



Savitribai Phule Pune University
(Formerly University of Pune)

F.Y.B.Sc.(Computer Science)

With

Major: Computer Science

(Faculty of Science and Technology)

(For Colleges Affiliated to Savitribai Phule Pune University)

Choice Based Credit System (CBCS) Syllabus Under
National Education Policy (NEP)

To be implemented from Academic Year 2024-2025

Title of the Course: B.Sc.(Computer Science)

F.Y.B.Sc.(Computer Science)

Semester-I

Lab Course – I

Work Book

Name: _____

College Name: _____

Roll No.: _____

Division: _____

Academic Year : _____

BOARD OF STUDIES

- | | |
|---------------------------|---------------------------|
| 1. Dr. Patil Ranjeet | 2. Dr. Joshi vinayak |
| 3. Dr. Wani Vilas | 4. Dr. Mulay Prashant |
| 5. Dr. Sardesai Anjali | 6. Dr. Shelar Madhukar |
| 7. Dr. Bharambe Manisha | 8. Dr. Deshpande Madhuri |
| 9. Dr. Kardile Vilas | 10. Dr. Korpai Ritambhara |
| 11. Dr. Gangurde Rajendra | 12. Dr. Manza Ramesh |
| 13. Dr. Bachav Archana | 14. Dr. Bhat Shridhar |

Co-ordinators

- **Dr. Prashant Mulay**, Annasaheb Magar College, Hadapsar , Pune.
Member, BOS Computer Science, Savitribai Phule Pune University
- **Dr. Sardesai Anjali** , Modern College of Arts, Science and Commerce ,
Shivajinagar, Pune
Member, BOS Computer Science, Savitribai Phule Pune University

Editor

Dr. Prashant Mulay, Annasaheb Magar College, Hadapsar , Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

Prepared by:

Ms. Gadekar Manisha Jankiram.	Annasaheb Magar College, Hadapsar, Pune.
--	--

Introduction

About the work book:

This LAB book / Workbook is intended to be used by F.Y.B.Sc. (Computer Science) students for the C Assignments in Semester–I. This workbook is designed by considering all the practical concepts / topics mentioned in syllabus. The lab book is to be used as a hands-on resource, reference and record of assignment submission and completion by the student. The lab book contains the set of assignments which the student must complete as a part of this course.

The objectives of this LAB-Book are:

Defining the scope of the course.

- To bring uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
- To have continuous assessment of the course and students.
- Providing ready reference for the students during practical implementation.
- Provide more options to students so that they can have good practice before facing the examination.
- Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

Instructions to the students

- Students are expected to carry this book every time they come to the lab for computer science practical.
- Students should prepare oneself beforehand for the Assignment by reading the relevant material.
- Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor.
- However student should spend additional hours in Lab and at home to cover as many problems as possible given in this work book.

Submission:

Problem Solving Assignments:

- The problem solving assignments are to be submitted by the student in the form of a journal containing individual assignment sheets.
- Each assignment includes the Assignment Title, Problem statement, Date of submission,
- Assessment date, Assessment grade and instructors sign.

Programming Assignments:

Programs should be done individually by the student in the respective login.

The codes should be uploaded on either the local server, Moodle, Github or any open source LMS. Print-outs of the programs and output may be taken but not mandatory for assessment.

Assessment:

Continuous assessment of laboratory work is to be done based on overall performance and lab assignments performance of student.

Each lab assignment assessment will be assigned grade/marks based on parameters with appropriate weightage.

Suggested parameters for overall assessment as well as each lab assignment assessment include- timely completion, performance, innovation, efficient codes and good programming practices.

Operating Environment:

For 'C' Programming:

Operating system: Linux

Editor: Any linux based editor
like vi, edit etc.

Compiler: cc or gcc.

Students will be assessed for each exercise on a scale from 0 to 5.

Not done	0
Incomplete	1
Late Complete	2
Needs improvement	3
Complete	4
Well Done	5

Instruction to the Instructors

- Explain the assignment and related concepts in around ten minutes using whiteboard if required or by demonstrating the software.
- You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- The value should also be entered on assignment completion page of the respective Lab course.

Lab Course I

Section I

Problem Solving using 'C' Programming

Assignment Completion Sheet

Lab Course I			
Sr. No	Assignment Name	Marks	Sign
<i>i</i>	Programming Environment Using basic Linux commands	-	
1	Problem Solving Aspects To demonstrate process of debugging		
2	'C' Fundamentals To demonstrate use of data types, simple operators (expressions)		
3	Control Structures : Conditional Structures To demonstrate decision making statements <ul style="list-style-type: none"> • Use of if ,if-else • Use of Switch case • Use of conditional operator 		
4	Control Structures : Loop Control Structures To demonstrate use of simple & nested loops <ul style="list-style-type: none"> • Use of While loop • Use of Do While loop • Use of for loop 		
5	Control Structures : Break continue and Nested Loop To demonstrate use of simple & nested loops <ul style="list-style-type: none"> • Use of break and continue. • Nested structures and goto statement. 		
6	Functions To demonstrate User defined functions:- declaration , definition, function call, parameter passing (by value), return statement.		
7	Recursive Functions To demonstrate use of recursive functions		
8	Scope of variables To demonstrate writing C programs use of <ul style="list-style-type: none"> • Use of Scope of variables • Use of Storage classes. 		

9	One Dimensional Arrays : Passing array to function To demonstrate <ul style="list-style-type: none"> One Dimensional Arrays (1D) Operations - declaration, initialization, accessing array elements. Assignment on Passing 1D arrays to function 		
10	One Dimensional Arrays : Array Operations , Sorting and Searching To demonstrate , <ul style="list-style-type: none"> Finding maximum and minimum, Counting occurrences, Linear search, Sorting an array Simple exchange sort, bubble sort ie arrange the data in ascending and descending order. 		
11	Two Dimensional Arrays : Basic Operations , Passing 2D arrays to functions To demonstrate <ul style="list-style-type: none"> use of multidimensional array(2-d arrays) and functions Passing 2D arrays to function. Merging two sorted arrays 		
12	Two Dimensional Arrays : matrix operations To demonstrate Matrix operations : <ul style="list-style-type: none"> Transpose Addition, Subtraction Multiplication Symmetric, Diagonal/upper/ lower triangular matrix 		
	Total		

Name and Signature of Batch In-charge

Head of Department

Date

Internal Examiner

External Examiner

Practice Exercise Programming Environment

Objective:

Using basic Linux / UNIX commands.

Reading:

You should read following topics before starting this exercise

UNIX and LINUX operating system

cat with options, ls with options, mkdir, cd, rmdir, cp, mv, cal, pwd, wc, grep with options, etc.

Ready Reference:

About UNIX and LINUX

The success story of UNIX starts with the failure of the MULTICS project. The project failed and the powerful GE-645 machine was withdrawn by GE. Two scientists at Bell Labs, Ken Thompson and Dennis Ritchie, who were part of the MULTICS team, continued to work and succeeded and named their Operating system UNIX, a pun on MULTICS.

The machine available at Bell Labs was a DEC PDP-7 with only 64 k memory while the Operating system they were developing was meant for a larger machine. The problematic situation was handled with an innovative solution. They developed most part of the software in a higher level language, C, which helped them in porting their Operating system from one hardware to another.

With the growing popularity of UNIX, it was available on a variety of machines, from personal computers to mainframes. The most popular amongst them was UNIX System V from AT&T. Each big player in the market came up with their own versions of UNIX. IBM had its own version of UNIX called AIX, which was used on high-end servers. Sun's version of UNIX called Solaris was used on Sun workstations. Novell marketed UnixWare along with Netware, its Network operating system.

LINUX is a version of UNIX, which though it resembles UNIX in looks and feels but differs from other versions in the way it was developed and distributed. In contrast to large proprietary UNIX versions, Linux was developed by Linus Torvalds, a Finish student. He made the source code available and invited partners via the internet in his development effort. He got professional help from all quarters and Linux evolved rapidly. It was made freely available for everyone to use. Linux that was initially meant for Personal computers is now available for a variety of hardware platforms from mainframes to handheld computers.

Linux supports multiple users. Every user need to have an account in order to use the system. One of the users called system administrator (root) is given the charge of creating user accounts and managing the system normally works on the “#” prompt.

You will be given a username and password, using which you can login into Linux operating system. For computer users, the operating system provides a

user-command interface that is easy to use, usually called the Shell. The user can type commands at the shell prompt and get the services of the operating system. Linux operating system shell has the “\$” prompt. You can open a system terminal that gives you a \$ prompt where you can type in various shell commands.

LINUX system will usually offer a variety of shell types:

sh or Bourne Shell: the original shell still used on UNIX systems and in UNIX-related environments. It is available on every Linux system for compatibility with UNIX programs.

bash or Bourne Again shell: the standard GNU shell, is the standard shell for common users on Linux and is a *superset* of the Bourne shell. csh or C shell: the

syntax of this shell resembles that of the C programming language. tcsh or Turbo C shell: a superset of the common C shell, enhancing user-friendliness and speed.

ksh or the Korn shell: A superset of the Bourne shell.

All LINUX commands are case sensitive single words optionally having arguments. One of the argument is options which starts with “-“ sign immediately followed by one or more characters indicating option.

Shell Variables

There are number of predefined shell variables called system or environment variables which are set by the system when the system boots up. Some important system variables are

PATH	It contains set of paths where the system searches for an executablefile
HOME	It is the home or login directory where the user is placed initially
PS1	It is the primary shell prompt which is usually \$
PS2	It is the secondary shell prompt which is usually >

Linux Files and directories

Linux defines three main types of files. Linux treats all devices also as files.

Ordinary or regular file	A file containing data or program
Directory file	A file containing the list of filenames and their unique identifiers.
Special or device file	A file assigned to a device attached to a system

Linux files may or may not have extensions. A file can have any number of dots in its name.

Linux file names are case sensitive. The root directory represented by / is the topmost directory containing number of subdirectories which in turn contains subdirectories and files.

