

Microsoft Visual Studio is a commercial integrated development environment (IDE) that includes a C compiler. Visual Studio is available for Windows only.

Code::Blocks
Code::Blocks is a free, open-source IDE that includes a C compiler. It's available for Windows, macOS, and Linux.
Once you install a C compiler and development environment, you can write your C program in a text editor and then compile and run it from the command line or within the IDE.

The process of writing, compiling, and running C programs can vary depending on the development environment you're using, so you may need to refer to the documentation for the specific tool you've chosen.
Best C Programming Apps or Software for Windows 10 PC
Here are some of the best C programming software for Windows 10:

Microsoft Visual Studio
An integrated development environment (IDE) from Microsoft, it supports a variety of programming languages, including C, C++, and C++/CLI.

Code::Blocks
A free, open-source, and cross-platform IDE for C, C++, and Fortran.

Dev-C++
A free, open-source IDE for C and C++ programming.

Turbo C++
An older but still widely used IDE for C and C++ programming, it was initially released in 1990.

Eclipse CDT (C/C++ Development Tooling)
A free, open-source, and multi-language IDE that provides a C/C++ development environment.

Clion
A commercial IDE for C and C++ programming developed by JetBrains.

Turbo C
It is one of the oldest IDE and C language compilers, which was most popular in the 1980s and early 1990s.

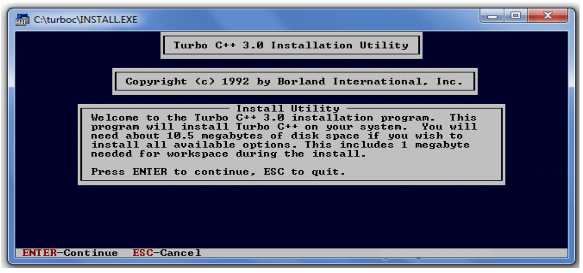
Installation
Step 1. Download Turbo C++ software
Turbo C++ can be downloaded from many sites on the internet.

Step 2. Now, Create turbo C directory in the C drive of windows and extract the tc3.zip

In this step, users have to create a new directory of turbo C inside the c: drive. After creating the directory, extract the tc3.zip file in c:\truboc directory.

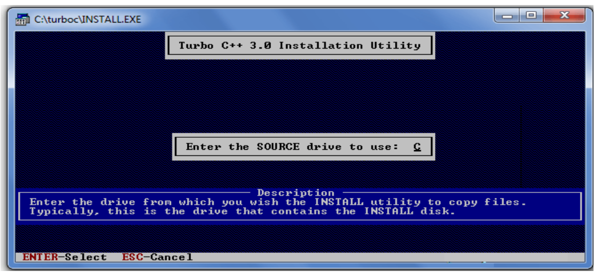
Step 3. Double click on the install.exe file and follow these steps
Click on the install icon that is located inside the c:\turboc

Step 4: To install c Press Enter



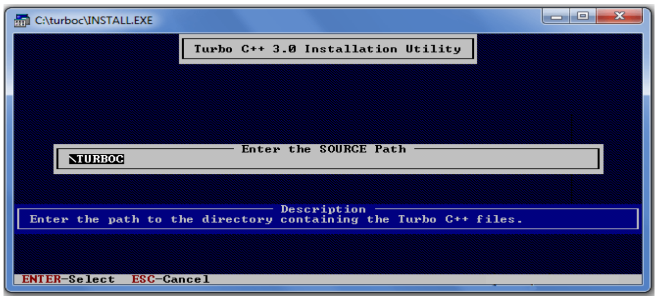
```
C:\turboc\INSTALL.EXE

Turbo C++ 3.0 Installation Utility

Copyright (c) 1992 by Borland International, Inc.

Install Utility
Welcome to the Turbo C++ 3.0 installation program.  This
program will install Turbo C++ on your system.  You will
need about 10.5 megabytes of disk space if you wish to
install all available options. This includes 1 megabyte
needed for workspace during the install.

Press ENTER to continue, ESC to quit.

ENTER-Continue  ESC-Cancel
```

Step 5: Select Your Drive in which you want to install "Press C"



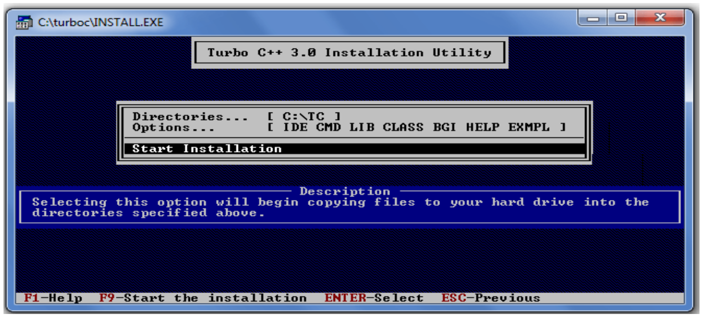
```
C:\turboc\INSTALL.EXE

Turbo C++ 3.0 Installation Utility

Enter the SOURCE drive to use:  C

Description
Enter the drive from which you wish the INSTALL utility to copy files.
Typically, this is the drive that contains the INSTALL disk.

ENTER-Select  ESC-Cancel
```

Step 6: Press Enter(it will get some required files from c/tubro c directory



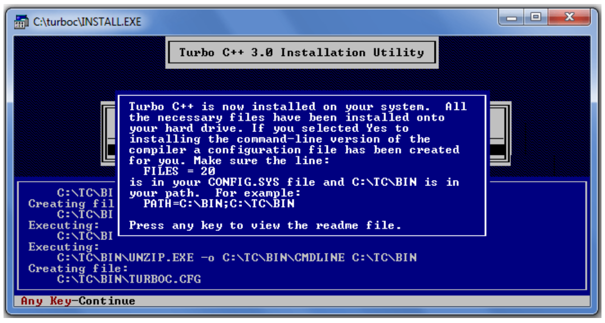
```
C:\turboc\INSTALL.EXE

Turbo C++ 3.0 Installation Utility

Enter the SOURCE Path
\TURBOC

Description
Enter the path to the directory containing the Turbo C++ files.

ENTER-Select  ESC-Cancel
```

Step 7: Select Down array key and Press Enter to start installation



```
C:\turboc\INSTALL.EXE

Turbo C++ 3.0 Installation Utility

Directories...   [ C:\TC ]
Options...       [ IDE CMD LIB CLASS BGI HELP EXMPL ]

Start Installation

Description
Selecting this option will begin copying files to your hard drive into the
directories specified above.

F1-Help  F9-Start the installation  ENTER-Select  ESC-Previous
```

Step 8: C installed Successfully



```
Turbo C++ is now installed on your system.  All
the necessary files have been installed onto
your hard drive. If you selected Yes to
installing the command-line version of the
compiler a configuration file has been created
for you. Make sure the line:
  FILES = 20
is in your CONFIG.SYS file and C:\TC\BIN is in
your path.  For example:
  PATH=C:\BIN;C:\TC\BIN

Press any key to view the readme file.
```

```
    C:\TC\BI
Creating fil
    C:\TC\BI
Executing:
    C:\TC\BI
Executing:
    C:\TC\BIN\UNZIP.EXE -o C:\TC\BIN\CMDLINE C:\TC\BIN
Creating file:
    C:\TC\BIN\TURBOC.CFG
Any Key-Continue
```

Step 9: Here "c:\TC\BIN" is located tc application click on it to Start and writing program

A Brief History and Introduction to GCC

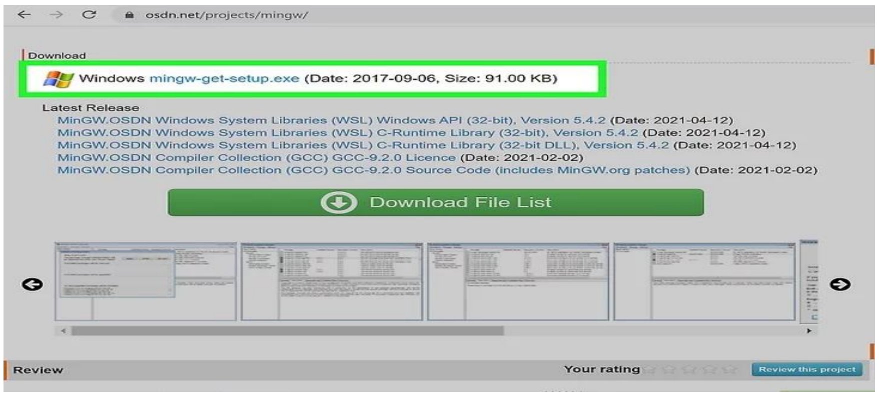
The original *GNU C Compiler* (GCC) is developed by Richard Stallman, the founder of the *GNU Project*. Richard Stallman founded the GNU project in 1984 to create a complete Unix-like operating system as free software, to promote freedom and cooperation among computer users and programmers.

GCC, formerly for "*GNU C Compiler*", has grown over times to support many languages such as C (gcc), C++ (g++), Objective-C, Objective-C++, Java (gcj), Fortran (gfortran), Ada (gnat), Go (gccgo), OpenMP, Cilk Plus, and OpenAcc. It is now referred to as "*GNU Compiler Collection*". The mother site for GCC is http://gcc.gnu.org/. The current version is GCC 7.3, released on 2018-01-25.

GCC is a key component of so-called "*GNU Toolchain*", for developing applications and writing operating systems. The GNU Toolchain includes:

1. GNU Compiler Collection (GCC): a compiler suite that supports many languages, such as C/C++ and Objective-C/C++.
2. GNU Make: an automation tool for compiling and building applications.
3. GNU Binutils: a suite of binary utility tools, including linker and assembler.
4. GNU Debugger (GDB).
5. GNU Autotools: A build system including Autoconf, Autoheader, Automake and Libtool.
6. GNU Bison: a parser generator (similar to lex and yacc).

Using MinGW for Windows

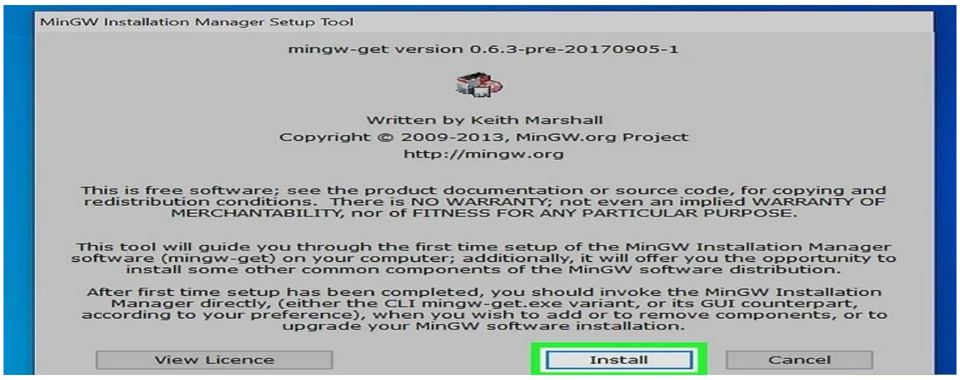


Download

Windows mingw-get-setup.exe (Date: 2017-09-06, Size: 91.00 KB)

Latest Release
- MinGW.OSDN Windows System Libraries (WSL) Windows API (32-bit), Version 5.4.2 (Date: 2021-04-12)
- MinGW.OSDN Windows System Libraries (WSL) C-Runtime Library (32-bit), Version 5.4.2 (Date: 2021-04-12)
- MinGW.OSDN Windows System Libraries (WSL) C-Runtime Library (32-bit DLL), Version 5.4.2 (Date: 2021-04-12)
- MinGW.OSDN Compiler Collection (GCC) GCC-9.2.0 Licence (Date: 2021-02-02)
- MinGW.OSDN Compiler Collection (GCC) GCC-9.2.0 Source Code (includes MinGW.org patches) (Date: 2021-02-02)

1. **Download Minimalist GNU for Windows (MinGW).** This is an easy-to-install version of GCC for Windows. You can download the installer from https://osdn.net/projects/mingw.
   - To get the latest version of the installer, click the **mingw-get-setup.exe** link.
   - If the download doesn't begin automatically, click **Save** or **Download** when prompted.
   - MinGW can only compile 32-bit versions of software. However, all 32-bit software compiled with MinGW will execute properly on a 64-bit system.[9]
   - If you need to compile 64-bit software, try MinGW-w64, a port of MinGW. You can download it from https://www.mingw-w64.org.[10] Alternatively, you can use Windows Subsystem for Linux.


MinGW Installation Manager Setup Tool

mingw-get version 0.6.3-pre-20170905-1

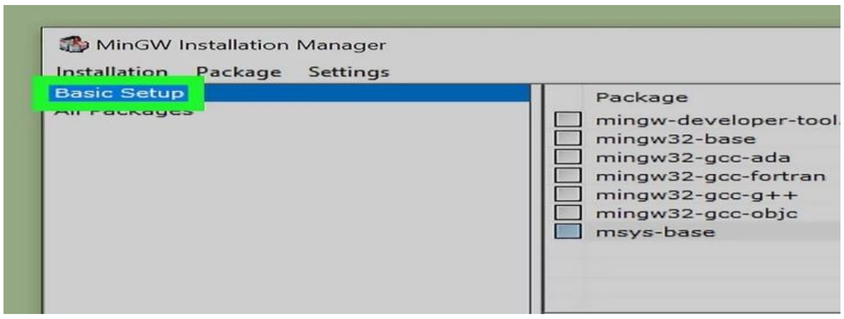
Written by Keith Marshall
Copyright © 2009-2013, MinGW.org Project
http://mingw.org

This is free software; see the product documentation or source code, for copying and redistribution conditions. There is NO WARRANTY; not even an implied WARRANTY OF MERCHANTABILITY, nor of FITNESS FOR ANY PARTICULAR PURPOSE.

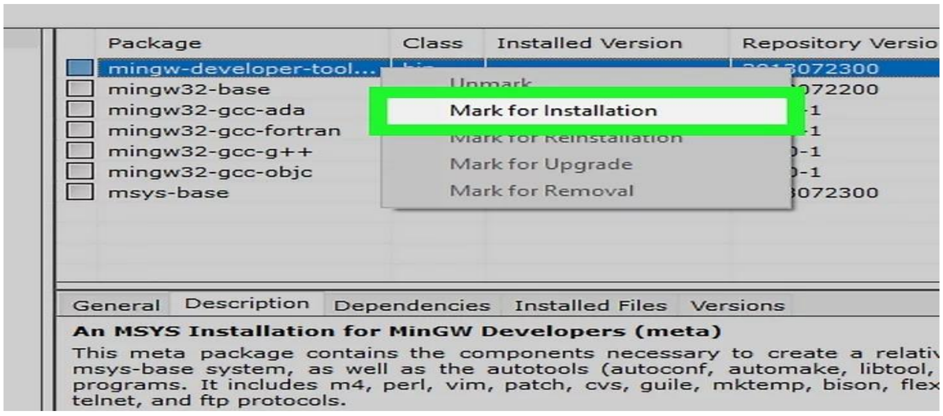
This tool will guide you through the first time setup of the MinGW Installation Manager software (mingw-get) on your computer; additionally, it will offer you the opportunity to install some other common components of the MinGW software distribution.