Shell Commands

The following is the list of shell commands

Command	Used for	Example
date	Displays both date and time The command can be used by the system administrator to change date and time.	\$date Format specifiers can be used as arguments +%m month in integer format +%h Name of the month +%d Day of the month +%y Last two digits of the year +%H hours +%M Minutes +%S Seconds \$date +%H \$date +"%h %m"
cal	Displays the calendar	\$cal 8 2007 Displays the calendar for the month august of year 2007 \$cal aug Displays the calendar for the month august of current year
cat	Displays the contents of the files used with the command	\$cat Displays immediately what is typed when you hit enter key \$ cat > abc.txt Whatever number of lines typed till you press ^D are placed in abc.txt file \$cat abc.txt Displays contents of file abc.txt
ls	Displays the contents of current directory. A single dot (.) stands for the current directory while a double dot (..) indicates the parent directory	\$ls lists all files in the current directory \$ls -a Lists also the hidden files \$ls -l Lists the permission information along with other information such as date of last modification, size in blocks etc. the first column of the output exhibits the file type and permissions. File type: -, d, b respectively for

		<p>ordinary, directory and block device file.</p> <p>Permissions are of the form r, w, x, - i.e. read, write, execute and none respectively.</p> <p>There are three groups of rwx. Owner, group and public.</p>
mkdir	Creates specified directory in the current directory, fails if a file or directory by that name is already present or user is not having permissions to create a directory	<p>\$mkdir bin</p> <p>Creates bin directory</p> <p>\$mkdir dir1 dir2 dir3</p> <p>Creates three directories dir1, dir2 and dir3</p>
cd	Switches to specified directory, fails if user is not having permissions to access the directory	<p>\$cd /</p> <p>Switches to root directory</p> <p>\$cd</p> <p>Changes to HOME directory</p>
rmdir	Removes specified directory fails if the directory is not empty	<p>\$rmdir dir1</p> <p>Removes dir1 directory</p> <p>\$rmdir dir2 dir3</p> <p>Removes dir2 and dir3 directories</p>
cp	Creates an exact copy of a file with a different name	<p>\$cp abc.txt xyz.txt</p> <p>Copies abc.txt into a new file named xyz.txt</p> <p>\$cp abc.txt bin</p> <p>Copies abc.txt into a new file with the same name in bin directory</p>
mv	It renames a file or moves a group of files to a different directory	\$mv xyz.txt pqr
rm	Deletes specified file. It can be used with wildcards * and ? as in DOS, to delete all files of a specified type	\$rm pqr
pwd	Displays the path of your present working directory	<p>\$pwd</p> <p>displays the directory in which you are currently working</p>
wc	Counts words, lines and characters or bytes	<p>\$wc -c abc.txt</p> <p>Displays the number of bytes in the file abc.txt</p> <p>\$wc -l abc.txt</p> <p>Displays the number of lines in the file abc.txt</p>

		<p><code>\$wc -w abc.txt</code> Displays the number of words in the file abc.txt</p> <p><code>\$wc abc.txt</code> Displays the number of bytes, words and lines in the file abc.txt</p>
grep	<p>The syntax is <code>grep options pattern filename</code> It displays the lines in the file in which the pattern is found</p>	<p><code>\$grep Agarwal names.txt</code> Displays lines in the names.txt where the string “Agarwal” is present</p> <p><code>\$grep -n Agarwal names.txt</code> Displays lines along with line numbers in the names.txt where the string “Agarwal” is present</p>
man	Offers help on the shell command	<p><code>\$man ls</code> Shows entire manual page of Linux manual pertaining to ls command</p>
passwd	It is used to change the password	<p><code>\$passwd</code> When invoked by an ordinary user asks for the old password and then demands typing and retyping of new password</p> <p><code>#passwd user1</code> Used by administrator to change the password of user1</p>
echo	Displays	<p><code>\$echo \$HOME</code> <code>\$echo \$PATH</code></p>
who	<p>Displays list of users currently logged in</p> <p>The command with arguments “am” and “i” displays login details of the user giving the command</p>	<p><code>\$who</code> <code>\$who am i</code></p>
tail	Displays last lines of the file	<p><code>\$tail -3 abc.txt</code> Displays last three lines of file abc.txt</p>
head	Displays top lines of the file	<p><code>\$head -5 abc.txt</code> Displays top five lines of file abc.txt</p>

Self-Activity -

Assignment

Set A

1. Using cat command, create a file named 'filenames.txt' containing at least ten names and addresses of your friends (firstname , surname, street name, cityname). Type the following commands and explain what the command is used for and give the output of the command

Command	Purpose of Command	Output
wc -lw filenames.txt		
mkdir ass1 ass2		
cp filenames.txt ass2		
cp filenames.txt list		
tail -3 list		
rmdir ass2		
cd ass2		
rm filenames.txt		
Cd		
Pwd		
ls -l		
mv list list.txt		
grep __ filenames.txt		

2. Using cat command create a file named college.txt containing at least ten names and location of colleges (collegename, place,pincode). Type the following commands and explain what the command is used for and give the output of the command

Command	Purpose	Output
mkdir p1 p2 p3 p4		
cp college.txt coll		
cp college.txt coll p1		
head -5 coll		
grep -n ____ college.txt		
rmdir p3 p4		
cd p1		
rm coll		
Pwd		
Cd		
mv coll xyz.txt		
rm *.txt		
ls -a		

Signature of the Instructor

Date

Set B

Give the commands to perform the following actions and give the output

1. List the last three lines of the file demo.txt.
2. Create a file named ____ containing of demo.txt append to itself.
3. Display the current month(string) and year.
4. Display the home directory followed by path
5. Write the contents of directory to a file
6. Append at the end of a file no of lines and the name of the file
7. Create a file named Manual cp containing manual for cp command
8. Display the number of lines containing pattern “ ____ “ in first five lines of the file _

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 1: Problem Solving Aspects

Objective:

To assess and decide which is most appropriate for your issue
Debugging is a process of finding bugs or error in a c program

Reading:

You should read following topics before starting this exercise

1. Pseudo-code to programs and naming conventions
2. Compilation process - compilers , interpreters , linking and loading
3. Error - syntax and semantic, testing a program

Ready References:

All of us have to deal with issues, whether they be small or large, in our personal or professional life. Being able to solve problems consistently is incredibly beneficial, and being able to solve problems well can make you stand out in the workplace.

Pseudo-Code

A Pseudocode is defined as a step-by-step description of an algorithm.

Pseudocode is meant for human comprehension rather than machine reading, it does not employ any programming languages in its representation. Instead, it uses straightforward English text.

Example of Pseudo code

```
/* Ex1. Addition of two numbers */
```

```
Begin  
Write "enter two numbers "  
Read num1  
Read num2  
Sum = num1+num2  
Write Sum  
End
```

```
/*Ex2. Calculate area of a circle */
```

```
Begin  
Input radius  
Area = 3.142 * radius * radius  
Output area  
End
```

Compilation Process:

The process of transforming source code into object code is called compilation. With the compiler's assistance, it is completed. After ensuring that there are no syntactical or structural mistakes in the source code, the compiler creates the object code.

Compiler :

The compiler receives the code that the preprocessor has extended. This code is translated into assembly code by the compiler. Alternatively said, the pre-processed code is transformed into assembly code by the C compiler.

Interpreter :

An interpreter is also a software program that translates a source code into a machine Language. On the other hand, an interpreter runs the program and interprets it by translating high-level programming language line by line into machine language.

Linker:

Every C program makes use of library functions. The object code of these library files is stored with the '.lib' (or '.a') extension, and these library functions are pre-compiled. The linker's primary function is to merge our program's object code with the object code of library files.

Loader:

Loader is a unique program that loads executable files from the linker into main memory and gets the code ready for computer execution. Memory is allotted to the program by the loader. It even calms down object-to-object symbolic references. It is in charge of the operating system's program and library loading.

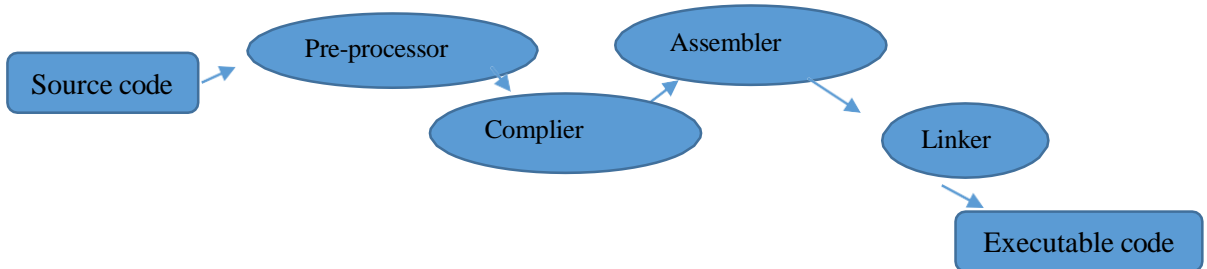


Fig : Compilation Process

Error

Errors are defects or issues that arise in the program and cause abnormal program behaviour. Even seasoned developers are capable of making these mistakes. Debugging is the process of eliminating programming errors, usually referred to as bugs or faults.

Syntax Error

Syntax errors are also known as the compilation errors. They occurred at the compilation time, or we can say that the syntax errors are thrown by the compilers.

syntax errors occurred are :

- If, when writing the code, the parenthesis (}) is overlooked.
- Displaying a variable's value in the absence of its declaration.
- If the statement's final semicolon (;) is overlooked.

Example :

```
/*Ex3. Declare the variable of type integer*/
```

```
    int a; // this is the correct form
```

```
    Int a; // this is an incorrect form.
```

```
/*Ex4. Not mention the datatype at the time of declaration.*/
```

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

```
    int main()
```

```
    {
```

```
        a = 10;
```

```
        printf("The value of a is : %d", a);
```

```
        return 0;
```

```
    }
```

```
    Error : 'a' is undeclared
```

Semantic Error

Semantic errors are those that arise when the compiler is unable to comprehend the assertions.

Example :

```
/* Ex5. Use of a un-initialized variable. */
```

```
    int i;
```

```
    i=i+2;
```

```
/* Ex6. Errors in expressions*/
```

```
    int a, b, c;
```

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

```
/*Ex7. Incorrect Statement */
```

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

```
    int main()
```

```
    {
```

```
        int a,b,c;
```

```
        a=2;
```

```
        b=3;
```

```
        c=1;
```

```
        a+b=c; // semantic error
```

```
return 0;
```

```
}
```

Error :

statement **a+b=c**, which is incorrect as we cannot use the two operands on the left-side.

Set A

1. Write a Pseudo code to multiplication of two numbers.
2. Write a Pseudo code to calculate the Sum of Natural Numbers.
3. Write a Pseudo code to calculate area of triangle.
4. Write a Pseudo code to check number is even or odd,
5. Write a Pseudo code to check number is prime or not.

Signature of the Instructor

Date

Set B

Find errors if any in the following program. Justify your answer.

1.

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

```
void main()
```

```
{
```

```
    int int_x;
```

```
    printf("Enter the value of x");
```

```
    scanf("%d",&int_x);
```

```
}
```

2.

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

```
void main()
```

```
{
```

```
printf("Good Day")
```

```
}
```

3.

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

```
void main()
```

```
{
```

```
int a = 8877;
```

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

```
}
```

4.

```
#include<stdio.h>
void main()
{
int a, b, c;
a + b = c;
}
```

5.

```
#include<stdio.h>

void main() {

    int var;
    var = 20 / 0;

    printf("%d", var);
}
```

6.

```
#include<stdio.h>
int main()
{
while(.)
{
printf("hello");
}
return 0;
}
```

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 2 : 'C' Fundamentals

Objective :

To demonstrate the use of data types, simple operators and expressions

Reading :

You should read following topics before starting this exercise

1. Different basic data types in C and rules of declaring variables in C
2. Different operators and operator symbols in C
3. How to construct expressions in C, operator precedence
4. Problem solving steps- writing algorithms and flowcharts

Ready References :

Data type :

Data type	Size (Bytes)	Range	Format Specifiers
Char	1	- 128 to 127	%c
Unsigned char	1	0 to 255	%c
int	2 (16 bit)	-32768 to 32767	%d
	4 (32 bit)	-2147483648 to 2147483647	%d
Short or int	2	- 32768 to 32767	% i or %d
Unsigned int	2	0 to 65535	%u
Float	4	3.4e - 38 to +3.4e +38	%f or %g
Long	4	- 2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%lu
Double	8	1.7e - 308 to 1.7e+308	%lf
Long double	10	3.4e - 4932 to 1.1e+4932	%lf

Operators Precedence

Category	Operator	Associativity
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to left
Postfix	() [] -> . ++ --	Left to right
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right

Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

Data	Data Format	C Data Type	C Variable declaration	Input Statement	Output statement
quantity month credit-card number	Numeric	int Short int long int	int quantity; short month; long ccno;	scanf("%d",&quantity); scanf("%d",&month); scanf("%ld",&ccno);	printf("The quantity is %d", quantity); printf("The credit card number is %ld, ccno);
price	real	float double	float price; const double pi=3.141593;	scanf("%f",&price);	printf("The price is %5.2f", price);
grade	character		char grade;	scanf("%c",&grade)	printf("The grade is %c",grade);

Expression Examples

Expression	C expression
Increment a by 3	a = a + 3
Decrement b by 1	b = b-1 or b- -
$2a^2 + 5b/2$	2*a*a + 5*b/2
$7/13(x-5)$	(float)7/13*(x-5)
5% of 56	(float)5/100*56
n is between 12 to 70	n>=12 && n<=70
$\pi r^2 h$	Pi*r*r*h
n is not divisible by 7	n % 7 != 0
n is even	n%2== 0
ch is an alphabet	ch>='A' && ch<='Z' ch>='a' && ch<='z'
x+=2	x=x+2
y-=50	y=y-50
m*=5	m=m*5
a/=10	a=a/10

Note: The operators in the above expressions will be executed according to precedence and associativity rules of operators.

Sample program- to calculate and print simple interest after accepting principal sum, number of years and rate of interest.

Step 1 : Writing the Algorithm	Step 2 : Draw the flowchart	Step 3 : Writing Program
<ol style="list-style-type: none"> 1. Start 2. Accept principal sum, rate of interest and number of years 3. Compute Simple interest 4. Output Simple Interest 5. Stop 	<pre> graph TD Start([Start]) --> Read[/Read principal sum, rate and no. of years/] Read --> Compute[Compute Simple interest] Compute --> Print[/Print Simple Interest/] Print --> Stop([Stop]) </pre>	<pre> /* Program to calculate simple interest */ #include <stdio.h> main() { /* variable declarations */ float amount, rateOfInterest, simpleInterest; int noOfYears; /* prompting and accepting input */ printf("Give the Principal Sum"); scanf("%f",&amount); printf("Give the Rate of Interest"); scanf("%f",&rateOfInterest); printf("Give the Number of years"); scanf("%d",&noOfYears); /* Compute the simple Interest*/ simpleInterest=amount*noOfYears*rateOfInterest /100; /* Print the result*/ printf("The simple Interest on amount %7.2f for %d years at the rate %4.2f is %6.2f", amount, noOfYears, rateOfInterest, simpleInterest); } </pre>

Follow the following guidelines (Save and Compiled code)

- At \$ prompt type vi followed by filename. The filename should have .c as extension

Example

\$vi simple_interest.c (Program Name)

- Type the sample program given above using vi commands and save it. Compile the program using cc compiler available in Linux

\$cc simple_interest.c or \$ gcc simple_interest.c

It will give errors if any or it will give back the \$ prompt if there are no errors
 A executable file a.out is created by the compiler in current directory.
 The program can be executed by typing name of the file as follows
 giving the path.

```
$ ./a.out
```

Alternatively the executable file can be given name by using -o option while compiling as follows

```
$cc simple_interest.c -o pnrexec
$./pnrexec
```

The executable file by specified name will be created. Note that you have to specify the path of pnr exec as ./pnrexec , i. e., pnrexec in current (. Stands for current directory) directory otherwise it looks for program by that name in the path specified for executable programs

Self Activity:

Type the sample program given above. Execute it for the different values as given below and fill the last column from the output given by the program.

Sr. No	Principal sum	No of years	Rate of interest	Simple Interest
1	20000	3		
2	42500	_____	4.5	
3	_____	6	8.3	

Set A . Write programs to solve the following problems.

1. Write a program to accept two numbers and interchange them without using third variable.
2. Write a program to Calculate Volume and Total Surface Area of Cuboid. Accept three dimensions length (l), breadth(b) and height(h) of a cuboid. (Hint : Total Surface area of Cuboid = $2 * (l*b + l*h + b*h)$ Volume of Cuboid = $l * b * h$)
3. Write a program to accept dimensions of a cylinder and display the surface area and volume (Hint: surfacearea = $2\pi r^2 + 2\pi rh$, volume = $\pi r^2 h$)
4. Write a program to accept temperatures in Fahrenheit (F) and print it in Celsius(C) and Kelvin (K) (Hint: $C = 5/9(F - 32)$, $K = C + 273.15$)
5. Write a program to accept initial velocity (u), acceleration (a) and time (t). Display the final velocity (v) and the distance (s) travelled. (Hint: $v = u + at$, $s = u + at^2$)

6. Write a program to accept inner and outer radius of a ring and print the perimeter and area of the ring(Hint: perimeter = $2\pi(a+b)$, area = $\pi(a^2-b^2)$)
7. Write a program to accept two numbers and print arithmetic and harmonic mean of the two numbers (Hint: AM= $(a+b)/2$, HM = $ab/(a+b)$)

Signature of the Instructor

Date

Set B . Write programs to solve the following problems.

1. Write a program to find total surface area of a cone accept height and radius from user.
(Hint :surfaceArea = $PI*radius*(radius + \sqrt{height*height + radius*radius})$); volume = $1.0/3 * (PI*radius*radius*height)$);
2. Write a program to accept a character from the keyboard and display its previous and next character in order. (Hint: Ex. If the character entered is 'n', display "The previous character is m", "The next character is o").
3. Write a program to accept a character from the user and display its ASCII value.
4. Write a program to accept the x and y coordinates of two points and compute the distance between the two points.
5. Write a program to accept the amount to be withdrawn from the user and print the total number of currency notes of each denomination the cashier will have to give.
(Hint : a cashier has currency notes of denomination 1, 5 and 10)
6. The basic salary of an employee is decided at the time of employment, which may be different for different employees. Apart from basic, employee gets 10% of basic as house rent, 30% of basic as dearness allowance. A professional tax of 5% of basic is deducted from salary. Accept the employee id and basic salary for an employee and output the take home salary of the employee.

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 3 : Control Structures : Conditional Structures

Objective :

To demonstrate use of decision making statements such as if and if-else and switch-case.

Reading :

You should read following topics before starting this exercise

1. Different types of decision-making statements available in C.
2. Syntax for these statements.
3. Syntax for switch case statements.

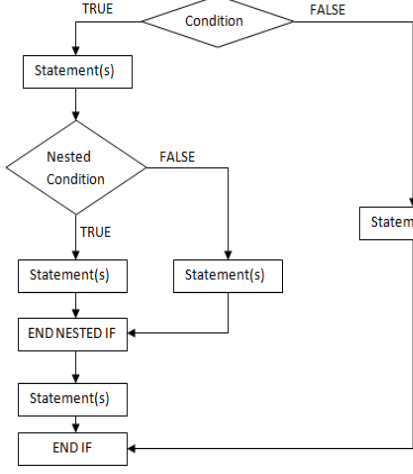
Reading References :

During problem solving, we come across situations when we have to choose one of the alternative paths depending upon the result of some condition. Condition is an expression evaluating to true or false. This is known as the Branching or decision-making statement. Several forms of If and else constructs are used in C to support decision-making.

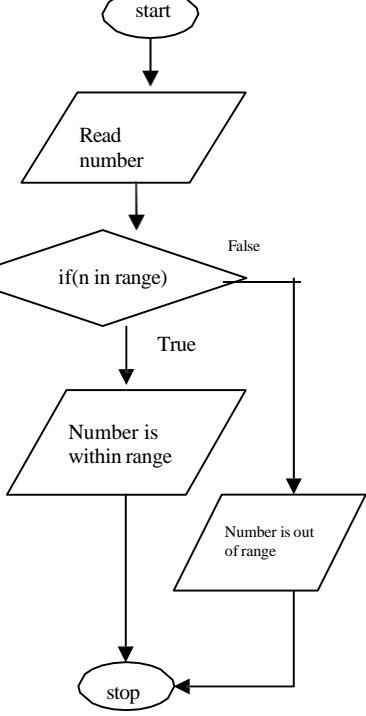
- 1) if statements
- 2) if – else
- 3) Nested if
- 4) switch case

Note: If there are more than one statement in the if or else part, it has to be enclosed in { } braces

Sr. No	Statement Syntax	Flowchart	Example
1.	if statement if (condition) { statement; }	<pre> graph TD Start(()) --> Condition{Condition} Condition -- True --> Statement[Statement(s)] Condition -- False --> EndIF[End IF] Statement --> EndIF </pre> <p>fig: Flowchart for if statement</p>	<pre> if(n > 0) printf("Numberis positive"); </pre>
2.	if - else statement if (condition) { statement; } else { statement; }	<pre> graph TD Start(()) --> Condition{Condition} Condition -- "True (if part)" --> Statement1[Statement(s)] Condition -- "False (else part)" --> Statement2[Statement(s)] Statement1 --> EndIF[End IF] Statement2 --> EndIF </pre> <p>fig: Flowchart for if ... else statement</p>	<pre> if(n % 2 == 0) printf("Even"); else printf("Odd"); </pre>

3.	Nested if <pre> if (condition) { if (condition) { statement; }else { statement;} } else { if (condition) { statement; }else { statement; } } </pre>	 <p>fig: Flowchart for nested if statement</p>	<pre> if (a >= b) { if (a >= c) printf(“ %d is maximum”,a);else printf(“ %d is maximum”,c); } else { if (b >= c) printf(“ %d is maximum”,b); else printf(“ %d is maximum”,c); } </pre>
----	---	--	--

4. Sample program to check whether a number is within range.

Step 1: Writing the Algorithm	Step 2 : Draw the flowchart	Step 3 : Writing Program
<ol style="list-style-type: none"> Start Accept the number Check if number is within range if true print “Number is within range” “ otherwise print “number is out of range”. Stop 		<pre> /* Program to check range */ #include <stdio.h> main() { /* variable declarations */ int n; int llimit=50, ulimit = 100; /* prompting and accepting input */ printf(“Enter the number”); scanf(“%d”,&n); if(n>=llimit && n <= ulimit) printf(“Number is within range”); else printf(“Number is out of range”); } </pre>

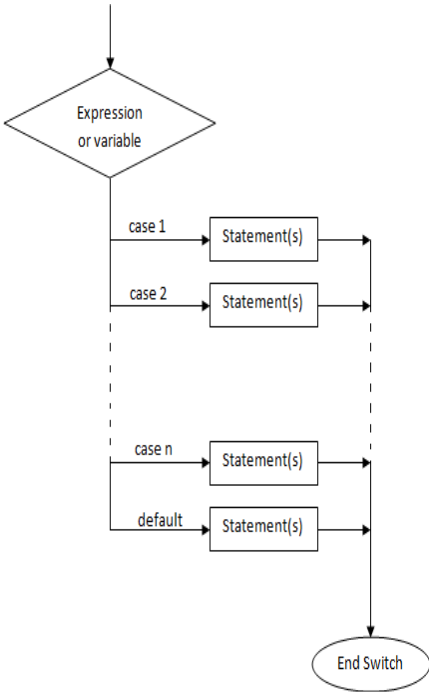
The control statement that allows us to make a decision from the number of choices is called as switch-case statement. It is a multi-way decision making statement.