After first time setup has been completed, you should invoke the MinGW Installation Manager directly, (either the CLI mingw-get.exe variant, or its GUI counterpart, according to your preference), when you wish to add or to remove components, or to upgrade your MinGW software installation.

2. **Install MinGW.** Once you've downloaded the installer, double-click it to install MinGW.
   - MinGW recommends using the default installation folder (C:\MinGW). If you must change the folder, don't use a folder with spaces in the name (e.g. "Program Files").

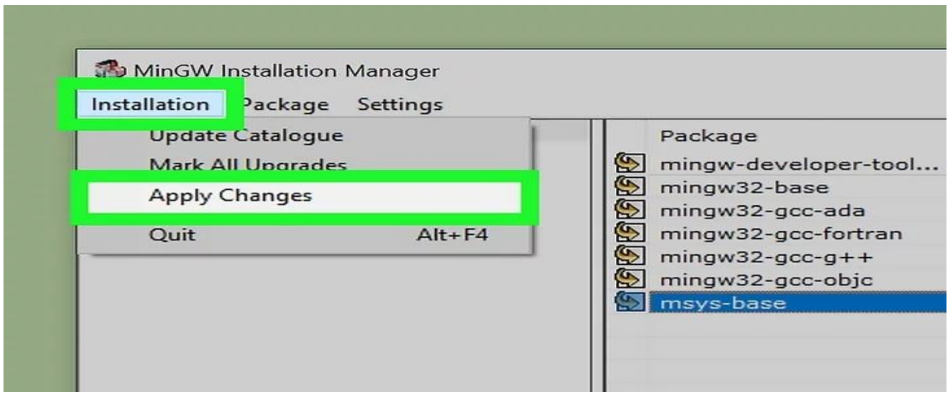


3. **Select Basic Setup to view the basic C compiling tools.** If you want more options, you can select **All Packages** to see all available libraries and build tools.

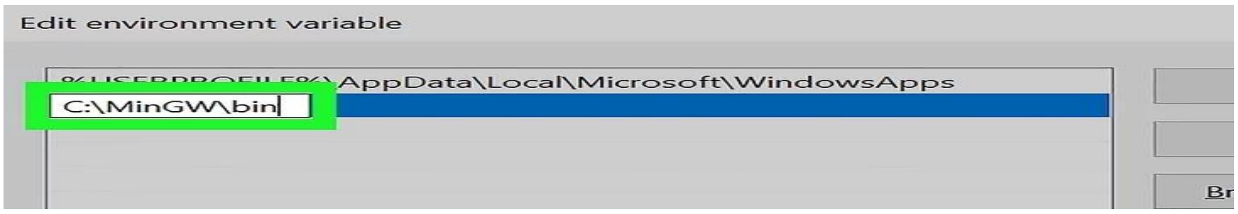


An MSYS Installation for MinGW Developers (meta)
This meta package contains the components necessary to create a relativ
msys-base system, as well as the autotools (autoconf, automake, libtool,
programs. It includes m4, perl, vim, patch, cvs, guile, mktemp, bison, flex
telnet, and ftp protocols.

4. **Right-click each package and click Mark for Installation.** The Basic Setup has about 7 packages listed in the box at the top. Right-click each one of them (or just the ones you want) and click **Mark for Installation**. This adds an icon with an arrow next to each one and marks it for installation.
   - At the very least, you will want to install **MinGW32-base** and **MinGW32-gcc-g++**. If you need to compile code written in objective C, you should also mark **MinGW32-gcc-objc** for installation.

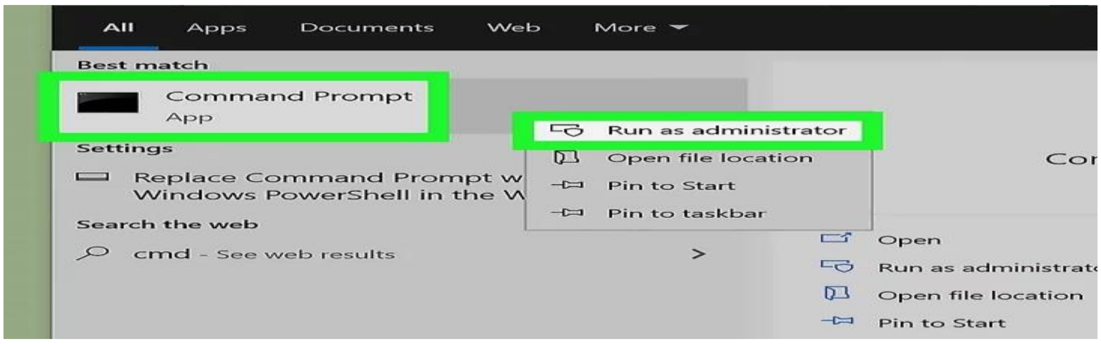


5. **Install the selected packages.** It may take your computer several minutes to install all packages. Use the following steps to install the packages that are marked for installation.
   - Click the **Installation** menu in the upper-left corner.
   - Click **Apply Changes**.
   - Click **Apply**.
   - Click **Close** once the installation is done.

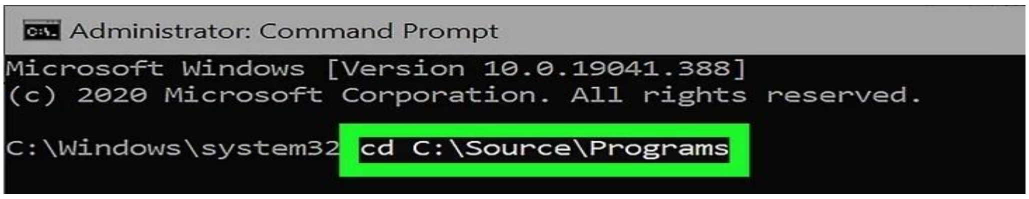

Edit environment variable
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\MinGW\bin

6. **Add the path to MinGW to system environment variables.** Use the following steps to add the path to MinGW to system environment variables:
   - Type environment in the search bar next to the Start menu.

- Click **Edit the system environment variables** in the search results.
- Click **Environment Variables…**
- Select the **Path** variable in the "System variables" section.
- Click **Edit** beneath the top box (under "User Variables")
- Click **New**.
- Type `C:\MinGW\bin` in the new space. Note that if you installed MinGW to a different directory, enter `C:\path-to-that-directory\bin`.
- Click **OK**, and then **OK** again. Click the one remaining **OK** button to close the window.

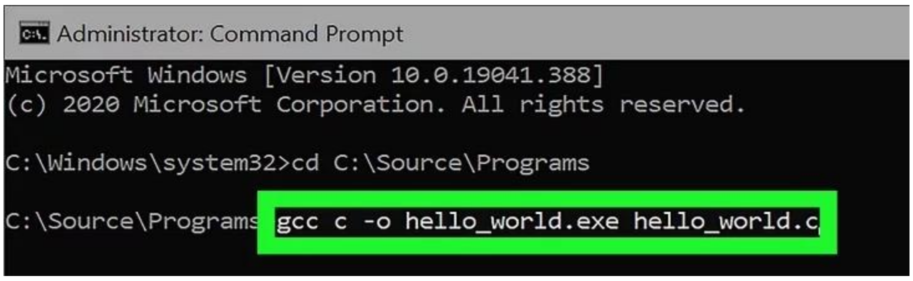


7. **Open the command prompt as an administrator.** You must be signed in to a Windows account with administrative privileges to open the Command Prompt as an administrator. Use the following steps to open the Command Prompt as an administrator:
   - Type `cmd` in the search bar next to the Start menu.
   - Right-click **Command Prompt** in the search results, then select **Run as Administrator**.
   - Click **Yes** to allow changes.



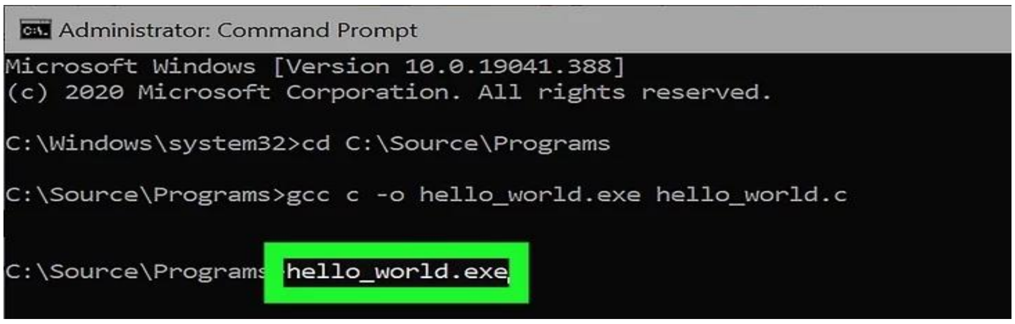
```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19041.388]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32 cd C:\Source\Programs
```

8. **Go to the folder where your source code is saved.** For example, if your source code file called helloworld.c is located in C:\Source\Programs, type `cd C:\Source\Programs` and press **Enter**.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19041.388]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Source\Programs

C:\Source\Programs gcc c -o hello_world.exe hello_world.c
```

9. **Type `gcc c –o [program_name].exe [program_name].c` and press ↲ Enter.** Replace "[program_name]" with the name of your source code and application. Once the program is compiled, you'll return to the command prompt without errors.
   - Any coding errors that appear must be corrected before the program will compile.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19041.388]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Source\Programs

C:\Source\Programs>gcc c -o hello_world.exe hello_world.c


C:\Source\Programs>hello_world.exe
```

10. **Type the name of your program to run it.** If it's called hello_world.exe, type that in the command prompt and press **Enter** to start your program.
    - If you receive an "Access is denied" or "Permission denied" error message when compiling a program or running the output executable file, check the folder permissions and make sure you have full read/write access to the folder that contains the source code. If that doesn't work, try temporarily disabling your virus software.

iii) Writing simple programs using printf(), scanf()

Algorithm

1. Start
2. Initialize variables a.
3. Read a numbers into a.
4. Print the entered value.
5. Stop

Source Code

```
#include <stdio.h>
int main()
{
int a;
printf("Enter an integer : \n");
scanf("%d", &a);
printf("Integer that you have entered is %d\n", a);
return 0;
}
```

Output

```
Enter an integer : 45
Integer that you have entered is 45
```

WEEK 2

Objective: Getting familiar with how to formally describe a solution to a problem in a series of finite steps both using textual notation and graphic notation.
Suggested Experiments /Activities:
Tutorial 2: Problem-solving using Algorithms and Flow charts.

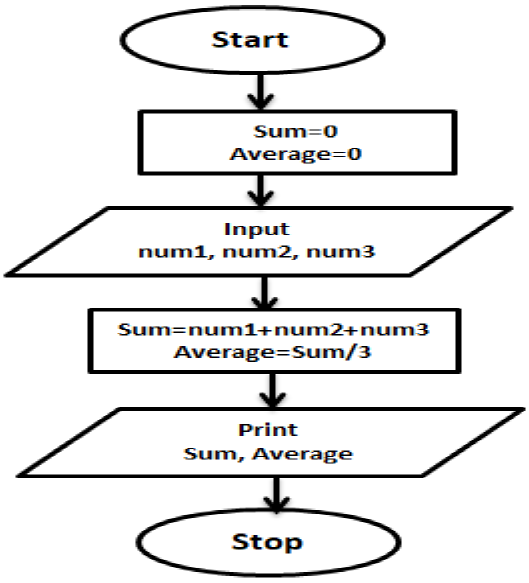
Lab 1: Converting algorithms/flow charts into C Source code. Developing the algorithms/flowcharts for the following sample programs

i) Sum and average of 3 numbers

Algorithm

1. Start
2. Initialize variables for sum and average to zero. (Sum=0 , Average=0)
3. Read three numbers using variable num1,num2 and num3.
4. Find the sum of three numbers where sum=num1+num2+num3
5. Find the average of three numbers where Average= sum / 3
6. Print sum, average.
7. Stop

Flowchart



ii) Conversion of Fahrenheit to Celsius and vice versa

Conversion table

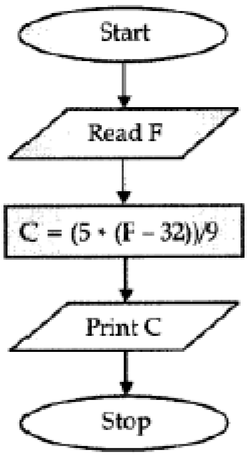
From	To	Formula
Fahrenheit (F)	Celsius (C or o)	(F - 32) * 5/9
Celsius (C or o)	Fahrenheit (F)	(C * 9/5) + 32

Conversion From Fahrenheit to Celsius

Algorithm

1. Start.
2. Read F.
3. C=(5(F-32))/9.
4. Print C.
5. Stop.

Flowchart



Source Code

```
#include<stdio.h>
void main()
{
 float celsius, fahrenheit;
  printf("\n Enter Temp in Fahrenheit : ");
 scanf("%f", &fahrenheit);
 celsius = (fahrenheit-32) / 1.8;
 printf("\n Temperature in Celsius : %.2f ", celsius);
}
```

Output

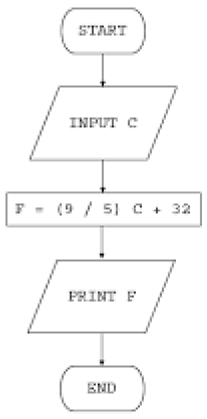
```
Enter Temp in Fahrenheit : 56
Temperature in Celsius : 13.33
```

Conversion From Celsius to Fahrenheit

Algorithm

1. Start.
2. Read Celsius.
3. fahrenheit = (1.8 * celsius) + 32.
4. Print C.
5. Stop.

Flowchart



Source Code

```
#include<stdio.h>
void main()
{
 float celsius, fahrenheit;
 printf("\n Enter Temp in Celsius : ");
 scanf("%f", &celsius);
 fahrenheit = (1.8 * celsius) + 32;
 printf("\n Temperature in Fahrenheit : %.2f ", fahrenheit);
}
```

Output

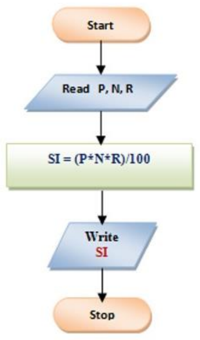
```
Enter Temp in Celsius : 13
 Temperature in Fahrenheit : 55.40
```

iii) Simple interest calculation

Algorithm

1. Enter the principal amount, rate of interest, and time period.
2. Calculate the simple interest using the formula **SI = (P * R * T)/100**.
3. Print the Simple Interest.

Flowchart



Source Code

```
#include<stdio.h>
int main ()
{
  float principal, rate, time;
  printf ("Enter the principal amount: ");
  scanf ("%f", &principal);
  printf ("Enter the rate of interest: ");
  scanf ("%f", &rate);
  printf ("Enter the time: ");
  scanf ("%f", &time);
  float simple_interest, amount;
  simple_interest = ((principal * rate * time) / 100);
  amount = simple_interest + principal;
  printf ("Simple Interest = %f \nAmount = %f", simple_interest, amount);
  return 0;
}
```

Output

```
Enter the principal amount: 20000
Enter the rate of interest: 2
Enter the time: 4
Simple Interest = 1600.000000
Amount = 21600.000000
```

WEEK 3

Objective: Learn how to define variables with the desired data-type, initialize them with appropriate values and how arithmetic operators can be used with variables and constants.

Suggested Experiments/Activities:

Tutorial 3: Variable types and type conversions:

Lab 3: Simple computational problems using arithmetic expressions.

i) Finding the square root of a given number

Algorithm

1. Declare an integer variable, as result.
2. Use the sqrt() function to pass the result variable as an argument to find the square root.
3. Print the result.
4. Exit or terminate the program.

Source Code

```
#include <stdio.h>
#include <math.h>
int main()
{
  double n, result;
  printf("Enter a number to calculate its square root\n");
  scanf("%lf", &n);
  result = sqrt(n);
  printf("Square root of %.2lf = %.2lf\n", n, result);
  return 0;
}
```

Output

```
Enter a number to calculate its square root 12
Square root of 12.00=3.46
```

ii) Finding compound interest

Algorithm

1. Input principle amount. Store it in some variable say principle_amount.
2. Input time in some variable say time_duration.
3. Input rate in some variable say rate.
4. Calculate compound interest using formula,
   amount = principle_amount*((pow((1+rate/100),time_duration)));
   compound_interest = amount - principle_amount;
5. Finally, print the resultant value of compound_interest.

Source Code

```
#include <stdio.h>
#include <math.h>
int main()
 {
   double principle_amount, rate, time_duration, amount, compound_interest;
   printf("Enter the principle amount: ");
   scanf("%lf",&principle_amount);
   printf("Enter the rate of interest: ");
   scanf("%lf",&rate);
   printf("Enter the time duration: ");
   scanf("%lf",&time_duration);
   amount = principle_amount*((pow((1+rate/100),time_duration)));
   compound_interest = amount - principle_amount;
   printf(" The compound interest is: %lf",compound_interest);
   return 0;
  }
```

Output

```
Enter the principle amount: 10000
Enter the rate of interest: 5
Enter the time duration: 2
 The compound interest is: 1025.000000
```

iii) Area of a triangle using heron's formulae

Algorithm

```
1. Start
2. Input a,b,c
3. Calculate s = (a + b + c ) / 2
4. Calculate area = sqrt( s*(s-a)*(s-b)*(s-c))
5. print "Area of Triangle=", area
6. End
```

Source Code

```
# include<stdio.h>
# include <math.h>
int main()
{
   float a,b,c,s,area;
   printf("Enter the length of three sides of a triangle:\n");
   scanf("%f%f%f", &a,&b,&c);
   s = (a+b+c)/2;
   printf("The value of S is %.2f\n",s);
   area = sqrt(s*(s-a) * (s-b) * (s-c));
   printf("The area of the triangle is:%f\n", area);
   return 0;
}
```

Output

```
Enter the length of three sides of a triangle:
5
6
7
The value of S is 9.00
The area of the triangle is:14.696939
```

iv) Distance travelled by an object

Algorithm

```
1. Start
2. Input u,a,t
3. Calculate s=u*t+0.5*(a*t*t);
4. Print s.
5. End
```

Source Code

```c
#include<stdio.h>
#include<math.h>
int main()
{
  int u,a,t;
  float s;
  printf("enter initial velocity(u),time(t),acceleration(a)");
  scanf ("%d%d%d",&u,&t,&a);
  s=u*t+0.5*(a*t*t);
  printf("s=%f",s);
}
```

Output

```
enter initial velocity(u),time(t),acceleration(a)8
6
7
s=174.000000
```

UNIT II
WEEK 4

Objective: Explore the full scope of expressions, type-compatibility of variables & constants and operators used in the expression and how operator precedence works.

Suggested Experiments/Activities:

Tutorial4: Operators and the precedence and as associativity:

Lab4: Simple computational problems using the operator' precedence and associativity

i) Evaluate the following expressions.

a. A+B*C+(D*E) + F*G **b. A/B*C-B+A*D/3**

c. A+++B---A **d. J= (i++) + (++i)**

Algorithm

1. Declare A,B,C,D,E,F,G,i,j.
2. Read A,B,C,D,E,F,G,i,j Values.
3. Evaluate the below expressions
4. result_a = A + B * C + (D * E) + F * G.
5. result_b = A / B * C - B + A * D / 3.
6. result_c = A++ + ++B - --A.
7. J = (i++) + (++i).
8. Print result_a,result_b,result_c,j
9. Exit or terminate the program.

Source Code

```
#include <stdio.h>
int main()
{
int A = 5, B = 10, C = 2, D = 7, E = 3, F = 4, G = 6;
int i = 5, j , result_a, result_b, result_c;
result_a = A + B * C + (D * E) + F * G;
printf("a. Result of expression A+B*C+(D*E) + F*G is: %d\n", result_a);
result_b = A / B * C - B + A * D / 3;
printf("b. Result of expression A/B*C-B+A*D/3 is: %d\n", result_b);
result_c = A++ + ++B - --A;
printf("c. Result of expression A+++B---A is: %d\n", result_c);
j = (i++) + (++i);
printf("d. Value of J after J = (i++) + (++i) is: %d\n", j);
return 0;
}
```

Output

```
a. Result of expression A+B*C+(D*E) + F*G is: 70
b. Result of expression A/B*C-B+A*D/3 is: 1
c. Result of expression A+++B---A is: 11
d. Value of J after J = (i++) + (++i) is: 12
```

ii) Find the maximum of three numbers using conditional operator

Algorithm

1. Declare num1, num2, num3, max.
2. Read num1, num2, num3 using scanf() function.
3. Calculate max value using the conditional operator (ternary operator) to find the maximum.
4. Print max value.
5. Exit or terminate the program.

Source Code

```
#include <stdio.h>
int main(){
int num1, num2, num3, max;
printf("Enter three numbers: ");
scanf("%d %d %d", &num1, &num2, &num3);
// Using the conditional operator (ternary operator) to find the maximum
max = (num1 > num2) ? ((num1 > num3) ? num1 : num3) : ((num2 > num3) ? num2 : num3);
printf("Maximum number is %d\n", max);
return 0;
}
```

Output

```
Enter three numbers: 5 9 3
Maximum number is: 9
```

iii) Take marks of 5 subjects in integers, and find the total, average in float

Algorithm

1. Declare marks[size],total=0,average.
2. Read marks of 5 subjects
3. Use for loop to read 5 subjects marks.
4. Calculate total and average marks.
5. Print total marks,Average marks.
6. Exit or terminate the program.

Source Code

```
#include <stdio.h>
int main()
{
int marks[5] ,i;
float total = 0.0, average;
printf("Enter marks of 5 subjects: ");
for ( i = 0; i < 5; i++)
{
scanf("%d", &marks[i]);
total += marks[i];
}
average = total / 5.0;
printf("Total marks: %.2f\n", total);
printf("Average marks: %.2f\n", average);
return 0;
}
```

Output

```
Enter marks of 5 subjects: 30 40 50 60 70
Total marks: 250.00
Average marks: 50.00
```

WEEK 5

Objective: Explore the full scope of different variants of ―if construct‖ namely if-else, null- else, if-else if*-else, switch and nested-if including in what scenario each one of them can be used and how to use them. Explore all relational and logical operators while writing conditionals for ―if construct‖.

Suggested Experiments/Activities:

Tutorial 5: Branching and logical expressions:

Lab 5: Problems involving if-then-else structures.

i) Write a C program to find the max and min of four numbers using if-else.

Algorithm

1. This program prompts the user to enter four numbers.
2. Uses if-else statements to compare and find the maximum and minimum numbers among them.
3. The max and min variables are used to store the maximum and minimum values, respectively.
4. Finally, the program prints the maximum and minimum numbers.

Source Code

```
#include <stdio.h>
int main()
{
  int num1, num2, num3, num4;
  int max, min;
  printf("Enter four numbers: ");
  scanf("%d %d %d %d", &num1, &num2, &num3, &num4);
  if (num1 >= num2 && num1 >= num3 && num1 >= num4)
{
  max = num1;
  }
  else if (num2 >= num1 && num2 >= num3 && num2 >= num4)
{
  max = num2;
  }
else if (num3 >= num1 && num3 >= num2 && num3 >= num4)
{
  max = num3;
  }
else
{
  max = num4;
  }

  if (num1 <= num2 && num1 <= num3 && num1 <= num4)
{
  min = num1;
  }
else if (num2 <= num1 && num2 <= num3 && num2 <= num4)
{
  min = num2;
  }
else if (num3 <= num1 && num3 <= num2 && num3 <= num4)
{
  min = num3;
  }
else
{
  min = num4;
  }
  printf("Maximum number: %d\n", max);
  printf("Minimum number: %d\n", min);
  return 0;
}
```

Output

```
Enter four numbers: 34
56
12
789
Maximum number: 789
Minimum number: 12
```

ii) Write a C program to generate electricity bill.

Algorithm

1. Input unit consumed by customer in some variable say unit.
2. If unit consumed less or equal to 50 units. Then amt = unit * 0.50.
3. If unit consumed more than 50 units but less than 100 units. Then add the first 50 units amount i.e. 25 to final amount and compute the rest 50 units amount. Which is given by amt = 25 + (unit-50) * 0.75. I have used units-50, since I already calculated first 50 units which is 25.
4. Similarly check rest of the conditions and calculate total amount.
5. After calculating total amount. Calculate the surcharge amount i.e. sur_charge = total_amt * 0.20. Add surcharge amount to net amount. Which is given by net_amt = total_amt + sur_charge

Source Code

```
#include <stdio.h>
int main()
{
    int unit;
    float amt, total_amt, sur_charge;
    printf("Enter total units consumed: ");
    scanf("%d", &unit);
    if(unit <= 50)
    {
        amt = unit * 0.50;
    }
    else if(unit <= 150)
    {
        amt = 25 + ((unit-50) * 0.75);
    }
    else if(unit <= 250)
    {       amt = 100 + ((unit-150) * 1.20);     }
    else
    {       amt = 220 + ((unit-250) * 1.50);     }
    sur_charge = amt * 0.20;
    total_amt  = amt + sur_charge;
    printf("Electricity Bill = Rs. %.2f", total_amt);
    return 0;
}
```

Output

```
Enter total units consumed: 600
Electricity Bill = Rs. 894.00
```

iii) Find the roots of the quadratic equation.

Algorithm

```
1. Input the value of a, b, c.
2. Calculate k = bb – 4a*c
3. If (d < 0)
   Print "Roots are Imaginary, calculate root1 = (-b +i ?k)/ 2a and root2 =(b + i?k)/ 2a.
4. else if (d = 0)
   Print "Roots are Equal" and calculate root1 = root2 = (-b / 2*a)
5. else  Print "Roots are real and calculate root1 = -b + ?d / 2a and root2 = -b – ?d / 2a.
6. Print root1 and root2.
7. End the algorithm.
```

Source Code

```c
#include <math.h>
#include <stdio.h>
int main()
{
   double a, b, c, discriminant, root1, root2, realPart, imagPart;
   printf("Enter coefficients a, b and c: ");
   scanf("%lf %lf %lf", &a, &b, &c);
   discriminant = b * b - 4 * a * c;
      if (discriminant > 0)     {
      root1 = (-b + sqrt(discriminant)) / (2 * a);
      root2 = (-b - sqrt(discriminant)) / (2 * a);
      printf("root1 = %.2lf and root2 = %.2lf", root1, root2);     }
   else if (discriminant == 0)
 {       root1 = root2 = -b / (2 * a);
        printf("root1 = root2 = %.2lf;", root1);      }
   else   {
      realPart = -b / (2 * a);
      imagPart = sqrt(-discriminant) / (2 * a);
      printf("root1 = %.2lf+%.2lfi and root2 = %.2f-%.2fi", realPart, imagPart, realPart, imagPart);
}
   return 0;
}
```

Output

```
Enter coefficients a, b and c: 2.4 5 8
root1 = -1.04+1.50i and root2 = -1.04-1.50i
```

iv) Write a C program to simulate a calculator using switch case.

Algorithm

1. Display a menu for the user to choose an arithmetic operation.
2. Take input of operator and then operands.
3. Check whether the user wants to quit the program if yes then quit it, for this, we can use a special character and tell the user about it, like here we used “x”.
4. Using if-else statements we will check operators and do operations accordingly.

Source Code

```
#include <stdio.h>
int main()
{
  char op;
  double first, second;
  printf("Enter an operator (+, -, *, /): ");
  scanf("%c", &op);
  printf("Enter two operands: ");
  scanf("%lf %lf", &first, &second);
  switch (op)
{
    case '+':     printf("%.1lf + %.1lf = %.1lf", first, second, first + second);
                  break;
    case '-':     printf("%.1lf - %.1lf = %.1lf", first, second, first - second);
                  break;
    case '*':     printf("%.1lf * %.1lf = %.1lf", first, second, first * second);
                  break;
    case '/':     printf("%.1lf / %.1lf = %.1lf", first, second, first / second);
                  break;
        default:
      printf("Error! operator is not correct");
  }
  return 0;
}
```

Output

```
Enter an operator (+, -, *, /): +
Enter two operands: 23
89
23.0 + 89.0 = 112.0
```

v) Write a C program to find the given year is a leap year or not.

Algorithm

1. Take a year as input.
2. Check whether a given year is divisible by 400.
3. Check whether a given year is divisible by 100.
4. Check whether a given year is divisible by 4.
5. If the condition at step 2 and 4 becomes true, then the year is a leap year.
6. If the condition at step 3 becomes true, then the year is not a leap year.

Source Code

```
#include<stdio.h>
void main()
{
    int year;
     printf("Enter a year \n");
    scanf("%d", &year);
    if ((year % 400) == 0)
        printf("%d is a leap year \n", year);
    else if ((year % 100) == 0)
        printf("%d is a not leap year \n", year);
    else if ((year % 4) == 0)
        printf("%d is a leap year \n", year);
    else
        printf("%d is not a leap year \n", year);
}
```

Output

```
Enter a year
2000
2000 is a leap year
```

WEEK 6

Objective: Explore the full scope of iterative constructs namely while loop, do-while loop and for loop in addition to structured jump constructs like break and continue including when each of these statements is more appropriate to use.

Suggested Experiments/Activities:

Tutorial 6: Loops, while and for loops

Lab 6: Iterative problems e.g., the sum of series

i) Find the factorial of given number using any loop.

Algorithm

1. Start the program.
2. Declare variables to store the input number and the factorial result.
3. Prompt the user to enter a number.
4. Read the input number from the user.
5. Set the factorial result variable to 1.
6. Use a loop (e.g., for loop or while loop) to iterate from 1 to the input number.
7. Inside the loop, multiply the factorial result by the current loop counter.
8. After the loop, the factorial result will hold the factorial of the input number.
9. Print the factorial result.
10. End the program.

Source Code

```
#include<stdio.h>
int main()
{
   int i, number, factorial = 1;
   printf("Enter a number: ");
   scanf("%d",&number);
   for(i=1; i<=number; i++)
  {
       factorial = factorial * i;
   }
   printf("Factorial of %d is: %d",number, factorial);
   return 0;
}
```

Output

```
Enter a number: 5
Factorial of 5 is: 120
```

ii) Find the given number is a prime or not.