1. Usage of switch statement

Statement Syntax	Flowchart	Example
<pre> switch(expression) { case value1: block1; break; case value2: block2; break; . . . default : default block; break; } </pre>	<pre> graph TD Start([star]) --> D1{case 1} D1 -- T --> B1[Block1] D1 -- F --> D2{case 2} D2 -- T --> B2[Block2] D2 -- F --> D3{case 3} D3 -- T --> B3[Block3] D3 -- F --> D4{case 2} D4 -- T --> B4[Block4] D4 -- F --> Default[Default] B1 --> Join(()) B2 --> Join B3 --> Join B4 --> Join Default --> Join Join --> Stop([Stop]) </pre>	<pre> switch (color) { case 'r' : case 'R' : printf ("RED"); break; case 'g' : case 'G' : printf ("GREEN"); break; case 'b' : case 'B' : printf ("BLUE"); break; default : printf ("INVALID COLOR"); } </pre>

The switch statement is used in writing menu driven programs where a menu displays several options and the user gives the choice by typing a character or number.

A Sample program to display the selected option from a menu is given below.

Step 1: Writing the Algorithm	Step 2: Draw the flowchart	Step 3: Writing Program
1. Start 2. Display the menu options 3. Read choice 4. Execute statement block depending on choice 5. Stop	 <p>fig: Flowchart for switch case statement</p>	<pre> /* Program using switch case and menu */ #include <stdio.h> main() { /* variable declarations */ int choice; /* Displaying the Menu */ printf("\n 1. Option 1\n"); printf(" 2. Option 2\n"); printf(" 3. Option 3\n"); printf("Enter your choice"); scanf("%d",&choice); switch(choice) { case 1: printf("Option 1 is selected"); break; case 2: printf("Option 2 is selected"); break; case 3: printf("Option 3 is selected"); break; default: printf("Invalid choice"); } } </pre>

Conditional Operator

The conditional operator is also known as a **ternary operator**. The conditional statements are the decision-making statements which depends upon the output of the expression.

It is represented by two symbols, i.e., '?' and ':'.

Syntax :

expression1? expression2: expression3;

/*Ex1. Demo of conditional operator. */

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

```
int main()
```

```
{
```

```
    int a=5,b; // variable declaration
```

```
    b = ((a==5)?(2024):(2025)); // conditional operator
```

```
    printf("Value is : %d",b);
```

```
    return 0;
```

```
}
```

//OUTPUT : 2024

/*Ex2. Program to find the largest number among three numbers*/

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

```
int main()
```

```
{
```

```
    int n1, n2, n3;          //variable declaraton
```

```
    printf("Enter three numbers: ");
```

```
    scanf("%d %d %d", &n1, &n2, &n3);
```

```
    if (n1 >= n2) // outer if statement
```

```
    {
```

```
        if (n1 >= n3) // inner if...else
```

```
            printf("%d is the largest number.", n1);
```

```
        else
```

```
            printf("%d is the largest number.", n3);
```

```
    }
```

```
    else // outer else statement
```

```
    {
```

```
        if (n2 >= n3) // inner if...else
```

```
            printf("%d is the largest number.", n2);
```

```
        else
```

```
            printf("%d is the largest number.", n3);
```

```
    }
```

```
    return 0;
```

```
}
```

/*Ex3. Write a Program of Switch Case display Weekday*/

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

```
main()
```

```
{
```

```
    char ch;
```

```
    printf("Enter the day character = ");
```

```
    scanf("%c",&ch);
```

```
switch(ch)
{
    case 'm': printf("\nMonday");
               break;
    case 't': printf("\nTuesday");
               break;
    case 'w': printf("\nWednesday");
               break;
    case 'T': printf("\nThursday");
               break;
    case 'f': printf("\nFriday");
               break;
    case 's': printf("\nSaturday");
               break;
    case 'S': printf("\nSunday");
               break;
}
}
```

Set A: Write programs to solve the following problems.

1. Write a program to accept an integer and check if it is even or odd.
2. Write a program to accept three numbers and check whether the first is between the other two numbers.
(Hint: Ex: Input 200 100 300. Output: 200 is between 100 and 300).
3. Write a program to check user age eligible for voting or not (using conditional operator).
4. Write a program to accept a character as input and check whether the character is a digit. (Check if it is in the range 0' to '9' both inclusive)
5. Write a program to accept a number and check if it is divisible by 5 and 7.
6. Write a program, which accepts annual basic salary of an employee and calculates and displays the Income tax as per the following rules.
Basic: < 1,50,000 Tax = 0
1,50,000 to 3,00,000 Tax = 20%
 > 3,00,000 Tax = 30%
7. Write a program to accept a lowercase character from the user and check whether the character is a vowel or consonant.
(Hint: a,e,i,o,u are vowels)

8. Accept a single digit from the user and display it in words.
For example, if digit entered is 9, display Nine.
9. Write a program to create Simple Calculator using switch case.

Signature of the Instructor

Date

Set B: Write programs to solve the following problems.

1. Write a program, which accepts two integers and an operator as a character (+ - * /), performs the corresponding operation and displays the result.
2. Accept two numbers in variables x and y from the user and perform the following operations

Options	Actions
1. Equality	Check if x is equal to y
2. Less Than	Check if x is less than y
3. Quotient and Remainder	Divide x by y and display the quotient and remainder
4. Range	Accept a number and check if it lies between x and y(both inclusive)
5. Swap	Interchange x and y

3. Write a program to accept the time as hour, minute and seconds and check whether the time is valid.
(Hint: $0 \leq \text{hour} < 24$, $0 \leq \text{minute} < 60$, $0 \leq \text{second} < 60$)
4. Write a program to any year as input through the keyboard. Write a program to check whether the year is a leap year or not. (Hint: leap year is divisible by 4 and not by 100 or divisible by 400)
5. Write a program to three sides of triangle as input, and print whether the triangle is valid or not. (Hint: The triangle is valid if the sum of each of the two sides is greater than the third side).
6. Write a program to the x and y coordinate of a point and find the quadrant in which the point lies.
7. Write a program to the cost price and selling price from the keyboard. Find out if the seller has made a profit or loss and display how much profit or loss has been made.
8. Write a program to radius from the user and write a program having menu with the following options and corresponding actions

Options	Actions
1. Area of Circle	Compute area of circle and print
2. Circumference of Circle	Compute Circumference of circle and print
3. Volume of Sphere	Compute Volume of Sphere and print

9. Write a program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to following:

Percentage $\geq 90\%$: Grade A

Percentage $\geq 80\%$: Grade B

Percentage $\geq 70\%$: Grade C

Percentage $\geq 60\%$: Grade D

Percentage $\geq 40\%$: Grade E

Percentage $< 40\%$: Grade F

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2: Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 4 : Control Structures : Loop Control Structures

Objective :

To demonstrate use of simple loops.& nested loops

Reading :

You should read following topics before starting this exercise

1. Different types of loop structures in C.
2. Syntax and usage of these statements.
3. Usage of each loop structure

Ready References :

We need to perform certain actions repeatedly for a fixed number of times or till some condition holds true. These repetitive operations are done using loop control statements. The types of loop structures supported in C are

1. while statement
2. do-while statement
3. for statement

Sr. No.	Statement Syntax	Flow Chart	Example
1	while statement while (condition) { statement 1; statement 2; . . }	 fig: Flowchart for while loop	<pre> /* accept a number*/ scanf("%d", &n); /* if not a single digit */while (n > 9) { /* remove last digitn = n /10; } </pre>
2	do-while statement do { statement 1; statement 2; . . } while (condition);	 fig: Flowchart for do-while loop	<pre> /*initialize sum*/ sum =0; do { /* Get a number */ printf(" give number"); scanf("%d",&n); /* add number to sum*/ sum=sum+n; } while (n>0); </pre>

3	for statement for(expr1; expr2; expr3) { statement 1 : : } expr1 = initialization expression expr2 = loop condition expr3 = alteration expression which alters the loop variable	<pre> graph TD A[expr1] --> B{test expr2} B -- F --> C[] B -- T --> D[Loop body] D --> E[Expr3] E --> B </pre>	<pre> /* display first 10 multiples of 2 */ for(i=1; i <= 10; i++) { printf ("2 X %d = %d\n", i, 2*i); } </pre>
---	---	--	---

Note: Usually the for loop is used when the statements have to be executed for a fixed number of times. The while loop is used when the statements have to be executed as long as some condition is true and the do-while loop is used when we want to execute statements at least once (example: menu driven programs)

4. Sample program- to print sum of 1+2+3+.....n.

Step 1: Writing the Algorithm	Step 2: Draw the flowchart	Step 3: Writing Program
1. Start 2. Initialize sum to 0. 3. Accept n. 4. Compute sum=sum+n 5. Decrement n by 1 6. if n > 0 go to step 4 7. Display value of sum. 8. Stop	<pre> graph TD Start([start]) --> Sum0[Sum = 0] Sum0 --> ReadN[/Read n/] ReadN --> ComputeSum[Compute Sum=sum+n] ComputeSum --> Ngt0{n > 0} Ngt0 -- True --> ComputeSum Ngt0 -- False --> PrintSum[/Print value of sum/] PrintSum --> Stop([stop]) </pre>	<pre> /* Program to calculate sum of numbers */ #include <stdio.h> main() { /* variable declarations */ int sum = 0, n; printf("enter the value of n : "); scanf("%d",&n); while (n>0) { sum = sum + n; n--; } printf("\n The sum of numbers is %d", sum); } </pre>

5. Sample program- To read characters till EOF (Ctrl+Z) and count the total number of characters entered.

Step 1 : Writing the Algorithm	Step 2 : Draw the flowchart	Step 3 : Writing Program
1. Start 2. Initialize count to 0. 3. Accept ch. 4. If ch !=EOF Count = count +1 Else Go to step 6 5. Go to step 3 7. Display value of sum. 8. Stop	<pre> graph TD Start([start]) --> Init[/Count = 0/] Init --> Read[/Read ch/] Read --> Decision{Ch == EOF} Decision -- T --> Print[/Print count/] Print --> Stop([stop]) Decision -- F --> Increment[Count = count + 1] Increment --> Read </pre>	<pre> /* Program to count number of characters */ #include <stdio.h> main() { char ch; int count=0; while((ch=getchar())!=EOF) count++; printf("Total characters = %d", count); } </pre>

Nested loop means a loop that is contained within another loop. Nesting can be done upto any levels. However the inner loop has to be completely enclosed in the outer loop. No overlapping of loops is allowed.