Algorithm

1. Declare variables n and count. Initialize count with 0. We will store the number of divisors of n in count. Input n. Check if the number is equal to 1. If yes, print that 1 is neither prime nor composite and return from the program.
2. Create a for loop that iterates from 2 to n.
3. Within the loop, for every index, we will check whether n is divisible by i or not. If yes, increment count(number of divisors of n). Otherwise, continue with the next iteration.
4. After we come out of the loop, check the value of count. If it is equal to zero, it means that n can only be divided by 1 and itself. If it is more than zero, n is a composite number.
5. Print the results.

Source Code

```
#include <stdio.h>
int main()
{
    int n;        //Declare the nummber
    printf("Enter the number: ");
    scanf("%d",&n);    //Initialize the number
    if(n == 1){
        printf("1 is neither prime nor composite.");
        return 0;
    }
    int count = 0;        //Declare a count variable
    for(int i = 2; i < n; i++) {  //Check for factors
            if(n % i == 0)
            count++;
    }
    if(count == 0)          //Check whether Prime or not
    {
      printf("%d is a prime number.", n);
   }
    else    {        printf("%d is not a prime number.", n);      }
    return 0;
}
```

Output

```
Enter the number: 11
11 is a prime number.
```

iii) Compute sine and cos series

C program for Sine Series

Theory

Sine Series:

Sine Series is a series which is used to find the value of Sin(x).where, **x** is the angle in **degree** which is converted to **Radian**.

The formula used to express the Sin(x) as Sine Series is

$$\sin x = \sum_{x=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Expanding the above notation, the formula of Sine Series is

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

For example,

Let the value of **x** be **30**.

$$x = 30 * \frac{\pi}{180} = 30 * \frac{3.14159}{180} = 0.52359$$

So, Radian value for **30** degree is **0.52359.**

$$\text{Sin}(0.52359) = 0.52359 - \frac{0.52359^3}{3!} + \frac{0.52359^5}{5!} - \frac{0.52359^7}{7!} + \dots\dots\dots$$

So, the value of **Sin(30)** is **0.5.**

Algorithm

1. First the computer reads the value of ‘x’ and ‘n’ from the user.
2. Then ‘x’ is converted to radian value.
3. Then using for loop the value of Sin(x) is calculate.
4. Finally the value of Sin(x) is printed.

Source Code

```
#include<stdio.h>
void main()
{
   int i, n;
   float x, sum, t;
   printf(" Enter the value for x : ");
   scanf("%f",&x);
   printf(" Enter the value for n : ");
   scanf("%d",&n);
```

```
 x=x*3.14159/180;
t=x;
sum=x;

/* Loop to calculate the value of Sine */
for(i=1;i<=n;i++)
{
   t=(t*(-1)*x*x)/(2*i*(2*i+1));
   sum=sum+t;
}
    printf(" The value of Sin(%f) = %.4f",x,sum);
}
```

Output

```
Enter the value for x : 2
Enter the value for n : 5
The value of Sin(0.034907) = 0.0349
```

C program for Cosine Series

Cosine Series:

Cosine Series is a series which is used to find the value of Cos(x). where, **x** is the angle in **degree** which is converted to **Radian**.

The formula used to express the Cos(x) as Cosine Series is

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

Expanding the above notation, the formula of Cosine Series is

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

For example,

Let the value of **x** be **30**.

$$x = 30 * \frac{\pi}{180} = 30 * \frac{3.14159}{180} = 0.52359$$

So, Radian value for **30** degree is **0.52359.**

$$\text{Cos}(0.52359) = 1 - \frac{0.52359^2}{2!} + \frac{0.52359^4}{4!} - \frac{0.52359^6}{6!} + \ldots\ldots\ldots$$

So, the value of **Cos(30)** is **0.8660.**

Algorithm

1. First the computer reads the value of ‘x’ and ‘n’ from the user.
2. Then ‘x’ is converted to radian value.
3. Then using for loop the value of Cos(x) is calculate.
4. Finally the value of Cos(x) is printed.

Source Code

```
#include<stdio.h>
void main()
{
    int i, n;
    float x, sum=1, t=1;
    printf(" Enter the value for x : ");
    scanf("%f",&x);
    printf(" Enter the value for n : ");
    scanf("%d",&n);
     x=x*3.14159/180;
         /* Loop to calculate the value of Cosine */
    for(i=1;i<=n;i++)
    {
        t=t*(-1)*x*x/(2*i*(2*i-1));
        sum=sum+t;
    }
    printf(" The value of Cos(%f) is : %.4f", x, sum);
}
```

Output

```
Enter the value for x : 5
 Enter the value for n : 3
 The value of Cos(0.087266) is : 0.9962
```

iv) Checking a number palindrome

Algorithm

1. Accept the input number from the user.
2. Copy the input number to a separate variable for comparison.
3. Initialize a variable to store the reverse of the number (initially set to 0).
4. Extract the last digit of the number using the modulus operator (%).
5. Multiply the reverse variable by ten and add the extracted digit to it.
6. Divide the original number by 10 to remove the last digit.
7. Repeat steps 4-6 until the original number becomes 0.
8. Compare the reverse variable with the initially copied number.
9. If they are equal, the number is a palindrome; otherwise, it is not.

Source Code

```
#include <stdio.h>
int main() {
    int number, reverse = 0, original, remainder;
    printf("Enter a number: ");
    scanf("%d", &number);
    original = number;

    while(number!=0){
        remainder = number%10;
        reverse = reverse*10 + remainder;
        number = number/10;
    }
   if (original == reverse)
{
    printf("%d is a palindrome number.\n", original);
}
else
{
    printf("%d is not a palindrome number.\n", original);
}
return 0;
}
```

Output

```
Enter a number: 323
323 is a palindrome number.
```

v) Construct a pyramid of numbers.

Algorithm

1. Take input for number of rows
2. First, we traversed over a for loop from 1 to number of rows to create N rows.
3. At ith row, using for loop we printed whitespaces in first rows-i columns and stars in remaining 2*i-1 columns.

Source Code

```
#include <stdio.h>
int main()
{
   int i, space, rows, k = 0;
   printf("Enter the number of rows: \n");
    // taking input for number of rows
   scanf("%d", &rows);
     // Outer loop to handle number of rows
   for (i = 1; i <= rows; ++i, k = 0)
   {
      for (space = 1; space <= rows - i; ++space)
      {
         // Printing spaces
         printf("  ");
      }
      while (k != 2 * i - 1)
      {
         printf("%d  ", k);
         ++k;
      }
      printf("\n");
   }
   return 0;
}
```

Output

```
Enter the number of rows:  5
      0
    0  1  2
   0  1  2  3  4
  0  1  2  3  4  5  6
0  1  2  3  4  5  6  7  8
```

UNIT III
WEEK 7

Objective: Explore the full scope of Arrays construct namely defining and initializing 1-D and 2-D and more generically n-D arrays and referencing individual array elements from the defined array. Using integer 1-D arrays, explore search solution linear search.

Suggested Experiments/Activities:

Tutorial 7: 1 D Arrays: searching.

Lab 7:1D Array manipulation, linear search

i) Find the min and max of a 1-D integer array.

Algorithm

1. Input size and element in array, store it in some variable say size and arr.
2. Declare two variables max and min to store maximum and minimum. Assume first array element as maximum and minimum both, say max = arr[0] and min = arr[0].
3. Iterate through array to find maximum and minimum element in array. Run loop from first to last array element i.e. 0 to size - 1. Loop structure should look like for(i=0; i<size; i++).
4. Inside loop for each array element check for maximum and minimum. Assign current array element to max, if (arr[i] > max). Assign current array element to min if it is less than min i.e. perform min = arr[i] if (arr[i] < min).

Source Code

```
#include <stdio.h>
#define MAX_SIZE 100   // Maximum array size
int main()
{
   int arr[MAX_SIZE];
   int i, max, min, size;
   /* Input size of the array */
   printf("Enter size of the array: ");
   scanf("%d", &size);
   /* Input array elements */
   printf("Enter elements in the array: ");
   for(i=0; i<size; i++)
   {
      scanf("%d", &arr[i]);
   }
   /* Assume first element as maximum and minimum */
   max = arr[0];
   min = arr[0];
```

```
    /*  Find maximum and minimum in all array elements.    */
    for(i=1; i<size; i++)
    {
        /* If current element is greater than max */
        if(arr[i] > max)
        {
            max = arr[i];
        }
        /* If current element is smaller than min */
        if(arr[i] < min)
        {
            min = arr[i];
        }
    }
    /* Print maximum and minimum element */
    printf("Maximum element = %d\n", max);
    printf("Minimum element = %d", min);
    return 0;
}
```

Output

```
Enter size of the array: 5
Enter elements in the array: 45
67
23
12
89
Maximum element = 89
Minimum element = 12
```

ii) Perform linear search on1D array.

Algorithm

1. First, we have to traverse the array elements using a **for** loop.
2. In each iteration of **for loop,** compare the search element with the current array element, and -
3. If the element matches, then return the index of the corresponding array element.
4. If the element does not match, then move to the next element.
5. If there is no match or the search element is not present in the given array, return **-1.**

Source Code

```
#include <stdio.h>
int main()
{
  int array[100], search, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);
  printf("Enter %d integer(s)\n", n);
  for (c = 0; c < n; c++)
  scanf("%d", &array[c]);
  printf("Enter a number to search\n");
  scanf("%d", &search);
  for (c = 0; c < n; c++)
  {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);
  return 0;
}
```

Output

```
Enter number of elements in array 5
Enter 5 integer(s)  45  12   89   9   99
Enter a number to search 0
0 isn't present in the array.
```

iii) The reverse of a 1D integer array

Algorithm

1. The program starts by asking the user to enter the limit of the array.
2. The user is then prompted to enter the values of the array, which are stored in the variable 'a' using a for loop.
3. The program then prints out the given values of the array using another for loop. After this, the program uses another for loop to reverse the array.
4. The loop starts by initializing the variable 'i' to 0, and it continues running until the value of 'i' is less than the limit of the array.
5. In each iteration, the value of a[n-i-1] is stored in the variable 'b' at the same position as 'i'. This effectively reverses the order of the elements in the array.
6. Finally, the program prints out the updated values of the array using another for loop, which shows the reversed array.

Source Code

```
#include<stdio.h>
int main()
{
  int i,j,n,a[10],m,o,b[10];
  printf("\nEnter the Limit:");
  scanf("%d",&n);
  printf("\nEnter the values:");
  for(i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
  }
  printf("\nGiven values are:");
  for(i=0;i<n;i++)
  {
    printf("\na[%d]=%d",i,a[i]);
  }
  for(i=0;i<n;i++)
  {
    b[i]=a[n-i-1];
  }
  printf("\nUpdated values are :");
  for(i=0;i<n;i++)
  {
    printf("\na[%d]=%d",i,b[i]);
  }
  return 0;
}
```

Output

```
Enter the Limit:5
Enter the values:45
67
12
5
43

Given values are:
a[0]=45
a[1]=67
a[2]=12
a[3]=5
a[4]=43
Updated values are :
a[0]=43
a[1]=5
a[2]=12
a[3]=67
a[4]=45
```

iv) Find 2's complement of the given binary number.

Theory
Signed integers are frequently represented in C using the **two's complement notation**. Using the same **binary representation** offers a mechanism to express both **positive** and **negative integers**. The **most significant bit (MSB)** is used as the **sign bit** in a **two's complement representation**, where **0** denotes a **positive integer**, and **1** denotes a **negative number**.

Starting with a **negative number's** absolute value in binary form, you may take the **one's complement (bitwise negation)** of that value to get the **two's complement** representation of the **negative integer**. You add **1** to the **resultant value** to acquire the representation of the **two's complement**.

The **two's complement encoding** in C can represent **signed integers** and can perform fast arithmetic operations. One benefit of employing two's complement is the ability to do **addition** and **subtraction** using the same binary operations as for unsigned numbers.

The **binary numbers** are added together like **unsigned integers** when adding two's complement. A carry-out from the location of the **main critical bit** is just disregarded. Due to this fact, handling **signed numbers** differently is not necessary, and addition becomes simple.
Consider adding **-5** and **-3** using the **8-bit two's complement** representation, for instance:
Binary number for **-5** is **11111011.**
Binary number for **-3** is 11111101.

carrying out the addition:

1. 11111011 (-5)
2. 11111101 (-3)
3. -------------
4. 111110100 (-8)

The answer is **111110100**, which in **two's complement** is equal to **-8**.

Similar to addition, subtraction may be done by treating the **second operand's** two's complement as if it were addition. In other words, you add the two's complement of a negative number to the first operand to remove it.

For instance, when **-3** is subtracted from **-5:**

In binary, **-5** is represented by **11111011** and **-(-3)** by **00000011** (two's complement of **-3**)

Carrying out the subtraction

1. 11111011 (-5)

2. 00000011 (+3)
3. -------------
4. 11111110 (-8)

The outcome is **11111110**, which in two's complement is equal to **-8**.

In C, the **2s complement** is a binary representation of a negative number that is created by adding one to the **1s complement**. Computer systems frequently employ this idea to represent signed numbers and efficiently carry out arithmetic operations.

To get the **2s complement** of a binary integer, one must first determine the **1s complement** of the number by flipping the bits. After that, the representation of the **2s complement** is obtained by **adding one** to the **1s complement**. The **most significant bit (MSB)** will function as a sign bit by expressing whether a number is **positive** or **negative**.

The computation of the **2s complement** for a given binary integer is shown in the attached C program. The user is prompted to input both the **binary number** and the number of bits. After that, the program does the required procedures to acquire the 1s complement, and then the **2s complement**. The findings are then shown.
In computer science and programming, it's crucial to comprehend the **2s complement** representation since it makes it possible to handle negative values expressed in binary effectively. It makes **addition, subtraction**, and **logical operations** simpler on both **positive** and **negative numbers**. The range of **representable integers** is symmetric about **zero** due to the **2s complement** representation, making it appropriate for various numerical operations.

Algorithm

1. First, we input the number of bits, and it gets stored in the '**n**' variable.
2. After entering the number of bits, we declare character array, i.e., **char binary[n+1],** which holds the binary number. The '**n**' is the number of bits which we entered in the previous step; it basically defines the size of the array.
3. We declare two more arrays, i.e., **onescomplement[n+1]**, and **twoscomplement[n+1].** The **onescomplement[n+1]** array holds the ones complement of a binary number while the **twoscomplement[n+1]** array holds the two's complement of a binary number.
4. Initialize the **carry** variable and assign 1 value to this variable.
5. After declarations, we input the binary number.
6. Now, we simply calculate the one's complement of a binary number. To do this, we create a **loop** that iterates throughout the binary array, **for(int i=0;i<n;i++)**. In for loop, the condition is checked whether the bit is 1 or 0. If the bit is 1

then **onescomplement[i]=0** else **onescomplement[i]=1**. In this way, one's complement of a binary number is generated.

7. After calculating one's complement, we generate the 2s complement of a binary number. To do this, we create a **loop** that iterates from the last element to the starting element. In for loop, we have three conditions:
8. If the bit of onescomplement[i] is 1 and the value of carry is 1 then we put 0 in twocomplement[i].
9. If the bit of onescomplement[i] is 0 and the value of carry is 1 then we put 1 in twoscomplement[i] and 0 in carry.
10. If the above two conditions are false, then onescomplement[i] is equal to twoscomplement[i].

Source Code

```
#include <stdio.h>
int main()
{
   int n;  // variable declaration
   printf("Enter the number of bits do you want to enter :");
   scanf("%d",&n);
   char binary[n+1];  // binary array declaration;
   char onescomplement[n+1]; // onescomplement array declaration
   char twoscomplement[n+1]; // twoscomplement array declaration
   int carry=1; // variable initialization
   printf("\nEnter the binary number : ");
   scanf("%s", binary);
   printf("%s", binary);
   printf("\nThe ones complement of the binary number is :");
   // Finding onescomplement in C
   for(int i=0;i<n;i++)
   {
       if(binary[i]=='0')
       onescomplement[i]='1';
       else if(binary[i]=='1')
       onescomplement[i]='0';
   }
   onescomplement[n]='\0';
   printf("%s",onescomplement);
 printf("\nThe twos complement of a binary number is : ");
  // Finding twoscomplement in C
for(int i=n-1; i>=0; i--)
    {
        if(onescomplement[i] == '1' && carry == 1)
        {
            twoscomplement[i] = '0';
        }
        else if(onescomplement[i] == '0' && carry == 1)
        {
            twoscomplement[i] = '1';
            carry = 0;
        }
        else
        {
            twoscomplement[i] = onescomplement[i];
        }
    }
twoscomplement[n]='\0';
printf("%s",twoscomplement);
return 0;
}
```

Output

```
Enter the number of bits do you want to enter :8
Enter the binary number : 10100011
10100011
The ones complement of the binary number is :01011100
The twos complement of a binary number is : 01011101
```

v) Eliminate duplicate elements in an array.

Algorithm

1. Take size of the array n as input from user.
2. Initialize an array arr of size n.
3. Enter the elements into the array.
4. Run two for loops and check for unique elements.
5. Store all the unique elements in another array temp.
6. Print all the elements inside the array temp.
7. Exit.

Source Code

```
#include <stdio.h>
int main()
{
    int n, count = 0;
    printf("Enter number of elements in the array: ");
    scanf("%d", &n);
    int arr[n], temp[n];
    if(n==0)
    {
        printf("No element inside the array.");
     }
    printf("Enter elements in the array: ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("\nArray Before Removing Duplicates: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    // To store unique elements in temp after removing the duplicate elements
    for (int i = 0; i < n; i++)
    {
        int j;
        for (j = 0; j < count; j++)
        {
          if (arr[i] == temp[j])
            break;
        }
        if (j == count)
        {
          temp[count] = arr[i];
          count++;
        }
    }

    printf("\nArray After  Removing Duplicates: ");
    for (int i = 0; i < count; i++)
        printf("%d ", temp[i]);
    return 0;
}
```

Output

```
Enter number of elements in the array: 5
Enter elements in the array: 34
34
67
12
67
Array Before Removing Duplicates: 34 34 67 12 67
Array After  Removing Duplicates: 34 67 12
```

WEEK 8

Objective: Explore the difference between other arrays and character arrays that can be used as Strings by using null character and get comfortable with string by doing experiments that will reverse a string and concatenate two strings. Explore sorting solution bubble sort using integer arrays.

Suggested Experiments/Activities:

Tutorial 8: 2 D arrays, sorting and Strings.

Lab 8: Matrix problems, String operations, Bubble sort

i) Addition of two matrices

Algorithm

1. We first prompt the user to enter the number of rows and columns of the matrices, and then we use nested loops to read in the elements of both matrices from the user.
2. We then add the two matrices by iterating through each element and storing the sum in the corresponding element of a third matrix c.
3. Finally, we print the resulting matrix c to the console.

Source Code

```
#include <stdio.h>
int main() {
  int r, c, a[100][100], b[100][100], sum[100][100], i, j;
  printf("Enter the number of rows (1-100): ");
  scanf("%d", &r);
  printf("Enter the number of columns (1-100): ");
  scanf("%d", &c);
   printf("\nEnter elements of Matrix 1:\n");
  for (i = 0; i < r; ++i)
    for (j = 0; j < c; ++j) {
      printf("Enter element a%d%d: ", i + 1, j + 1);
      scanf("%d", &a[i][j]);
    }
   printf("Enter elements of Matrix 2:\n");
  for (i = 0; i < r; ++i)
    for (j = 0; j < c; ++j) {
      printf("Enter element b%d%d: ", i + 1, j + 1);
      scanf("%d", &b[i][j]);
    }
   // adding two matrices
  for (i = 0; i < r; ++i)
    for (j = 0; j < c; ++j) {
      sum[i][j] = a[i][j] + b[i][j];
    }
   // printing the result
  printf("\nSum of the two matrices: \n");
  for (i = 0; i < r; ++i)
    for (j = 0; j < c; ++j) {
      printf("%d   ", sum[i][j]);
      if (j == c - 1) {
        printf("\n\n");
      }
    }
   return 0;
}
```

Output

```
Enter the number of rows (1-100): 2
Enter the number of columns (1-100): 2

Enter elements of Matrix 1:
Enter element a11: 34
Enter element a12: 23
Enter element a21: 12
Enter element a22: 11
Enter elements of Matrix 2:
Enter element b11: 4
Enter element b12: 56
Enter element b21: 33
Enter element b22: 9

Sum of the two matrices:
38   79

45   20
```

ii) Multiplication two matrices

Algorithm

1. Input the size of both matrices
2. If matrix can't be multiplied print that they can't be multiplied
3. Else, take input of elements of both matrices
4. Declare a resultant matrix
   Fill the matrix using formula res[i][j] += a[i][k] * b[k][j];
5. Print the multiplied matrix.

Source Code

```
#include<stdio.h>
int main() {
   int m, n, p, q, i, j, k;
   int a[10][10], b[10][10], res[10][10];
   printf("Enter the order of first matrix\n");
   scanf("%d%d", & m, & n);
   printf("Enter the order of second matrix\n");
   scanf("%d%d", & p, & q);
   if (n != p) {
      printf("Matrix is incompatible for multiplication\n");
   } else {
      printf("Enter the elements of Matrix-A:\n");
      for (i = 0; i < m; i++) {
         for (j = 0; j < n; j++) {
            scanf("%d", & a[i][j]);
         }
      }
      printf("Enter the elements of Matrix-B:\n");
      for (i = 0; i < p; i++) {
         for (j = 0; j < q; j++) {
            scanf("%d", & b[i][j]);
         }
      }
      for (i = 0; i < m; i++) {
         for (j = 0; j < q; j++) {
            res[i][j] = 0;
            for (k = 0; k < p; k++) {
               res[i][j] += a[i][k] * b[k][j];
            }
         }
```

```
        }
        printf("The product of the two matrices is:-\n");
        for (i = 0; i < m; i++) {
            for (j = 0; j < q; j++) {
                printf("%d\t", res[i][j]);
            }
            printf("\n");
        }
    }
    return 0;
}
```

Output -1

```
Enter the order of first matrix
2
2
Enter the order of second matrix
1
1
Matrix is incompatible for multiplication
```

Output -2

```
Enter the order of first matrix
2
2
Enter the order of second matrix
2
2
Enter the elements of Matrix-A:
1
2
3
4
Enter the elements of Matrix-B:
4
3
5
6
The product of the two matrices is:-
14      15
32      33
```

iii) Sort array elements using bubble sort

Algorithm

1. In Bubble sort algorithm we compare the first two elements of an array and swap them if required.
2. If we want to sort the elements of array in ascending order and if the first element is greater than second then, we need to swap the elements.
3. If the first element is smaller than second, we don’t need to swap the elements. This process go on until last and second last element is compared and swapped.

Source Code

```
#include <stdio.h>
int main()
{
   int  n, j, i, swap;
   printf("Enter number of elements\n");
   scanf("%d", &n);
   int array[n];
   printf("Enter %d integers\n", n);
   for (i= 0; i < n; i++)
   {
      scanf("%d", &array[i]);
   }
   for (i = 0 ; i < n - 1; i++)
   {
      for (j = 0 ; j < n - i- 1; j++)
      {
         if (array[j] > array[j+1])
         {
            swap       = array[j];
            array[j]   = array[j+1];
            array[j+1] = swap;
         }
      }
   }
    printf("Sorted list in ascending order:\n");
    for (i = 0; i < n; i++)
      printf("%d\n", array[i]);
   return 0;
}
```

Output

```
enter number of elements 5
Enter 5 integers
34
12
67
1
2
Sorted list in ascending order:
1
2
12
34
67
```

iv) Concatenate two strings without built-in functions

Algorithm

1. First, ensure the first string has enough space to hold all characters of the second string (i.e. the size of the first string is the sum of the sizes of both strings).
2. Iterate the first string and find the end where a null-terminator is encountered.
3. Now, copy the second string to the first string starting from the found position.
4. Finally, add a null terminator at the end of the concatenated string.

Source Code

```
#include <stdio.h>
int main()
{
  char str1[50], str2[50], i, j;
  printf("\nEnter first string: ");
  scanf("%s",str1);
  printf("\nEnter second string: ");
  scanf("%s",str2);
  /* This loop is to store the length of str1 in i  It just counts the number of characters in str1
   You can also use strlen instead of this.  */
  for(i=0; str1[i]!='\0'; ++i);
   /* This loop would concatenate the string str2 at  the end of str1   */
  for(j=0; str2[j]!='\0'; ++j, ++i)
  {
    str1[i]=str2[j];
  }
  // \0 represents end of string
  str1[i]='\0';
  printf("\nOutput: %s",str1);
  return 0;
}
```

Output

```
Enter first string: hello
Enter second string: world
Output: helloworld
```

v) Reverse a string using built-in and without built-in string functions

Reverse a string using built-in string functions

Algorithm

1. The user is prompted to enter the input string in the main function.
2. Subsequently, the main function calls the reverseString function and prints the reversed string.

Source Code

```
#include<stdio.h>
#include<string.h>
 int main()
{
 char strng[10] = "World";
printf("The input string is %s\n", strng);
 // strrev function is called on the given string
printf("The reversed string is %s", strrev(strng));
 return 0;
}
```

Output

```
The input string is Reading
The reversed string is gnideaR
```

Reverse a string without using built-in string functions

Algorithm

1. If you want to reverse a string in C using function, you must first determine the string's length.
2. Subsequently, you need to run a for loop to allocate characters of another string equal to the characters of the entered string in reverse order.
3. The part of this program that uses for loop will resemble the program that reverse a string in C using for loop.

Source Code

```
#include <stdio.h>
#include <string.h>
void  main ()
{
  char string[20], temp;
  int i, length;
  printf ("Enter String : ");
  scanf ("%s", string);
  length = strlen (string) - 1;
for (i = 0; i < strlen (string) / 2; i++)
          {
            temp = string[i];
            string[i] = string[length];
            string[length--] = temp;
          }
printf ("Reverse string :%s", string);
}
```

Output

```
Enter String : happy
Reverse string :yppah
```

UNIT IV
WEEK 9

Objective: Explore pointers to manage a dynamic array of integers, including memory allocation value initialization, resizing changing and reordering the contents of an array
and memory de-allocation using malloc (), calloc (), realloc () and free () functions. Gain experience processing command-line arguments received by C
Suggested Experiments/Activities:
Tutorial 9: Pointers, structures and dynamic memory allocation

Lab 9: Pointers and structures, memory dereference.
i) Write a C program to find the sum of a 1D array using malloc()

Algorithm

```
1. sum := 0
2. take one input and store it to n
3. arr := dynamically create an array of size n
4. for initialize i := 0, when i < n, update (increase i by 1),
   do: take an input and store it to arr[i]
5. for initialize i := 0, when i < n, update (increase i by 1),
    do: sum := sum + arr[i]
6. return sum
```

Source Code

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int* ptr; //declaration of integer pointer
   int size; //to store array size
   int i; //loop counter
   int sum; //to store sum of all elements
   printf("Enter size of the array: ");
   scanf("%d", &size);
   //declare memory dynamically
   ptr = (int*)malloc(size * sizeof(int));
   //read array elements
   for (i = 0; i < size; i++) {
      printf("Enter element %02d: ", i + 1);
      scanf("%d", (ptr + i));
   }
   //print array elements
```

```
    printf("\nEntered array elements are:\n");
    for (i = 0; i < size; i++) {
        printf("%d\n", *(ptr + i));
    }

    //calculate sum of all elements
    sum = 0; //assign 0 to replace garbage value
    for (i = 0; i < size; i++) {
        sum += *(ptr + i);
    }
    printf("Sum of array elements is: %d\n", sum);
    //free memory
    free(ptr); //hey, don't forget to free dynamically allocated memory.
    return 0;
}
```

Output

```
Enter size of the array: 5
Enter element 01: 12
Enter element 02: 45
Enter element 03: 1
Enter element 04: 3
Enter element 05: 9

Entered array elements are:
12
45
1
3
9
Sum of array elements is: 70
```

ii) Write a C program to find the total, average of n students using structures

Algorithm

1. Start the program.
2. Declare a structure student with variables name, roll no, mark, tot.
3. Declare the necessary variables.
4. Create a structure variable using arrays.
5. Get the student roll no, name, mark using for loop.
6. Calculate the total marks using arithmetic operator.
7. Using for loop display name, roll no and total for each student.
8. Stop the program.

Source Code

```
#include <stdio.h>
struct student
{
int rollno,tot;
char name[25];
int mark[5];
 float avg;
};

void main()
{
struct student s[5]; //Data type of '*s' is struct student
int i,n,j;
printf("Enter the number of students:");
scanf("%d",&n);
printf("\t*Students Records*\n");
//take input from user
for(i=0;i<n;i++)
{
printf("\nEnter Student Roll Number: ");
scanf("%d",&s[i].rollno);
printf("\nEnter Student name: ");
scanf("%s",s[i].name);
printf("\nEnter Student 3 subject's marks: ");
for(j=0;j<3;j++)
scanf("%d",&s[i].mark[j]);
}
//calculation
for(i=0;i<n;i++)
{
s[i].tot=0;

for(j=0;j<3;j++)
s[i].tot = s[i].tot+ s[i].mark[j];
s[i].avg=s[i].tot/3;
}
//Display result
for(i=0;i<n;i++)
{
printf("\t*Students Records*\n");
printf("\n=================================\n");
printf("\nStudent's Roll no. :  %d", s[i].rollno);
printf("\nStudent's Name :  %s", s[i].name);
printf("\nStudent's Total Marks :  %d", s[i].tot);
printf("\nStudent's average : %f", s[i].avg);
}
}
```

Output

```
Enter the number of students:3
        *Students Records*
Enter Student Roll Number: 1
Enter Student name: qwe
Enter Student 3 subject's marks: 98
78
95

Enter Student Roll Number: 2
Enter Student name: asd
Enter Student 3 subject's marks: 89
67
99

Enter Student Roll Number: 3
Enter Student name: zxc
Enter Student 3 subject's marks: 99
99
99
```

```
      *Students Records*
=================================
Student's Roll no. :  1
Student's Name :  qwe
Student's Total Marks :  271
Student's average : 90.000000   *Students Records*
=================================
Student's Roll no. :  2
Student's Name :  asd
Student's Total Marks :  255
Student's average : 85.000000   *Students Records*
=================================
Student's Roll no. :  3
Student's Name :  zxc
Student's Total Marks :  297
Student's average : 99.000000
```

iii) Enter n students data using calloc() and display failed students list

Algorithm

1. Start the program.
2. Declare a structure student with variables name, roll no, mark, tot.
3. Declare the necessary variables.
4. Create a structure variable using arrays.
5. Get the student roll no, name, mark using for loop.
6. Calculate the total marks using arithmetic operator.
7. Display pass or fail for each student.
8. Stop the program.

Source Code