Sr. No	Format	Sample Program
1.	Nested for loop <pre> for(exp1; exp2 ; exp3) { for(exp11; exp12 ; exp13) { } } </pre>	<pre> /* Program to display triangle of numbers*/ #include <stdio.h> void main() { int n , line_number , number; printf("How many lines: "); scanf("%d",&n); for(line_number =1 ;line_number <=n; line_number++) { for(number = 1; number <= line_number; number++) printf ("%d\t", number); printf ("\n"); } } </pre>

2.	Nested while loop / do while loop <pre> while(condition1) { while(condition2) { } } do { while(condition1) { } } while (condition2); </pre>	<pre> /* Program to calculate sum of digits till sum is a single digit number */ #include <stdio.h> void main() { int n , sum; printf("Give any number "); scanf("%d",&n); do { sum =0; printf("%d --->",n); while (n>0) { sum +=n%10; n= n/10; } n=sum; } while(n >9); printf ("%d" , n); } </pre>
----	--	--

Note: It is possible to nest any loop within another. For example, we can have a for

/*Ex1. Write a program to display the series using while loop.

Example : 1 2 3 4 5.... */

```

#include<stdio.h>
void main()
{
int limit,i;
printf("How many number you want into series?");
scanf("%d",&limit);
printf("\n Series are = ");
i = 1;          //initialization of variable 'i'
while(i<=limit)
{
printf("%d ",i);
i++; //increment the variable 'i'
} //End of while()

} //End of main()

```

/*OUTPUT

```

[root@localhost FOR]# cc w1_series.c
[root@localhost FOR]# ./a.out
How many number you want into series?
10
Series are = 1 2 3 4 5 6 7 8 9 10 */

```

loop inside a while or do while or a while loop inside a for.

/*Ex2. Write a program to display reverse of given number using while loop.

Example : 986 - 689

```
*/  
  
#include<stdio.h>  
void main()  
{  
    int n,rev,temp;  
    printf("\nEnter the number = ");  
    scanf("%d",&n);  
    rev = 0;  
    while(n>0)                //check the number greater than '0'  
    {  
        temp = n % 10;        // getting remainder value  
        rev = (rev*10) + temp; // reverse of number  
        n = n / 10;           //new value of 'n'  
    }//End of while()  
  
    printf("\nReverse is = %d",rev);  
}//End of main()
```

/*OUTPUT

[root@localhost FOR]# cc reverse.c

[root@localhost FOR]# ./a.out

Enter the number = 986

Reverse is = 689

***/**

// Ex3. Write a program to using do_while loop

```
#include<stdio.h>    //header file  
void main()  
{  
    int n,i;  
    i=1;  
    printf("\nEnter the number = ");  
    scanf("%d",&n);  
    do                //start of do  
    {  
        printf("\nGood Moring...");  
    }while(i++<=n);    //End of while()  
}//End main()
```

/*OUTPUT

```
[root@localhost FOR]# cc w3_dowhile.c
[root@localhost FOR]# ./a.out
Enter the number = 4
Good Moring...
Good Moring...
Good Moring...
Good Moring...
Good Moring...
*/
```

//Ex4. Write a program to display table of any number.

```
#include<stdio.h>
void main()
{
    int n,i,j;
    printf("\nEnter the number = ");
    scanf("%d",&n);
    printf("\nTable is\n");
    for(i=1;i<=10;i++)
    {
        printf("\n%d * %d = %d",n,i,n*i);
    }//End of for() loop
}//End fo main()
```

/* OUTPUT

```
[root@localhost FOR]# cc table.c
[root@localhost FOR]# ./a.out
```

Enter the number = 7

Table is

```
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
*/
```

/*Ex5. Write a program to display pattern as below*/

```
$  
$ $  
$ $ $
```

```
#include<stdio.h>  
void main()  
{  
    int n,i,j;  
    printf("\nEnter the number = ");  
    scanf("%d",&n);  
    printf("\nPattern is \n");  
    for(i=1;i<=n;i++)  
    {  
        for(j=1;j<=i;j++)  
        {  
            printf("$ ");  
        }//End of j loop  
        printf("\n");  
    }//End of i loop  
}//End fo main()
```

/* OUTPUT

```
[root@localhost FOR]# cc patt.c  
[root@localhost FOR]# ./a.out
```

Enter the number = 5

Pattern is

```
$  
$ $  
$ $ $  
$ $ $ $  
$ $ $ $ $
```

***/**

Set A. Write programs to solve the following problems

1. Write a program to accept an integer n and display all even numbers upto n.
2. Write a program to accept two integers x and y and calculate the sum of all integers between x and y (Hint. x = 2 and y = 7 sum = 3+4+5+6=18)
3. Write a program to accept base and power value and find the power of base value (Hint : compute (base)^{power} eg. 2³ = 8)
4. Write a program to accept an integer and check if it is prime or not.

5. Write a program to accept an integer and count the number of digits in the number.
6. Write a program to accept an integer and reverse the number.
(Hint: Input: 456, Output: 654).
7. Write a program to accept a character, an integer n and display the next n characters.
8. Write a program to display all prime numbers between 1 and 500.

Signature of the Instructor

Date

Set B . Write programs to solve the following problems

1. Write a program to accept the limit and display Fibonacci series up to n terms. (Hint : 0 1 1 2 3 5)
2. Write a program to display multiplication tables from 2 to 9 having n multiples each. The output should be displayed in a tabular format. example, the multiplication tables of 2 to 9 having 10 multiples each is shown below.

2 * 1 = 2	3 * 1 = 3	9 * 1 = 9
2 * 2 = 4	3 * 2 = 6	9 * 2 = 18

2 * 10 = 20 3 * 10 = 30..... 9 * 10 = 90

3. Write a program to accept a 'n' from user and display pattern as below.
(here n=4).

```

A B C D
E F G
H I
J

```

4. Write a program to accept real number x and integer n and calculate the sum of first n terms of the series $x + 3x + 5x + 7x + \dots$
5. Write a program to accept characters till the user enters EOF and count number of alphabets and digits entered.
6. Write a program, which accepts a number n and displays each digit in words.
Example: 6702 Output = Six-Seven-Zero-Two.
(Hint: Reverse the number and use a switch statement)
7. Write a program to display all Armstrong numbers between 1 and 500.
(Hint : An Armstrong number is a number such that the sum of cube of digits = number itself Ex. $371 = 3^3 + 7^3 + 1^3$)
8. Accept characters till the user enters EOF and count the number of lines entered. Also display the length of the longest line.
(Hint: A line ends when the character is '\n')

9. Display all perfect numbers below 500. (Hint: A perfect number is a number, such that the sum of its factors is equal to the number itself . Ex. 6 (1 + 2 + 3), 28 (1+2+4+7+14))

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 5 : Control Structures : Break , Continue and Nested Loop

Objective :

Use of break and continue.

Reading :

Control structures – break , continue and goto loop.

Ready References :

The jump statements "continue" and "break" both transfer control to a different area of the program.

The continue statement allowed control to move on to the next iteration of the loop, while the break statement allowed control to exit the loop.

Break Statement

Use the break statement to end a loop when a particular condition is satisfied.

The break statement should terminate with a semicolon (;).

In a loop construct, the control ends instantly when it hits the break statement.

Syntax

It can be written as:

break;

A break statement interrupts the execution of a case by causing a switch or a loop to cease. It indicates that a switch or loop will terminate suddenly if there is a break in it.

//Ex1. Demo example of break statement

```
#include<stdio.h>

int main()
{
    int num;
    printf("Enter your number from 0 to 99: ");
    scanf("%d", &num);
    int cnt = 0;
    while(cnt < 100)
    {
        printf("\nThe value of count is %d.", cnt);
        if (count == num)
            break;
        cnt++;
    }
}
```

```
}  
    return 0;  
}
```

Continue Statement

The continue statement doesn't break the loop, in contrast to the break statement.

Instead, it only moves past iterations where the condition is met.

It does not allow a user to exit an overall loop structure.

As soon as the loop statement starts, control moves from the beginning to the continue statement.

Syntax

It can be written as:

continue;

The loop is led to the subsequent iteration rather than being terminated by the continue statement. It indicates that a loop will finish all of its iterations if it comes across a continue statement.

The sentences that come after the continue in a loop are skipped when you employ the continue statement.

//Ex2. Demo example of continue statement

```
#include <stdio.h>  
int main()  
{  
    int i;  
    double number, sum = 0.0;  
  
    for (i = 1; i <= 10; i++)  
    {  
        printf("Enter a n%d: ", i);  
        scanf("%lf", &number);  
  
        if (number < 0.0)  
        {  
            continue;  
        }  
  
        sum = sum + number;  
    }  
  
    printf("Sum = %.2lf", sum);  
  
    return 0;  
}
```

Nested structures and goto statement.

A jump statement, often known as an unconditional jump statement, is a goto statement. Within a function, you can hop from one place to another using the goto command.

Syntax

```
goto label;  
... ..  
... ..  
label:  
statement;
```

The only situation in which breaking out of several loops with a single statement at the same time is advantageous is when we use goto.

//Ex3. Demo example of goto statement

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

```
int main()  
{
```

```
    const int maxInput = 100;  
    int i;  
    double number, average, sum = 0.0;
```

```
    for (i = 1; i <= maxInput; ++i)  
    {  
        printf("%d. Enter a number: ", i);  
        scanf("%lf", &number);
```

```
        // go to jump if the user enters a negative number  
        if (number < 0.0)
```

```
        {  
            goto jump;
```

```
        }  
        sum += number;  
    }
```

```
jump:
```

```
    average = sum / (i - 1);  
    printf("Sum = %.2f\n", sum);  
    printf("Average = %.2f", average);
```

```
    return 0;  
}
```

Set A . Write programs to solve the following problems

1. Write a program to printing numbers using the goto statement
(Hint : 10 20 30 ...)
2. Write a program to find ceil division of two numbers.
3. Write a program to calculate and outputs the absolute value of any given integer.
4. Write a program to accept the number whose table you want to print.
5. Write a program to read the age of 100 persons and count the number of persons in the age group 40 to 50. (Hint : use continue statement.)

Signature of the Instructor

Date

Set B . Write programs to solve the following problems

1. Write a program to accept the number from user and check the number is prime number or not.
2. Write a program to display all Armstrong numbers in between 1 to 1000.
3. Write a program to evaluates the square root for 5 numbers. The variable 'count' keeps the count of number read. When count is less than or equal to 5 , goto **read**; directs the control to the label **read**; otherwise, the program prints a message and stops.

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 6 : Functions

Objective :

A function is a collection of statements that work together to accomplish a goal.

Reading :

The main() function is present in all C programs, and even the most basic ones have the ability to specify more functions.

Ready References :

A function is a collection of statements that receives inputs, performs a particular calculation,

and outputs the result.

The goal is to combine some often performed tasks into a function so that we can call it rather

than having to write the same code over and over again for various inputs.

The benefits of using C functions are as follows. Using functions allows us to avoid repeatedly repeating the same logic or code within a program. It is possible to call C functions

The function body contains the declarations and the statements necessary for performing the required task.

The body is enclosed within curly braces { ... } and consists of three parts.

- local variable declaration (if required).
- function statements to perform the task inside the function.
- a return statement to return the result evaluated by the function (if return type is void, then no return statement is required).

User defined functions:- declaration

A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.