```
#include <string.h>
#include <stdio.h>
struct student
{
  char name[10];
  int m[3];
  int total;
  char result[5];
} *p, *s;
void main ()
{
  int i, j, l, n;
  printf ("Enter the no. of students : ");
  scanf ("%d", &n);
  p = (struct student *) calloc (n , sizeof (struct student));
  s = p;
  for (i = 0; i < n; i++)
          {
            printf ("Enter a name : ");
            scanf ("%s", &p->name);
            p->total = 0;
            l = 0;
            for (j = 0; j < 3; j++)
                    {
                    one:printf ("Enter Marks of %d Subject : ", j + 1);
                      scanf ("%d", &p->m[j]);
                      if ((p->m[j]) > 100)
                              {
```

```
                              printf ("Wrong Value Entered");
                              goto one;
                             }
                      p->total += p->m[j];
                      if (p->m[j] < 40)
                             l = 1;
                     }
               if (l == 0)
                     strcpy (p->result, "PASS");
               else
                     strcpy (p->result, "FAIL");
               p++;
              }
      for (i = 0; i < n; i++)
              {
                      printf ("\n%s\t%s", s->name, s->result);
                      s++;
              }
}
```

Output

```
Enter a name : qwe
Enter Marks of 1 Subject : 78
Enter Marks of 2 Subject : 67
Enter Marks of 3 Subject : 90
Enter a name : asd
Enter Marks of 1 Subject : 89
Enter Marks of 2 Subject : 99
Enter Marks of 3 Subject : 89
Enter a name : zxc
Enter Marks of 1 Subject : 99
Enter Marks of 2 Subject : 99
Enter Marks of 3 Subject : 99

qwe     PASS
asd     PASS
zxc     PASS
```

iv) Read student name and marks from the command line and display the student details along with the total.

Algorithm

```
1. Start
2. Prompt the user to enter values through command line
3. Take the values into the varialbes a,b,c,d,sum.
4. Calculate the value of sum =a+b+c.
5. Print the sum.
6. End
```

Source Code

```
#include <stdio.h>
int main (int argc, char *argv[])
{
 char a[100];
  int d, b,c, sum;
  if (argc != 4)
          {
            printf ("please use \"prg_name value1 value2 \"\n");
            return -1;
          }
  a[100] = atoi (argv[1]);
  b = atoi (argv[2]);
  c = atoi (argv[3]);
d = atoi (argv[4]);
  sum = a + b+c;
printf("name =%c", a[100]);
  printf ("Sum of %d, %d , %dis: %d\n", a, b,c, sum);
  return 0;
}
```

Output

```
C:\Users\Dell\OneDrive\Desktop> .\Demo.exe happy  99 99  99
Happy 297
```

v) Write a C program to implement realloc()

Algorithm

1. Start
2. allocating memory for only 1 integer
3. realloc memory size to store 3 integers
4. Print the values.
5. End

Source Code

```
#include<stdio.h>
//To use realloc in our program
#include<stdlib.h>
int main()
{
   int *ptr,i;
   //allocating memory for only 1 integer
   ptr = malloc(sizeof(int));
   ptr[0] = 1;
   //realloc memory size to store 3 integers
   ptr = realloc(ptr, 3 * sizeof(int));
   ptr[1] = 2;
   ptr[2] = 3;
   //printing values
   for(i = 0; i < 3; i++)
        printf("%d\n",ptr[i]);
   return 0;
}
```

Output

```
1
2
3
```

WEEK 10

Objective: Experiment with C Structures, Unions, bit fields and self-referential structures (Singly linked lists) and nested structures

Suggested Experiments/Activities:

Tutorial 10: Bitfields, Self-Referential Structures, Linked lists

Lab10 : Bitfields, linked lists Read and print a date using dd/mm/yyyy format using bit-fields and differentiate the same without using bit- fields

i) Create and display a singly linked list using self-referential structure.

Algorithm

1. Create a Structure for a node in a linked list
2. Declare Functions to create the linked list and display the linked list
3. Inputting the number of nodes for the linked list
4. Call void createNodeList(int n); to Creating the linked list with n nodes
5. Call void displayList(); to Displaying the data entered in the linked list
6. Function to create a linked list with n nodes Allocating memory for the starting node using malloc for first node
7. Checking if memory allocation is successful
8. Reading data for the starting node from user input
9. Creating n nodes and adding them to the linked list and add the links
10. Traversing the linked list and printing each node's data

Source Code

```
#include <stdio.h>
#include <stdlib.h>
// Structure for a node in a linked list
struct node {
    int num;              // Data of the node
    struct node *nextptr;   // Address of the next node
} *stnode;                  // Pointer to the starting node
// Function prototypes
void createNodeList(int n); // Function to create the linked list
void displayList();        // Function to display the linked list
int main()
{
    int n;
    // Displaying the purpose of the program
    printf("\n\n Linked List : To create and display Singly Linked List :\n");
    printf("-------------------------------------------------------------\n");
    // Inputting the number of nodes for the linked list
    printf(" Input the number of nodes : ");
    scanf("%d", &n);
    // Creating the linked list with n nodes
    createNodeList(n);

    // Displaying the data entered in the linked list
    printf("\n Data entered in the list : \n");
    displayList();
    return 0;
}

// Function to create a linked list with n nodes
void createNodeList(int n) {
    struct node *fnNode, *tmp;
    int num, i;
    // Allocating memory for the starting node
    stnode = (struct node *)malloc(sizeof(struct node));
    // Checking if memory allocation is successful
    if(stnode == NULL) {
        printf(" Memory can not be allocated.");
    } else {
        // Reading data for the starting node from user input
        printf(" Input data for node 1 : ");
        scanf("%d", &num);
        stnode->num = num;
        stnode->nextptr = NULL; // Setting the next pointer to NULL
        tmp = stnode;
        // Creating n nodes and adding them to the linked list
        for(i = 2; i <= n; i++) {
            fnNode = (struct node *)malloc(sizeof(struct node));
            // Checking if memory allocation is successful
            if(fnNode == NULL) {
                printf(" Memory can not be allocated.");
                break;
            } else {
                // Reading data for each node from user input
                printf(" Input data for node %d : ", i);
                scanf(" %d", &num);
                fnNode->num = num;      // Setting the data for fnNode
                fnNode->nextptr = NULL; // Setting the next pointer to NULL
                tmp->nextptr = fnNode;  // Linking the current node to fnNode
                tmp = tmp->nextptr;     // Moving tmp to the next node
            }
        }
    }
}

// Function to display the linked list
void displayList()
{
    struct node *tmp;
    if(stnode == NULL) {
        printf(" List is empty.");
    } else {
        tmp = stnode;
        // Traversing the linked list and printing each node's data
        while(tmp != NULL) {
            printf(" Data = %d\n", tmp->num); // Printing the data of the current node
            tmp = tmp->nextptr;               // Moving to the next node in the list
        }
    }
}
```

Output

```
Linked List : To create and display Singly Linked List :
-------------------------------------------------------------
 Input the number of nodes : 5
 Input data for node 1 : 34
 Input data for node 2 : 56
 Input data for node 3 : 78
 Input data for node 4 : 12
 Input data for node 5 : 99
 Data entered in the list :
 Data = 34
 Data = 56
 Data = 78
 Data = 12
 Data = 99
```

ii) Demonstrate the differences between structures and unions using a C program.

Algorithm

1. Declaring structure
2. Declaraing union
3. Creating variable for structure and initializing values
4. Creating variable for union and initializing values
5. Print the values
6. Accessing all members at a time in structure using dot operator and print them
7. Accessing one member at a time in union using dot operator and print them
8. Altering a member value in structure and union and print them
9. Display the Size of structure and union using sizeof operator.

Source Code

```
// A simple C program showing differences between Structure and Union
#include <stdio.h>
#include <string.h>
// declaring structure
struct struct_example
{
   int integer;
   float decimal;
   char name[20];
};
// declaraing union
union union_example
{
   int integer;
   float decimal;
   char name[20];
};
void main()
{
   // creating variable for structure and initializing values difference six
   struct struct_example stru ={5, 15, "John"};
        // creating variable for union and initializing values
   union union_example uni = {5, 15, "John"};
     printf("data of structure:\n integer: %d\n decimal: %.2f\n name: %s\n", stru.integer, stru.decimal,
stru.name);
```

```
   printf("\ndata of union:\n integer: %d\n" "decimal: %.2f\n name: %s\n", uni.integer, uni.decimal, uni.name);
       // difference five
   printf("\nAccessing all members at a time:");
   stru.integer = 163;
   stru.decimal = 75;
   strcpy(stru.name, "John");
   printf("\ndata of structure:\n integer: %d\n " "decimal: %f\n name: %s\n", stru.integer, stru.decimal, stru.name);
        uni.integer = 163;
   uni.decimal = 75;
   strcpy(uni.name, "John");
   printf("\ndata of union:\n integeer: %d\n " "decimal: %f\n name: %s\n", uni.integer, uni.decimal, uni.name);
     printf("\nAccessing one member at a time:");
   printf("\ndata of structure:");
   stru.integer = 140;
   stru.decimal = 150;
   strcpy(stru.name, "Mike");
     printf("\ninteger: %d", stru.integer);
   printf("\ndecimal: %f", stru.decimal);
   printf("\nname: %s", stru.name);
     printf("\ndata of union:");
   uni.integer = 140;
   uni.decimal = 150;
   strcpy(uni.name, "Mike");
     printf("\ninteger: %d", uni.integer);
   printf("\ndecimal: %f", uni.decimal);
   printf("\nname: %s", uni.name);
       //difference four
   printf("\nAltering a member value:\n");
   stru.integer = 512;
   printf("data of structure:\n integer: %d\n decimal: %.2f\n name: %s\n", stru.integer, stru.decimal, stru.name);
   uni.integer = 512;
   printf("data of union:\n integer: %d\n decimal: %.2f\n name: %s\n", uni.integer, uni.decimal, uni.name);
      // difference two and three
   printf("\nsizeof structure: %d\n", sizeof(stru));
   printf("sizeof union: %d\n", sizeof(uni));
```

```
}
```

Output

```
Accessing all members at a time:
data of structure:
 integer: 163
 decimal: 75.000000
 name: John

data of union:
 integeer: 1852337994
 decimal: 17983765624912253034071851008.000000
 name: John

Accessing one member at a time:
data of structure:
integer: 140
decimal: 150.000000
name: Mike
data of union:
integer: 1701538125
decimal: 69481161252302940536832.000000
name: Mike
Altering a member value:
data of structure:
 integer: 512
 decimal: 150.00
 name: Mike
data of union:
 integer: 512
 decimal: 0.00
 name:

sizeof structure: 28
sizeof union: 20
```

iii) Write a C program to shift/rotate using bitfields.

Theory

What is meant by rotating bits of a number

Rotating bits of a number to left, means shifting all bits to left and pushing the dropped Most Significant Bit to Least Significant Bit.

Rotating bits of a number to right, means shifting all bits to right and pushing the dropped Least Significant Bit to Most Significant Bit.

Rotating bits of -15 two times to right.

There is small difference in logic of left and right rotation of number. I have explained logic of both problems separately.

Algorithm to left rotate bits in a number

Left rotation of bits in C is supported using bitwise left shift operator <<. But left shift operator drops Most Significant Bit (MSB) on each shift. Which is what we don't want. Instead of dropping MSB on each rotation, Least Significant Bit (LSB) should get replaced as dropped MSB.

Step by step descriptive logic to left rotate bits of a number.

1. Input number to rotate and rotation amount from user. Store it in some variable say num and rotation.
2. Find most significant bit of the number that is about to drop and store it in some variable say DROPPED_MSB = (num >> INT_BITS) & 1.
3. Left Shift num one times and set its least significant bit to DROPPED_MSB i.e. perform num = (num << 1) | DROPPED_MSB.
4. Repeat step 2 and 3 for given number of rotations.

Algorithm to right rotate bits of a number

Right rotation of bits in C programming is supported using bitwise right shift operator >>. Similar to left shift, right shift operations also results in bit loss. On every shift operation the least significant bit is dropped.

What we need to do is, instead of dropping the least significant bit replace most significant bit with the dropped least significant bit.

Step by step descriptive logic to right rotate bits of a number.

1. Input number to rotate and rotation amount from user. Store it in some variable say num and rotation.
2. Find least significant bit of the number about to drop and store it in some variable say DROPPED_LSB = num & 1.
3. Right shift number one time i.e. perform num = num >> 1.
4. Clear most significant bit of number i.e. perform num = num & (~(1 << INT_BITS)).
5. Set the DROPPED_LSB as most significant bit of num. Perform num = num | (DROPPED_LSB << INT_BITS).
6. Repeat step 2 to 5 for given number of rotations.

Note: INT_BITS *is total number of bits in integer – 1.*

Example: Input data is 252. 3 be the number of bits to be rotated.
Binary Equivalent of 252 is 1111 1100

Left Rotation: Left rotate 252 by 3 bits.
Find 252 << 3 => (1111 1100 << 3) => 1110 0000
Find 252 >> (8 -3) => (1111 1100 >> 5) => 0000 0111

255 & ((252 << 3) | (252 >> 5)) = 1110 0111 => 231
Binary equivalent of 255 is 1111 1111. 255 is used in above manipulation to get 8 bit output.

Right Rotation: Right rotate 252 by 3 bits.
Find 252 >> 3 => (1111 1100 >> 3) => 0001 1111
Find 252 << (8 - 3) => 1111 1100 << 5 => 1000 0000

255 & ((252 >> 3) | (252 << 5)) => 1001 1111 => 159

Source Code

```
#include <stdio.h>
  int main()
{
        int value, n, lRotate, rRotate;
        /* get 8 bit input from the user */
        printf("Enter your input value(0-255):");
        scanf("%d", &value);
        /* threshold checking */
        if (value < 0 || value > 255) {
                printf("Threshold Value Exceeded!!\n");
                return 0;
        }

        /* number of bits to rotate */
        printf("No of bits to rotate(0-7):");
        scanf("%d", &n);
        if (n < 0 || n > 7) {
                printf("Threshold Value Exceeded!!\n");
                return 0;
        }
```

```
    /* left rotation */
    lRotate = 255 & ((value << n) | (value >> (8 - n)));
    /* right rotation */
    rRotate = 255 & ((value >> n) | (value << (8 - n)));
    /* print the results */
    printf("Left Rotation: %d\n", lRotate);
    printf("Right Rotation: %d\n", rRotate);

    return 0;
}
```

Output

```
Enter your input value(0-255):200
No of bits to rotate(0-7):5
Left Rotation: 25
Right Rotation: 70
```

iv) Write a C program to copy one structure variable to another structure of the same type.

Theory

Copying and Comparing Structure Variables

Two variables of the same structure type can be copied the same way as ordinary variables.
If e1 and e2 belong to the same type, then the following statement is valid. e1 = e2, and e2 = e1;
However, the statements that are shown here: e1 < e2; and e1 != e2; are not permitted.
C language doesn't permit any logical operations on structure variables.
We can compare two structure variables but comparison of members of a structure can only be done individually.

Algorithm

1. Declaring and initializing structures of 'class' type
2. Copying student2 to student3
3. verifying results of copy

Source Code

```
#include<stdio.h>
#include<conio.h>
struct class
{
int number; char name[20];
float marks;
};
void main()
{
int x;
struct class student2 = {2, "gita", 78.00};
struct class student3;
student3 = student2; // Copying student2 to student3
if ((student3.number = student2.number) && (student3.marks = student2.marks)) {
printf("\n student2 and student3 are equal");
printf("%d %s %f\n", student3.number, student3.name, student3.marks);
}
else
printf("\n student2 and student3 are different");
}
```

Output

```
student2 and student3 are equal2 gita 78.000000
```

UNIT V
WEEK 11

Objective: Explore the Functions, sub-routines, scope and extent of variables, doing some experiments by parameter passing using call by value. Basic methods of numerical integration
Suggested Experiments/Activities:
Tutorial 11: Functions, call by value, scope and extent,

Lab 11: Simple functions using call by value, solving differential equations using Eulers theorem.

i) Write a C function to calculate NCR value.

Algorithm

1. Initially, the program will prompt the user to enter the values of **n** and **r**.
2. Now, we invoke the function int nCr(int n, int r), within this function, we call int fact(int n) function to calculate the required factorial values.
3. Factorial values are used to calculate the nCr value and then we print this value on the console.

Source Code

```
#include <stdio.h>
int fact(int n);
int nCr(int n, int r)
{
 return fact(n) / (fact(r) * fact(n - r));
 }
int fact(int n)
{
  int res = 1;
  for (int i = 2; i <= n; i++)
{
    res = res * i;
  }
  return res;
}
int main()
{
  int n, r;
  printf("Enter the value of n: ");
  scanf("%d", &n);
  printf("Enter the value of r: ");
  scanf("%d", &r);
  printf("The value of %dC%d is %d", n, r, nCr(n, r));
  return 0;
}
```

Output

```
Enter the value of n: 9
Enter the value of r: 9
The value of 9C9 is 1
```

ii) Write a C function to find the length of a string.

Algorithm

1. Start
2. Declare a string and len
3. strlen() is the pre-defined function to find the length of a string. Use it and print the result.
4. Stop

Source Code

```
#include<stdio.h>
#include<string.h>
 int main()
{
   char str[100];
   int len;
      printf("\nEnter the String : ");
   scanf("%s",str);
      /*
      strlen() is the pre-defined function  to find the length of a string
   */
   len = strlen(str);
      printf("\nLength of the given string is %d", len);
   return(0);
}
```

Output

```
Enter the String : keerthi
Length of the given string is 7
```

iii) Write a C function to transpose of a matrix.

Algorithm

1. Start
2. Declare an array.
3. Initialize the array.
4. Declare a transpose matrix.
5. Call a function that will perform the transpose operation.
6. Store the elements in the transpose matrix.
7. Now, print the elements in the transpose matrix.
8. Stop

Source Code

```
#include <stdio.h>
int main()
 {
   int m, n, i, j;
   // Requesting the dimensions of the matrix from the user
   printf("Enter the number of rows and columns of the matrix: ");
   scanf("%d%d", &m, &n);
   int A[m][n], Transposed[n][m];
   // Capturing the matrix elements from the user
   printf("Enter the elements of the matrix:\n");
   for(i = 0; i < m; i++) {
      for(j = 0; j < n; j++) {
         printf("A[%d][%d] = ", i+1, j+1);
         scanf("%d", &A[i][j]);
      }
   }
   // Calculating the transpose
   for(i = 0; i < m; i++) {
      for(j = 0; j < n; j++) {
         Transposed[j][i] = A[i][j];
      }
   }
   // Displaying the transpose
   printf("\nTranspose of the matrix is:\n");
   for(i = 0; i < n; i++) {
      for(j = 0; j < m; j++) {
         printf("%d  ", Transposed[i][j]);
```

```
        }
        printf("\n");
    }

    return 0;
}
```

Output

```
Enter the number of rows and columns of the matrix: 3
2
Enter the elements of the matrix:
A[1][1] = 23
A[1][2] = 12
A[2][1] = 45
A[2][2] = 67
A[3][1] = 56
A[3][2] = 78

Transpose of the matrix is:
23  45  56
12  67  78
```

iv) Write a C function to demonstrate numerical integration of differential equations using Euler's method

Euler Method for solving differential equation

Given a differential equation dy/dx = f(x, y) with initial condition y(x0) = y0. Find its approximate solution using Euler method.

Euler Method :

In mathematics and computational science, the Euler method (also called forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value.

Consider a differential equation dy/dx = f(x, y) with initial condition y(x0)=y0

then a successive approximation of this equation can be given by:

y(n+1) = y(n) + h * f(x(n), y(n))

where h = (x(n) – x(0)) / n

h indicates step size. Choosing smaller values of h leads to more accurate results and more computation time.

Example :

Consider below differential equation

dy/dx = (x + y + xy)

with initial condition y(0) = 1

and step size h = 0.025.

Find y(0.1).

Solution:

f(x, y) = (x + y + xy)

x0 = 0, y0 = 1, h = 0.025

Now we can calculate y1 using Euler formula

y1 = y0 + h * f(x0, y0)

y1 = 1 + 0.025 *(0 + 1 + 0 * 1)

y1 = 1.025

y(0.025) = 1.025.

Similarly we can calculate y(0.050), y(0.075), ...y(0.1).

y(0.1) = 1.11167

Algorithm:

1. Enter the initial values of and (xi and yi respectively).
2. Enter the value of , for which is to be determined.
3. Enter the width of the interval, ‘ ’.

```
4. Do:y=y0+(h*dy/dx(xi,yi))
   yi=y.
   xi=xi+h
   Until (xi>=x)
5. Print y, which is the solution.
```

Source Code

```
#include<stdio.h>
float fun(float x,float y)
{
   float f;
   f=x+y;
   return f;
}
void main()
{
   float a,b,x,y,h,t,k;
   printf("\nEnter x0(initial value of x),y0(initial value of y),h(step-width ),xn(value of x for which y is required): ");
   scanf("%f%f%f%f",&a,&b,&h,&t);
   x=a;
   y=b;
   printf("\n  x\t  y\n");
   while(x<=t)
   {
      k=h*fun(x,y);
      y=y+k;
      x=x+h;
      printf("%0.3f\t%0.3f\n",x,y);
   }
}
```

Output

```
Enter x0(initial value of x),y0(initial value of y),h(step-width ),xn(value of x for which y is required):
1
1
3
1
  x      y
4.000   7.000
```

WEEK 12

Objective: Explore how recursive solutions can be programmed by writing recursive functions that can be invoked from the main by programming at-least five distinct problems that have naturally recursive solutions.

Suggested Experiments/Activities:

Tutorial 12: Recursion, the structure of recursive calls

Lab 12: Recursive functions

i) Write a recursive function to generate Fibonacci series.

Algorithm

1. When the input n is 0 or 1, we return n itself, as these are the first two numbers in the Fibonacci series.
2. In the recursive step, if n is greater than 1, we recursively call the fibonacci() function with n-1 and n-2. These calls continue until the base cases are reached.
3. When the base cases are reached (i.e., n is 0 or 1), we add the results of the recursive calls for n-1 and n-2 to calculate the Fibonacci number for n.
4. Finally, we return the calculated Fibonacci number for the input n.
5. In the main() , we prompt users to enter the number of current terms they want in the Fibonacci series. Then, we call the recursive fibonacci() function for each term and print the series. as it's generated.

Source Code

```
#include <stdio.h>
void printFibonacci(int n)
{
  int n1 = 0, n2 = 1, n3;
  printf("%d %d ", n1, n2); // printing 0 and 1
  for(int i=2; i<n; i++)
{
    n3 = n1 + n2;
    printf("%d ", n3);
    n1 = n2;
    n2 = n3;
  }
}
int main()
{
  int n;
  printf("Enter the number of elements: ");
  scanf("%d", &n);
```

```
   printf("Fibonacci Series: ");
   printFibonacci(n);
     return 0;
}
```

Output

```
Enter the number of elements: 5
Fibonacci Series: 0 1 1 2 3
```

ii) Write a recursive function to find the lcm of two numbers.

Algorithm

1. Include the required header files using include keyword.
2. define a function lcm() which returns an integer value and takes num1 and num2 as arguments.
3. Set a base case which gets satisfied when the remainder after dividing num1 and num2 with the static variable smallestcm is zero repectively.
4. Until the base case is satisfied, keep calling the recursive function and and increment smallestcm by 1.
5. Return the smallestcm when the base case is satisfied.
6. In the int main section initialize all the required variables and print the returned value after calling the lcm(n1,n2) function using printf() function.

Source Code

```
#include <stdio.h>
int lcm(int n1, int n2) {
   static int lowestcm = 1;
   if(lowestcm%n1 == 0 && lowestcm%n2 == 0)
   {
      return lowestcm;
   }
   else
   {
      lowestcm++;
      lcm(n1,n2);
      return lowestcm;
   }
}
int main() {
   int n1, n2;
   printf("enter two values to find lcm");
   scanf("%d  %d" , &n1, &n2);
   printf("L.C.M of %d and %d is %d.", n1, n2, lcm(n1, n2));
   return 0;
}
```

Output

```
enter two values to find lcm3
4
L.C.M of 3 and 4 is 12.
```

iii) Write a recursive function to find the factorial of a number.

Algorithm

1. Get the input from the user, by using the entered value the fact() is called,
2. The n-1 value is passed to fact() from the function,
3. Every time the function is called the n value is decremented by 1,
4. Once the value of n is 1, the recursive function will be stopped and send the value to the main() function.

Source Code

```
#include<stdio.h>
long int fact(int x);
int main()
{
   int x;
   printf("Enter A Number To Find Factorial: ");
   scanf("%d",&x);
   printf("The Factorial of %d = %ld", x, fact(x));
   return 0;
}

long int fact(int x)
{
   if (x>=1)
      return x*fact(x-1);
   else
      return 1;
}
```

Output

```
Enter A Number To Find Factorial: 5
The Factorial of 5 = 120
```

iv) Write a C Program to implement Ackermann function using recursion.

Theory

The Ackermann function is a two-parameter function that takes non-negative integers as inputs and produces a non-negative integer as its result. While it may seem deceptively simple, this function exhibits astonishing growth rates, rapidly surpassing the capabilities of traditional computational methods.

What is the Ackermann Function in C?

The Ackermann function, named after the German mathematician Wilhelm Ackermann, is a recursive mathematical function that takes two non-negative integers as inputs and produces a non-negative integer as its output. In C, the Ackermann function can be implemented using recursion.

Ackermann function is defined as:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

The Ackerman recursion can be described as below: A (0, n)
= n + 1
A (m + 1, 0) = A (m, 1)
A (m + 1, n + 1) = A (m, A (m + 1, n))
Given m and n as input, write a program to calculate A(m, n), using recursion.

Algorithm

1. Start
2. Create a function Ackerman(int m,int n);
3. In the main(), prompt the user to enter 2 numbers
4. Recursevily call the function.
5. Inside the function perform ,
6. The Ackerman recursion can be described as below:
   A (0, n)= n + 1
   A (m + 1, 0) = A (m, 1)
   A (m + 1, n + 1) = A (m, A (m + 1, n))
7. Print the result

Source Code

```
#include<stdio.h>
int Ackerman(int m,int n);
int main()
{
   int m,n;
   printf("Enter two numbers: ");
```

```
    scanf("%d%d",&m,&n);
    printf("Ackerman(%d,%d) = %d.\n",m,n,Ackerman(m,n));
    return 0;
}
int Ackerman(int m,int n)
{
    if(m==0)
        return n+1;
    if(n==0 && m>0)
        return Ackerman(m-1,1);
    if(m>0 && n>0)
        return Ackerman(m-1,Ackerman(m,n-1));
}
```

Output

```
Enter two numbers: 3  4
Ackerman(3,4) = 125.
```

v) Write a recursive function to find the sum of series.

Algorithm

1. First give a meaningful name to the function, say sumOfNaturalNumbers().
2. Next the function must accept two inputs i.e. the lower and upper limit to find sum. Hence, pass two integer parameters to the function say sumOfNaturalNumbers(int start, int end).
3. Finally, the function must return sum of natural numbers between *start* and *end*. Therefore return type of function should be int.

Source Code

```
#include <stdio.h>
/* Function declaration */
int sumOfNaturalNumbers(int start, int end);
int main()
{
   int start, end, sum;
      /* Input lower and upper limit from user */
   printf("Enter lower limit: ");
   scanf("%d", &start);
   printf("Enter upper limit: ");
   scanf("%d", &end);
      sum = sumOfNaturalNumbers(start, end);
      printf("Sum of natural numbers from %d to %d = %d", start, end, sum);
      return 0;
}
//Recursively find the sum of natural number
int sumOfNaturalNumbers(int start, int end)
{
   if(start == end)
      return start;
   else
      return start + sumOfNaturalNumbers(start + 1, end);
}
```

Output

```
Enter lower limit: 5
Enter upper limit: 9
Sum of natural numbers from 5 to 9 = 35
```

WEEK 13

Objective: Explore the basic difference between normal and pointer variables, Arithmetic operations using pointers and passing variables to functions using pointers

Suggested Experiments/Activities:

Tutorial 13: Call by reference, dangling pointers

Lab 13: Simple functions using Call by reference, Dangling pointers.

i) Write a C program to swap two numbers using call by reference.

Algorithm

1. Copy the value of first number say num1 to some temporary variable say temp.
2. Copy the value of second number say num2 to the first number. Which is num1 = num2.
3. Copy back the value of first number stored in temp to second number. Which is num2 = temp.

Source Code

```
#include <stdio.h>
/* Swap function declaration */
void swap(int * num1, int * num2);
int main()
{
    int num1, num2;
    /* Input numbers */
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    /* Print original values of num1 and num2 */
    printf("Before swapping in main \n");
    printf("Value of num1 = %d \n", num1);
    printf("Value of num2 = %d \n\n", num2);
    /* Pass the addresses of num1 and num2 */
    swap(&num1, &num2);
    /* Print the swapped values of num1 and num2 */
    printf("After swapping in main \n");
    printf("Value of num1 = %d \n", num1);
    printf("Value of num2 = %d \n\n", num2);
    return 0;
}
//  Function to swap two numbers
 void swap(int * num1, int * num2)
{
    int temp;
    // Copy the value of num1 to some temp variable
    temp = *num1;
    // Copy the value of num2 to num1
    *num1= *num2;

    // Copy the value of num1 stored in temp to num2
    *num2= temp;
}
```

Output

```
Enter two numbers: 5
9
Before swapping in main
Value of num1 = 5
Value of num2 = 9