A function declaration tells the compiler about a function name and how to call the function.

The actual body of the function can be defined separately.

A function declaration has the following parts –

return_type function_name(parameter list);

the function declaration is as follows –

int max(int num1, int num2);

Parameter names are not important in function declaration only their type is required, so the

following is also a valid declaration –

int max(int, int);

You can also create functions as per your need.
Such functions created by the user are known as user-defined functions.
How user-defined function works?

Ex.

```
include <stdio.h>
void functionName()
{
    ... ..
    ... ..
}
int main()
{
    ... ..
    ... ..
```

```
functionName();
```

```
... ..
... ..
}
```

The execution of a C program begins from the main() function.
When the compiler encounters functionName();, control of the program jumps to void functionName()
And, the compiler starts executing the codes inside functionName().

Definition

The actual assertions that need to be performed are contained in it.
When the function is called, control is applied to the most crucial feature.
It is important to note that the function can only return a single value at this point.

the general syntax of function definition is,

```
returntype functionName(type1 parameter1, type2 parameter2,...)
{
    // function body goes here
}
```

The first line returntype functionName(type1 parameter1, type2 parameter2,...) is known as function header and the statement(s) within curly braces is called function body.

Note

Unlike when a function is declared or called, when a function is defined, there is no semicolon (;) following the parenthesis in the function header.

Function call

Program control is passed to the called function when a program calls a function. When a called function completes a specified task and executes its return statement or reaches its function-ending closing brace, the program control is returned to the main program.

All you have to do to invoke a function is give the function name and the necessary parameters.

If the function returns a value, you can store the result.

parameter passing (by value), return statement.

Call by value OR Pass by Value:

The formal parameters of the function are copied with the values of the actual parameters in this parameter passing mechanism, and the two kinds of parameters are kept in separate memory regions. As a result, modifications done inside functions do not affect the caller's actual parameters.

Parameters are always passed by value.

Example. in the below code, value of x is not modified using the function fun().

//Ex1. Demo example of call by value.

```
#include <stdio.h>
void fun(int x)
{
    x = 300;
}
int main(void)
{
    int x = 200;

    fun(x);
    printf("x = %d", x);
    return 0;
}
```

Output:

x = 200

Call by reference OR Pass by Reference

Since the actual and formal parameters correspond to the identical locations, any modifications done inside the function are mirrored in the caller's actual parameters.

//Ex2. Demo example of call by reference.

```
#include <stdio.h>
void fun(int *ptr)
{
    *ptr = 300;
}

int main()
{
    int x = 200;
    fun(&x);
    printf("x = %d", x);

    return 0;
}
Output:
x = 300
```

The function fun() expects a pointer ptr to an integer (or an address of an integer).

It modifies the value at the address ptr. The dereference operator * is used to access the value at an address.

In the statement ‘*ptr = 30’, value at address ptr is changed to 30.

The address operator & is used to get the address of a variable of any data type.

In the function call statement ‘fun(&x)’, the address of x is passed so that x can be modified using its address

return Statement

A value may or may not be returned by a function.

Use void as the return type if the function does not need to return any values.

Example without return value:

```
void hello()
{
    printf("hello c");
}
```

If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function. example of function that returns int value from the function.

Example with return value:

```
int get()
{
    return 10;
}
```

In the above example, we have to return 10 as a value, so the return type is int.

If you want to return floating-point value (e.g., 10.2, 3.1, 54.5, etc), you need to use float as the return type of

the method.

```
float get()
{
    return 10.2;
}
```

Now, you need to call the function, to get the value of the function.

Set A . Write programs to solve the following problems

1. Write a function isEven, which accepts an integer as parameter and returns 1 if the number is even, and 0 otherwise. Use this function in main to accept n numbers and check if they are even or odd.
2. Write a function, which accepts a character and integer n as parameter and displays the next n characters.
3. Write a function isPrime, which accepts an integer as parameter and returns 1 if the number is prime and 0 otherwise. Use this function in main to display the first 10 prime numbers.
4. Write a function to convert Binary to Decimal number system.
5. Write a function to convert Binary to Hexadecimal number system.

Signature of the Instructor

Date

Set B . Write programs to solve the following problems

1. Write a function that accepts a character as parameter and returns 1 if it is an alphabet, 2 if it is a digit and 3 if it is a special symbol. In main, accept characters till the user enters EOF and use the function to count the total number of alphabets, digits and special symbols entered.
2. Write a program, which accepts a character from the user and checks if it is an alphabet, digit or punctuation symbol. If it is an alphabet, check if it is uppercase or lowercase and then change the case.
3. Write a menu driven program to perform the following operations till the user selects Exit. Accept appropriate data for each option. Use standard library functions from math.h
 - i. Sine
 - ii. Cosine
 - iii. log
 - iv. ex
 - v. Square Root
 - vi. Exit

4. Write a menu driven program to perform the following operations till the user selects Exit. Accept two complex numbers from the user (real part, imaginary part).

i. ADD ii. SUBTRACT iii. MULTIPLY iv. EXIT

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2: Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 7 : Recursive Functions

Objective :

Use of Recursive functions.

Reading :

You should read the following topics before starting this exercise

1. Recursive definition
2. Declaring and defining a function
3. How to call a function
4. How to pass parameters to a function

Ready References :

Recursion is the process of repeating items in a self-similar way
OR

The process that arises when a function calls a copy of itself to work on a smaller problem is known as recursion. Recursive functions are any functions that call themselves, and calls made by these functions are also known as recursive calls. Multiple recursive calls are made during recursion.

It's crucial to enforce a recursion-based termination condition, though. Although recursion code is shorter than iterative code, it is more challenging to read.

```
void recursion()
{
    recursion(); /* function calls itself */
}
int main() {
    recursion();
}
```

while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

//Ex1. recursion is used to calculate the factorial of a number.

```
#include <stdio.h>
int fact (int);
int main()
{
    int n,f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("factorial = %d",f);
}
int fact(int n)
{
```

```

if (n==0)
{
return 0;
}
else if ( n == 1)
{
return 1;
}
else
{
return n*fact(n-1);
}
}

```

Output

Enter the number whose factorial you want to calculate?5
factorial = 120

Reduce unnecessary calling of function.

Recursion is a simple method for solving issues, however its iterative solution is quite large and intricate.

Set A . Write programs to solve the following problems

1. Write a recursive function to calculate the sum of digits of a number. Use this function in main to accept a number and print sum of its digits.
2. Write a recursive function to calculate product of numbers without using of '*' operator.
3. Write a recursive function to calculate the GCD of two numbers. Use this function in main.
The GCD is calculated as :

$$\text{gcd}(a,b) = a \quad \text{if } b = 0$$

$$= \text{gcd}(b, a \bmod b) \text{ otherwise}$$
4. Write a recursive function for the following recursive definition. Use this function in main to display the first 10 numbers of the following series

$$a_n = 3 \quad \text{if } n = 1 \text{ or } 2$$

$$= 2 * a_{n-1} + 3 * a_{n-2} \text{ if } n > 2$$
5. Write a recursive function to calculate x^y . (Do not use standard library function)

Signature of the Instructor

Date

Set B . Write programs to solve the following problems

1. Write a recursive function to calculate the nth Fibonacci number. Use this function in main to display the first n Fibonacci numbers.
The recursive definition of nth Fibonacci number is as follows:
$$\begin{aligned} \text{fib}(n) &= 1 && \text{if } n = 1 \text{ or } 2 \\ &= \text{fib}(n-2) + \text{fib}(n-1) && \text{if } n > 2 \end{aligned}$$
2. Write a recursive function to calculate the sum of digits of a number till you get a single digit number. Example: 961 -> 16 -> 5. (Note: Do not use a loop).
3. Write a recursive C function to print the digits of a number in reverse order. Use this function in main to accept a number and print the digits in reverse order separated by tab.
Example 3456 – 6543 (Hint: Recursiveprint(n) = print n if n is single digit number
= print n % 10 + tab + Recursiveprint(n/10)

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 8 : Scope of variables

Objective :

Use of variable and scope of variable.

Reading :

Local and global variable

auto, static, register, extern variable and storage classes.

Ready References :

Use of Scope of variables

A scope in any programming is a region of the program where a defined variable can have its

existence and beyond that variable it cannot be accessed.

There are variables can be declared in C programming language –

Inside a function or a block which is called local variables.

Use of Storage classes. Outside of all functions which is called global variables.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used

only by statements that are inside that function or block of code. Local variables are not known

to functions outside their own. The following example shows how local variables are used. Here

all the variables a, b, and c are local to main() function.

//Ex1. Demo example of local variable.

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

```
int main ()
```

```
{
```

```
/* local variable declaration */
```

```
int a, b;
```

```
int c;
```

```
/* actual initialization */
```

```
a = 10;
```

```
b = 20;
```

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

```
printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
```

```
return 0;
```

```
}
```


Global Variables

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. The following program shows how global variables are used in a program.

//Ex2. Demo example of global variable.

```
#include <stdio.h>
/* global variable declaration */
int g;
int main ()
{
/* local variable declaration */
int a, b;
/* actual initialization */
a = 10;
b = 20;
g = a + b;
printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
return 0;
}
```

Computer Programming is all about storing and processing information to generate some results. Therefore, the concept of Variables plays a critical role.

Variables allow us to store information in some memory location and manipulate them using some reference. So, it's important to describe some features of those variables like their scope, default values, lifetime, etc. This is done using Storage Classes, which determine the visibility, lifetime, default values, memory location, and scope of the variables within a C Program.

Thus, Storage Classes in C help us to trace the existence of the variables during Program execution. It is important to know the difference between the scope and lifetime of a variable in C.

The scope of a variable is the region or space in code up to which the variable is available to use. Whereas, the lifetime of a variable is the time for which the variable occupies a valid space in the system memory.

Types of Storage Class

Now, let's see the storage classes in C with examples. C Language provides us with four types of Storage Classes which are named below :

- **Automatic (auto)**
- **External (extern)**
- **Static**
- **Register**

Storage Classes	Default Value	Scope	USE	Lifetime	Memory Location
Auto	Garbage	Local (within block)	Default scope class	Within the function	RAM (System Memory)
Extern	Zero	Global (accessed anywhere)	To declare variable outside the scope	Till termination of main program	RAM (System Memory)
Static	Zero	Local (within block)	To preserves the value of variable in multiple function call	Till termination of main program	RAM (System Memory)
Register	Garbage	Local (within block)	To maintain the frequency used variables so that they are accessible in a faster way.	Within the function	CPU registers

Auto storage class

```
int a;
auto int a;
//Both are the same.
```

//Ex3. Demo example of auto variable.

```
#include <stdio.h>

int main()
{
    int a; //auto
    char b;
    float c;
    printf("%d %c %f",a,b,c); // printing initial default value of automatic variable
                                //s a, b, and c.

    return 0;
}
```

Static storage class

```
static int a;
```

//Ex4. Demo example of static variable.

```
#include<stdio.h>

static char ch;
static int i;
```

```

static float f;
void main ()
{
    printf("%c %d %f",ch,i,f);
        // the initial default value of c, i and f will be printed.
}

```

Extern storage class

```
extern int x;
```

//Ex5. Demo example of extern variable.

```

#include <stdio.h>
int x;
int main()
{
    extern int x; // variable a is defined globally, the memory will not be allocated to x
    printf("%d",x);
}

```

Register storage class

```
register int a;
```

//Ex6. Demo example of register variable.

```

#include <stdio.h>
int main()
{
    register int a; // The initial default value of a is 0.
    printf("%d",a);
}

```

Set A Write programs to solve the following problems

1. Write a program to sum of integer use static variable to store the cumulative sum.
2. Write a program to display prime number upto 'n' using any suitable variable.
3. Write program to define a global integer variable count and increment it in factorial function
4. Write a program to accept numbers till a negative number is entered and Calculate sum of list of numbers.