After swapping in main
Value of num1 = 9
Value of num2 = 5
```

ii) Demonstrate Dangling pointer problem using a C program.

A dangling pointer occurs at the time of the object destruction when the object is deleted or de-allocated from memory without modifying the value of the pointer. In this case, the pointer is pointing to the memory, which is de-allocated.

In C Programming Language, there are three different ways where we can make a pointer acts as a dangling pointer. They are as follows:

- **Deallocation of Memory:** When the memory to which the pointer was pointing is freed or deallocated, the pointer is not set to NULL. Thus, it still points to the memory location, which is no longer valid. So, setting the pointer to NULL after freeing it is important to avoid this issue.
- **Function Call:** When a pointer points to a local variable in a function, and the function call is over. As local variables get destroyed after a function call, the pointer pointing to such a variable becomes dangling.
- **Variable Goes Out of Scope:** When a pointer points to a variable, and the variable goes out of scope, It becomes a dangling pointer.

1. Deallocation of Memory

Algorithm

1. An integer pointer ptr points to an integer variable with value 10, ptr contains the address of the variable allocated dynamically using malloc() method.
2. When the integer variable gets deallocated from memory using the free(ptr); function, ptr points to some garbage value i.e. invalid location/data and acts as a dangling pointer.

Source Code

```
#include <stdlib.h>
int main() {
   // 4 bytes of int memory block (64bit compiler)
   // allocated using malloc() during runtime
   int *ptr = (int *)malloc(sizeof(int)); // normal pointer
   *ptr = 10;
  // memory block deallocated using free() function
   free(ptr);
   // here ptr acts as a dangling pointer
   printf("%d", *ptr);
   return 0;
}
```

Output

```
1567999163
```

2. Function Call

Algorithm

1. A function() is called inside the main() function, memory is allocated by the system for the function() block.
2. A local variable temp is declared and initialized inside the function(). Let the address of temp is 2000. After returning the address of the temp variable function execution finishes and temp also gets deleted from the memory.
2. Returned address 2000 is stored in ptr pointer but as temp is not there in the memory anymore, ptr points to some garbage value and acts as a dangling pointer.

Source Code

```
#include <stdio.h>
// definition of danglingPointer() function
int *danglingPointer() {
   // temp variable has local scope
   int temp = 10;
   // returning address of temp variable
   return &temp;
}
int main()
{
// ptr will point to some garbage value as temp variable will be destroyed after the execution of
// below line
   int *ptr = danglingPointer();
   // ptr is a Dangling Pointer now  ptr contains some random address and
   // is pointing to some garbage value
   printf("%d", *ptr);

   return 0;
}
```

Output

```
main.c: In function ‘danglingPointer’:
main.c:16:12: warning: function returns address of local variable [-Wreturn-local-addr]
  16 |     return &temp;
```

3. Variable goes out of scope

Algorithm

1. A pointer ptr is declared in the main() function, it is acting as a wild pointer.
2. When we enter the inner block of code, ptr points to the temp variable having value 10. temp has local scope and will be deleted from the memory as soon as the program control moves out of the inner block.
3. temp goes out of scope and ptr still contains the address of deleted memory. So, ptr will point to some garbage value and will act as a dangling pointer.

Source Code

```
// Variable goes out of scope
#include <stdio.h>
int main()  {
   // A pointer that has not been initialized is
   // known as a Wild Pointer, ptr is a Wild Pointer.
   int *ptr;
   // variables declared inside the block of will get destroyed
   // at the end of execution of this block
   {
      int temp = 10;
      ptr = &temp; // acting as normal pointer
   }
   // temp is now removed from the memory (out of scope)
   // now ptr is a dangling pointer
   printf("%d %d", *ptr, temp);

   // as temp is not in the memory anymore so it can't be modified using ptr

   // prints garbage value
   printf("%d", *ptr);
   return 0;
}
```

Output

```
main.c: In function ‘main’:
main.c:25:27: error: ‘temp’ undeclared (first use in this function)
   25 |     printf("%d %d", *ptr, temp);
      |                           ^~~~
main.c:25:27: note: each undeclared identifier is reported only once for each function it appears in
```

iii) Write a C program to copy one string into another using pointer.

Algorithm

1. Start
2. Declare the variable str1 and str2.
3. Read input str1 from the user or predefine it according to the need.
4. Use the syntax to copy the string into the empty string.
5. Print the result of the program compiled.
6. Program end.

Source Code

```
#include <stdio.h>
void stringCopy(char *dest, const char *src) {
   // Copy characters from src to dest
   while (*src != '\0') {
      *dest = *src;
      src++;
      dest++;
   }
   // Add null terminator to the destination string
   *dest = '\0';
}

int main() {
   char source[100] ;
   char destination[20]; // Make sure the destination array has enough space
printf("enter a string");
 scanf("%s", source);
   // Copy the string using pointers
   stringCopy(destination, source);

   printf("Original string: %s\n", source);
   printf("Copied string: %s", destination);

   return 0;
}
```

Output

```
enter a string happy
Original string: happy
Copied string: happy
```

iv) Write a C program to find no of lowercase, uppercase, digits and other characters using pointers.

Algorithm

1. Scan string str from 0 to length-1.
2. check one character at a time based on ASCII values
   - if(str[i] >= ‘A’ and str[i] <=’Z’), then it is uppercase letter,
   - if(str[i] >= ‘a’ and str[i] <=’z’), then it is lowercase letter,
   - if(str[i] >= ‘0’ and str[i] <=’9’), then it is number,
   - else it is a special character
3. Print all the counters

Source Code

```
#include<stdio.h>
 int main()
{
    char str[100];
    int i,a=0, b=0,d=0,s=0;
    printf("\nEnter The String : ");
    scanf("%s", str);
    for(i=0; str[i]!='\0'; i++)
    {
        if(str[i]>='A'&&str[i]<='Z')                a++;
            else if(str[i]>='a' &&str[i]<='z')      b++;
        else if (str[i]>='0' &&str[i]<='9')         d++;
        else
            s++;
    }
    printf("\nTotal Alphabets  (upper)  : %d",a);
     printf("\nTotal Alphabets  (lower)  : %d",b);
    printf("\nTotal Digits      : %d",d);
    printf("\nTotal Special      : %d",s);
    return 0;
}
```

Output

```
Enter The String : Happy@123
Total Alphabets  (upper)  : 1
Total Alphabets  (lower)  : 4
Total Digits      : 3
Total Special      : 1
```

WEEK14

Objective: To understand data files and file handling with various file I/O functions. Explore the differences between text and binary files.

Tutorial 14: File handling

Lab 14: File operations

i) Write a C program to write and read text into a file.

Algorithm

1. Open the file in reading mode using fopen()
2. Using fprintf, write anything in the file.
3. Once done, close the file pointer using fclose.
4. Open the file in reading mode using fopen()
5. In a loop, read each character from the file one by one using fgetc and print it.
6. If end of file is reached, then break the loop. This can be checked using feof().

Source Code

```c
#include <stdio.h>
int main ()
{
   FILE * fp;
  int c;
   fp = fopen ("file-content.txt", "a+");
   fprintf(fp, "This is a sample line ");
   fclose(fp);
  fp = fopen("file-content.txt","r");
   while(1) {
      c = fgetc(fp);
      if( feof(fp) ) {
         break ;
      }
      printf("%c", c);
   }
      fclose(fp);
      return 0;
}
```

Output

```
This is a sample line
file-content.txt
This is a sample line
```

ii) Write a C program to write and read text into a binary file using fread() and fwrite()

Algorithm

1. Open the file in reading mode using fopen()
2. Using fwrite, write anything in the file.
3. Once done, close the file pointer using fclose.
4. Open the file in reading mode using fopen()
5. In a loop, read each character from the file one by one using fread and print it.
6. If end of file is reached, then break the loop. This can be checked using feof().

Source Code

```
#include<stdio.h>
#include<string.h>
struct student{
  int sno;
  char sname [30];
  float marks;
  char temp;
};
void main ( ){
  struct student s[60];
  int i,n;
  FILE *fp;
  fp = fopen ("student1.txt", "w");
  printf("enter total students");
  scanf("%d", &n);
  for (i=0; i<n; i++){
    printf ("enter details of student %d \n", i+1);
    printf("student number:");
    scanf("%d",&s[i].sno);

    printf("student name:");
    scanf("%s", s[i].sname);
    printf("student marks:");
    scanf("%f",&s[i].marks);
    fwrite(&s[i], sizeof(s[i]),1,fp);
  }
  fclose (fp);
  fp = fopen ("student1.txt", "r");
  for (i=0; i<n; i++){
    printf ("details of student %d are \n", i+1);
    fread (&s[i], sizeof (s[i]) ,1,fp);
    printf("student number = %d \n", s[i]. sno);
    printf("student name = %s \n", s[i]. sname);
    printf("marks = %f \n", s[i]. marks);
  }
  fclose(fp);

}
```

Output

```
enter total students2
enter details of student 1
student number:1
student name:happy
student marks:99
enter details of student 2
student number:2
student name:sraavas
student marks:98
details of student 1 are
student number = 1
student name = happy
marks = 99.000000
details of student 2 are
student number = 2
student name = sraavas
marks = 98.000000
```

student1.txt

iii) Copy the contents of one file to another file.

Algorithm

1. Input file path of source and destination file.
2. Open source file in r (read) and destination file in w (write) mode.
3. Read character from source file and write it to destination file using fputc().
4. Repeat step 3 till source file has reached end.
5. Close both source and destination file.

Source Code

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   FILE *sourceFile;
   FILE *destFile;
   char sourcePath[100];
   char destPath[100];
   char ch;
   /* Input path of files to copy */
   printf("Enter source file path: ");
   scanf("%s", sourcePath);
   printf("Enter destination file path: ");
   scanf("%s", destPath);

   /*  Open source file in 'r' and   destination file in 'w' mode     */
   sourceFile  = fopen(sourcePath, "r");
   destFile    = fopen(destPath,   "w");

   /* fopen() return NULL if unable to open file in given mode. */
   if (sourceFile == NULL || destFile == NULL)
   {
      /* Unable to open file hence exit */
      printf("\nUnable to open file.\n");
      printf("Please check if file exists and you have read/write privilege.\n");
      exit(EXIT_FAILURE);
   }
   /*  Copy file contents character by character.    */
   ch = fgetc(sourceFile);
   while (ch != EOF)
   {
        /* Write to destination file */
        fputc(ch, destFile);

        /* Read next character from source file */
        ch = fgetc(sourceFile);
    }
    printf("\nFiles copied successfully.\n");
    /* Finally close files to release resources */
    fclose(sourceFile);
    fclose(destFile);

    return 0;
}
```

Output

```
Enter source file path: abc.txt
Enter destination file path: xyz.txt

Files copied successfully.
abc.txt
            happy
xyz.txt
            happy
```

iv) Write a C program to merge two files into the third file using command-line arguments.

Algorithm

1. Open file1.txt and file2.txt in read mode.
2. Open file3.txt in write mode.
3. Run a loop to one by one copy characters of file1.txt to file3.txt.
4. Run a loop to one by one copy characters of file2.txt to file3.txt.
5. Close all files.

Source Code

```
void main(int argc, char *argv[])
{
FILE *f1,*f2,*f3;
intn,i,m,count,reminder;
char s[50];
printf("Please enter a number for writing in to a file");
scanf("%d",&n);
f1=fopen(argv[1],"w");
if(n%2==0)
{
fprintf(f1,"Entered number is an even number",n);
}
else
{
fprintf(f1,"Entered number is an odd number",n);
}
fclose(f1);
count=0;
for(i=1;i<=n;i++)
{
reminder=n%i;
if(reminder==0)
count=count+1;
}
f1=fopen(argv[2],"w");
if(count==2)
{
fprintf(f1,"Entered number is a prime number",n);
}
else
{
fprintf(f1,”Entered number  is not a prime number”,n);
fclose(f1);
f1=fopen(argv[1],”r”);
f2=fopen(argv[2],”r”);
f3=fopen(“filefor merging.txt”],”w”);
fgets(s,50,f1);
fputs(s,f3);
fgets(s,50,f2);
fputs(s,f3);
}
```

Output

The output for the above program is explained below.

```
peofn1.txt n2.txt
Enter a number: 11
typen1.txt
Entered number is an odd number
typen2.txt
Entered number is a prime number
typefileformerge.txt
Entered number is an odd number
Entered number is a prime number
```

v) Find no. of lines, words and characters in a file

Algorithm

1. Open source file in r (read) mode.
2. Initialize three variables characters = 0, words = 0 and lines = 0 to store counts.
3. Read a character from file and store it to some variable say ch.
4. Increment characters count.
   Increment words count if current character is whitespace character i.e. if (ch == ' ' || ch == '\t' || ch == '\n' || ch == '\0').
   Increment lines count if current character is new line character i.e. if (ch == '\n' || ch == '\0').
5. Repeat step 3-4 till file has reached end.
6. Finally after file has reached end increment words and lines count by one if total characters > 0 to make sure you count last word and line.

Source Code

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   FILE * file;
   char path[100];
   char ch;
   int characters, words, lines;
   /* Input path of files to merge to third file */
   printf("Enter source file path: ");
   scanf("%s", path);

   /* Open source files in 'r' mode */
   file = fopen(path, "r");

   /* Check if file opened successfully */
   if (file == NULL)
   {
      printf("\nUnable to open file.\n");
      printf("Please check if file exists and you have read privilege.\n");
      exit(EXIT_FAILURE);
   }

   /* Logic to count characters, words and lines.      */
   characters = words = lines = 0;
```

```
    while ((ch = fgetc(file)) != EOF)
    {
        characters++;

        /* Check new line */
        if (ch == '\n' || ch == '\0')
            lines++;

        /* Check words */
        if (ch == ' ' || ch == '\t' || ch == '\n' || ch == '\0')
            words++;
    }

    /* Increment words and lines for last word */
    if (characters > 0)
    {
        words++;
        lines++;
    }
    /* Print file statistics */
    printf("\n");
    printf("Total characters = %d\n", characters);
    printf("Total words      = %d\n", words);
    printf("Total lines      = %d\n", lines);

    /* Close files to release resources */
    fclose(file);

    return 0;
}
```

Output

```
Enter source file path: main.c
Total characters = 1683
Total words      = 522
Total lines      = 73
```

vi) Write a C program to print last n characters of a given file.

Algorithm

1. Firstly open file in the read mode.fp=fopen("test.txt","r");
2. Now we need to accept position number so that we can start reading from that position. We are moving file pointer to the last location using fseek() function and passing SEEK_END constant. fseek(fp,0,SEEK_END);
3. Now we need to evaluate the current position of the file pointer. len = ftell(fp);
4. ftell() will tell you the location of file pointer.
5. File Location = Total Number of Characters on File
6. We need to read last n characters from the file so we need to move pointer to (length-n) character back on the file. and from that location we need to read file.
7. fseek(fp,(len-n),SEEK_SET);

Source Code

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
FILE *fp;
char ch;
int n;
long len;
printf("Enter the value of n : ");
scanf("%d",&n);
fp=fopen("main.c","r");
if(fp==NULL)
    {
    puts("cannot open this file");
    exit(1);
    }
fseek(fp,0,SEEK_END);
len = ftell(fp);
fseek(fp,(len-n),SEEK_SET);
do {
  ch = fgetc(fp);
  putchar(ch);
}while(ch!=EOF);
fclose(fp);
}
```

Output

```
Enter the value of n : 50
fp);
  putchar(ch); }while(ch!=EOF); }
```

Textbooks:
1. Ajay Mittal, Programming in C: A practical approach, Pearson.
2. Byron Gottfried, Schaum&#39; s Outline of Programming with C, McGraw Hill

Reference Books:
1. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice- Hall of India
2. C Programming, A Problem-Solving Approach, Forouzan, Gilberg, Prasad, CENGAGE