Signature of the Instructor

Date

Set B . Write programs to solve the following problems

1. Write a program to accept input an integer number of seconds, print as output the equivalent time in hours, minutes and seconds use auto.
(Hint : 7322sec. = 2hrs 2min 2sec)
2. Write a program to accept input an integer number and print stars sequence 3 using suitable variable.
*

3. Write a program to display sum of series $1 + 1/2 + 1/3 + \dots + 1/n$ using suitable variable.

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2: Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 9 : One Dimensional Arrays : Passing array to function

Objective :

One Dimensional Arrays (1D) Operations - declaration, initialization, accessing array elements.

Assignment on Passing 1D arrays to function

Reading :

You should read the following topics before starting this exercise

1. What are arrays?
2. How to declare an array?
3. How to enter data in to array and access the elements of an array.
4. How to initialize an array and how to check the bounds of an array?
5. How to pass the array element to function.

Ready References :

An array is defined as the collection of similar type of data items stored at contiguous memory locations.

Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.

C array is beneficial if you have to store similar elements.

For example, if we want to store the marks of a student in 6 subjects, then we don't need to

define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

The array contains the following properties :

- Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

One Dimensional (1D) Array

An array which has only one subscript is known as **one dimensional array** i.e. int arr[10].

Syntax :

data_type varname[size];

Example :

int arr[5]; // Valid

int arr[5] = {1, "two", 3, 4, "five", 6}; // Invalid, mixed data types

Initialization of array

data-type array_name[]={element1,element2,, element n};

data-type array_name[size]={element-1, element-2,, element-size};

You cannot give more number of initial values than the array size. If you specify less values, the remaining will be initialized to 0.

Example :

int arr[5] = {1, 2, 3}; // Valid, the remaining elements will be set to 0

int arr[5] = {1, 2, 3, 4, 5}; // Valid

int arr[5] = {1, 2, 3, 4, 5, 6}; // Invalid, more elements than specified

a[0]	a[1]	a[2]	a[3]	a[4]
10	15	1	3	20
1000	1002	1004	1006	1008

int a[5]=0;

a[0]	a[1]	a[2]	a[3]	a[4]
0	0	0	0	0
1000	1002	1004	1006	1008

char b[8]= {'C', 'O', 'M', 'P', 'U', 'T', 'E', 'R'}

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]
C	O	M	P	U	T	E	R
1000	1001	1002	1003	1004	1005	1006	1007

Accessing elements of an array

The array index begins from 0 (zero) To access an array element, we need to refer to it as `array_name[index]`.

```
Value = arr[3];
```

This refers to the 4th element in the array

Entering data into an array.

```
for (i=0; i<=9; i++)
    scanf("%d", &arr[i]);
```

Printing the data from an array

```
for(i=0; i<=9; i++)
    printf("%d", arr[i]);
```

Rules For Declaring One Dimensional (1D) Array

- An array variable must be declared before being used in a program.
- The declaration must have a data type (int, float, char, double, etc.), variable name, and subscript.
- The subscript represents the size of the array. If the size is declared as 10, programmers can store 10 elements.
- An array index always starts from 0. For example, if an array variable is declared as `s[10]`, then it ranges from 0 to 9.
- Each array element stored in a separate memory location.

One Dimensional (1D) Array

In C programming, programmers can also initialize the array variable without mentioning the size of an array. The compiler will automatically deduct the size of an array.

```
int student[5] = {89, 76, 68, 91, 84};
```

OR

```
int student[] = {89, 76, 68, 91, 84};
```

Example :

//Ex1. Display Array Elements.

```
#include <stdio.h>
int main()
{
    int s[5] = {89, 76, 98, 91, 84}, i;
    printf("\n---Students marks details--- ");
    for(i = 0; i < 5; i++)
    {
        printf("\ns[%d] = %d ", i + 1, s[i]);
    }
    return 0;
}
---Students marks details---
S[1] = 89
S[2] = 76
S[3] = 98
S[4] = 91
S[5] = 84
```

Note:

The above program illustrates that the declaration and initialization of one dimensional array. The first element of an array is s[0].

The last element of an array is a[4].

We can pass an array to a function using two methods.

- Pass the array element by element
- Pass the entire array to the function

Example

/* Passing the whole array*/

```
void modify(int a[5])
{
    int i;
    for(i=0; i<5 ; i++)
        a[i] = i;
}
```

Sample program to find the largest element of an array.

/* Ex2. Program to find largest number from array */

```
#include<stdio.h>
int main()
{
    int arr[20]; int n;
```



```

void accept(int a[20], int n);
void display(int a[20], int n);
int maximum(int a[20], int n);

printf("How many numbers :");
scanf("%d", &n);
accept(arr,n);
display(arr,n);
printf("The maximum is :%d" , maximum(arr,n));
}
void accept(int a[20], int n)
{
    int i;
    for(i=0; i<n ; i++)
        scanf("%d", &a[i]);
}
void display(int a[20], int n)
{
    int i;
    for(i=0; i<n ; i++)
        printf("%d\t", a[i]);
}
int maximum(int a[20], int n)
{
    int i, max = a[0];
    for(i=1; i<n ; i++)
        if(a[i] > max)
            max = a[i];
    return max;
}

```

Set . Write programs to solve the following problems

1. Write a program to accept n numbers in an array and display elements.
2. Write a program to accept n numbers calculate the range of elements in the array.
3. Write a program to accept n numbers in an array and calculate the average.
4. Write a program to accept n numbers in the range of 1 to 25 and count the frequency of occurrence of each number.
5. Write a program to accept n numbers and print Reverse an Array.
6. Write a function to accept n numbers and display the array in the reverse order. Write separate functions to accept and display.
7. Write a function to accept n numbers and store all prime numbers in an array called prime. Display this array.

Signature of the Instructor

Date

Set B. Write programs to solve the following problems

1. Write a program to accept n numbers and count Even and Odd Elements in an Array
2. Write a program to accept n numbers and print Array Elements Present in Even Position
3. Write a program to accept n numbers and print Array Elements Present in Odd Position
4. Write a program to accept n numbers and print Even Numbers in an Array
5. Write a program to accept n numbers and print Odd Numbers in an Array
6. Write a program to accept n numbers and print the Sum of Even and the Product of Odd Digits
7. Write a program to accept n numbers and print Reverse an Array
8. Write a program to Insert an Element in an Array
9. Write a program to accept n numbers and Delete an Element from an Array and Print a New Array

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 10 : One Dimensional Arrays : Array Operations , Sorting and Searching

Objective :

Use of one dimensional Array Operations

Use of Sorting an array and searching.

Reading :

1. How to enter data in to array and access the elements of an array
2. How to exchange sort, bubble sort (ie arrange the data in ascending and descending order))

Ready References :

Insertion, deletion, searching, display, traversal, and updating are the fundamental operations

in arrays. These actions are typically taken to report the array's status or to change the data

within the array.

Following are the basic Array operations.

- Traverse – Print each element in the array one by one.
- Insertion – At the specified index, adds an element.
- Deletion – The element at the specified index is deleted.
- Search – Uses the provided index or the value to search for an element.
- Update – The element at the specified index is updated.

/*Ex1. Write a program for finding the largest number in an array */

```
#include<stdio.h>
void main()
{
int arr[30], i, j, n, large;
printf("Enter the number of elements in the array");
scanf("%d", &n);
for(i=0; i<n; i++)
{
    printf("Enter a number");
    scanf("%d", &arr[i]);
}
large=arr[0];
for(i=1; i<n; i++)
{
    if(arr[i]>large)
        large=arr[i];
}
printf("The largest number in the array is : %d",large);
}
```

Sorting an array in ascending order means arranging the elements from smallest element to largest element.

Example

arr = { 170, 45, 75, 90, 802, 24, 2, 66 }	//unsorted array element
arr = { 2, 24, 45, 66, 75, 90, 170, 802 }	//sorted array element

/*Ex1. program to accept a set of numbers and arrange them in a descending order. */

```
#include <stdio.h>
void main ()
{
    int arr[30],i, j, temp, n;
    printf("Enter the value of N\n");
    scanf("%d", &n);
    printf("Enter the numbers \n");
    for (i = 0; i < n; ++i)
        scanf("%d", &number[i]);
    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            if (arr[i] < arr[j])
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }

    printf("The numbers arranged in descending order are given below\n");
    for (i = 0; i < n; ++i)
    {
        printf("%d\n", arr[i]);
    }
}
```

Set A. Write programs to solve the following problems

1. Write a program to find the union and intersection of the two sets of integers (store it in two arrays).
2. Write a program to accept n numbers and find the largest element or number in an array using function
3. Write a program to accept n numbers and print the Second Largest and Second Smallest Array Element.
4. Write a function for Linear Search, which accepts an array of n elements and a key as parameters and returns the position of key in the array and - 1 if the key is not found. Accept n numbers from the user, store them in an array. Accept the key to be searched and search it using this function. Display appropriate messages.
5. Write a function, which accepts an integer array and an integer as parameters and counts the occurrences of the number in the array.
Example: Input 1 5 2 1 6 3 8 2 9 1 5 1 3 0
Occurrence : Number : 1
Output : 1 occurs 4 times
6. Write a program to accept n numbers sort the elements using bubble sort arrange in ascending order.
7. Write a program to accept n numbers sort the elements using arrange in descending order using function.
8. Write a program to accept n numbers sort the elements using insertion sort arrange in ascending order using function.

Signature of the Instructor

Date

Set B. Write programs to solve the following problems

1. Write a program to accept n numbers and print Reverse an Array using function.
2. Write a program to accept the two one dimensional array from user and find and print common elements from two arrays using function.
3. Write a program to insert an element in an array accept the element and position where user want using function.
4. Write a program to accept n numbers sort the elements using selection sort arrange in descending order.

5. Write a program to merge two sorted arrays into a third array such that the third array is also in the sorted order using function.

a1	10	25	90					
a2	9	16	22	26	100			
a3	9	10	16	22	25	26	90	100

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 11 : Two Dimensional Arrays : Basic Operations , Passing 2D arrays to functions

Objective :

To demonstrate Two and multidimensional array (2D) operations
To demonstrate use of 2-D arrays and functions.

Reading :

You should read the following topics before starting this exercise

1. How to declare and initialize two-dimensional array
2. How to accessing elements of two dimensional array.
3. Usage of two dimensional arrays
4. How to declare and initialize two-dimensional array
5. Accessing elements
6. Usage of two dimensional arrays

Ready References :

An array consisting of two subscripts is known as two-dimensional array.

These are often known as array of the array. In two dimensional arrays the array is divided into rows and columns.

These are well suited to handle a table of data. In 2-D array we can declare an array as :

Declaration:-

Syntax: data_type array_name[row_size][column_size];

Ex:- int arr[3][3];

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]
arr[2][0]	arr[2][1]	arr[2][2]

where first index value shows the number of the rows and second index value shows the number the columns in the array

Initializing two-dimensional arrays:

Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration

with a list of initial values enclosed in braces.

Ex: int a[2][3]={0,0,0,1,1,1};

initializes the elements of the first row to zero and the second row to one.

The initialization is done row by row

The above statement can also be written as int a[2][3] = {{ 0,0,0},{1,1,1}};
by surrounding the elements of each row by braces.

We can also initialize a two-dimensional array in the form of a matrix as shown below

Example :

```
int a[2][3]=
{
    {0,0,0},
    {1,1,1}
};
```

When the array is completely initialized with all values, explicitly we need not specify the size of the first dimension.

Example :

```
int a[][3] =
{
    {0,2,3},
    {2,1,2}
};
```

If the values are missing in an initializer, they are automatically set to zero.

Example :

```
int a[2][3] =
{
    {1,1},
    {2}
};
```

Will initialize the first two elements of the first row to one, the first element of the second row to two and all other elements to zero.

//Ex1.Sample program of read and display the 2D (two dimensional)array

```
#include<stdio.h>
void main()
{
    int a[3][3],lim,i,j;
    printf("\nEnter the limit = ");//accept the limit from user
    scanf("%d",&lim);

    printf("\nEnter the elements ="); //accept the elements from user
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            scanf("%d",&a[i][j]);
        }//End of j loop
    }//End of i loop
```



```

printf("\nArray elements are ="); //display the elements with index no.
for(i=0;i<lim;i++)
{
    for(j=0;j<lim;j++)
    {
        printf("\na[%d][%d] = %d ",i,j,a[i][j]);
    }//End of j loop
} //End of i loop

printf("\nArray elements are =\n"); //display the elements in tabular format
for(i=0;i<lim;i++)
{
    for(j=0;j<lim;j++)
    {
        printf("%d ",a[i][j]);
    }//End of j loop

    printf("\n");

} //End of i loop

} //END main()

```

/*OUTPUT

```

[root@localhost C]# cc arr2d.c
[root@localhost C]# ./a.out

```

Enter the limit = 3

Enter the elements =

```

1
3
5
7
9
11
13
15
17

```

Array elements are =

```

a[0][0] = 1
a[0][1] = 3
a[0][2] = 5
a[1][0] = 7
a[1][1] = 9

```

```
a[1][2] = 11
a[2][0] = 13
a[2][1] = 15
a[2][2] = 17
Array elements are =
1 3 5
7 9 11
13 15 17 */
```

The column size of a two-dimensional array in C must be specified in the function parameters when the array is passed to a function. However, in order to handle arrays of various sizes, the row size can be left undefined or supplied as a function argument.

A two-dimensional array facilitates tabular data storage and manipulation. Matrices, tables, grids, and other tabular data structures can be represented with them. They are kept in memory regions that are close together. Thus, they promote effective memory use.

Syntax :

```
void functionName(dataType arrayName[][numCols], int numRows)
{
// Function body
}
```

Here, dataType represents the data type of the elements defined in the array, ArrayName represents the array's name, and numCols represents the number of columns in a particular array.

//Ex2. Sample program of read and display the 2D (two dimensional)array using function

```
#include<stdio.h>
void accept(int a[3][3],int lim); //prototype of function
void display(int a[3][3],int lim);

void main()
{
    int a[3][3],lim;
    printf("\nEnter the limit = ");//accept the limit from user
    scanf("%d",&lim);

    //function declaration of read and display
    accept(a,lim);
    display(a,lim);

} //END main()
```

```

//function definition of read and display
void accept(int a[3][3],int lim)
{
    int i,j;
    printf("\nEnter the elements ="); //accept the elements from user
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            scanf("%d",&a[i][j]);
        }//End of j loop
    }//End of i loop
}//End of accept()

void display(int a[3][3],int lim)
{
    int i,j;
    printf("\nArray elements are ="); //display the elements with index no.
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            printf("\na[%d][%d] = %d ",i,j,a[i][j]);
        }//End of j loop
    }//End of i loop

    printf("\nArray elements are =\n"); //display the elements in tabular format
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            printf("%d ",a[i][j]);
        }//End of j loop
        printf("\n");

    }//End of i loop

}//End of accept()

```

/*OUTPUT

```

[root@localhost C]# cc arr2dfunc.c
[root@localhost C]# ./a.out

```

Enter the limit = 2

```

Enter the elements =
11
33
44
55
Array elements are =
a[0][0] = 11
a[0][1] = 33
a[1][0] = 44
a[1][1] = 55
Array elements are =
11 33
44 55
*/

```

//Ex3. Sample program of read and display the 2D (two dimensional) array using function

```

#include<stdio.h>
void accept(int a[3][3],int lim); //prototype of function
void display(int a[3][3],int lim);
void maximum(int a[3][3],int lim);
void main()
{
    int a[3][3],lim;
    printf("\nEnter the limit = ");//accept the limit from user
    scanf("%d",&lim);

    //function declaration of read and display
    accept(a,lim);
    display(a,lim);
    maximum(a,lim);
} //END main()

//function defination of read and display

void accept(int a[3][3],int lim)
{
    int i,j;
    printf("\nEnter the elements ="); //accept the elements from user
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            scanf("%d",&a[i][j]);

```

```

    }//End of j loop
  }//End of i loop
} //End of accept()

void display(int a[3][3],int lim)
{
    int i,j;
    printf("\nArray elements are ="); //display the elements with index no.
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            printf("%d ",a[i][j]);
        }//End of j loop
        printf("\n");
    }//End of i loop
} //End of display()

void maximum(int a[3][3],int lim)
{
    int i,j,max;
    max = a[0][0];
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            if(max < a[i][j])
            {
                max = a[i][j];
            }
        }//End of if loop
    }//End of j loop
} //End of i loop
printf("\nmaximum of array element is = %d ",max);
} //End of maximum()

```

/*OUTPUT

```

[root@localhost C]# cc arr2dfuncmax.c
(base) [root@localhost C]# ./a.out

```

Enter the limit = 3

Enter the elements =

22

44

55
66
77
88
99
10
89

Array elements are =
22 44 55
66 77 88
99 10 89

maximum of array element is = 99 */

Set A . Write C programs for the following problems.

1. Write a program to accept a matrix A of size mxn and store its transpose in matrix B. Display transpose matrix .
2. Write a program to add and multiply two matrices to accept, display, add and multiply the matrices. Perform necessary checks before adding and multiplying the matrices.
3. Write a program to accept a matrix A of size mxn and display identical matrix.

Eg

1	0	0
0	1	0
0	0	1

4. Write a program to accept a matrix A of size m x n and store its transpose in matrix B. Display matrix B. Write separate functions.
5. Write a menu driven program to perform the following operations on a square matrix. Write separate functions for each option.
 - i. Display the trace of the matrix (sum of diagonal elements).
 - ii. Check if the matrix is an upper triangular matrix.
 - iii. Check if the matrix is a lower triangular matrix.
 - iv. Exit

2	7	6	15
9	5	1	15
4	3	8	15
15	15	15	15

Signature of the Instructor

Date

Set B . Write C programs for the following problems.

1. Write a program to check if a number has three consecutive 5s. If yes, print YES, else print NO.
Example: Number: 1353554 Result: NO
Number: 345559 Result: YES
2. Write a program, to accept the array elements and store in another array which will print two digit numbers whose sum of both digit is multiple of seven.
Example: 16,25,34.....
3. Write a program to accept two array and Merge Arrays in Descending Order.
4. A magic square of order n is an arrangement of n^2 numbers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant. A normal magic square contains the integers from 1 to n^2 . The magic constant of a magic square depends on n and is $M(n) = (n^3+n)/2$. For n=3,4,5, the constants are 15, 34, 65 resp. Write a program to generate and display a magic square of order n.

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Assignment 12 : Two Dimensional Arrays : Matrix operations

Objective :

To demonstrate use of 2-D arrays and matrix operation.

Reading :

1. You should read the following topics before starting this exercise
2. Usage of two dimensional arrays

Ready References :

In C, memory is allotted for two-dimensional arrays in contiguous blocks, which are determined by multiplying the size of the data type by the product of rows and columns. storing and working with graphical data, such as photos. modifying and storing matrices for use in calculations involving linear algebra putting board games like chess and checkers into practice. working with and displaying data in spreadsheet applications.

Definition of Multidimensional Array

A multidimensional array in C is an array comprising multiple dimensions. It is essentially an array of arrays.

Syntax :

dataType arrayname[size1][size2]...[sizeN];

//Ex1. Sample program of read and display the 2D (two dimensional) array

```
#include<stdio.h>
void main()
{
    int a[3][3],lim,i,j,sum = 0;
    printf("\nEnter the limit = "); //accept the limit from user
    scanf("%d",&lim);

    printf("\nEnter the elements ="); //accept the elements from user
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            scanf("%d",&a[i][j]);
        } //End of j loop
    } //End of i loop
    printf("\nArray elements are ="); //display the elements with index no.
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            printf("\na[%d][%d] = %d ",i,j,a[i][j]);
        } //End of j loop
    } //End of i loop
```



```

printf("\nArray elements are =\n"); //display the elements in tabular format

for(i=0;i<lim;i++)
{
    for(j=0;j<lim;j++)
    {
        printf("%d ",a[i][j]);
    }//End of j loop
    printf("\n");
} //End of i loop
printf("\nDiagonal Array elements are =\n");
    //display the elements in tabular format
for(i=0;i<lim;i++)
{
    for(j=0;j<lim;j++)
    {
        if(i == j)
        {
            printf("%d ",a[i][j]);
            sum = sum + a[i][j];

        } //End of if loop
    } //End of j loop
} //End of i loop
printf("\nDiagonal Sum = %d",sum);
} //END main()

```

/*OUTPUT

```
[root@localhost C]# cc arr2diasum.c
```

```
[root@localhost C]# ./a.out
```

```
Enter the limit = 3
```

```
Enter the elements =
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
Array elements are =
```

```
a[0][0] = 1
```

```
a[0][1] = 2
```

```
a[0][2] = 3
```

```
a[1][0] = 4  
a[1][1] = 5  
a[1][2] = 6  
a[2][0] = 7  
a[2][1] = 8  
a[2][2] = 9
```

Array elements are =

1 2 3

4 5 6

7 8 9

Diagonal Array elements are =

1 5 9

Diagonal Sum = 15*/

/*Ex2. Sample program of read and display Inverse of the 2D (two dimensional) array using function */

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

```
void accept(int a[3][3],int lim); //prototype of function
```

```
void display(int a[3][3],int lim);
```

```
void inverse(int a[3][3],int lim);
```

```
void main()
```

```
{
```

```
    int a[3][3],lim;
```

```
    printf("\nEnter the limit = "); //accept the limit from user
```

```
    scanf("%d",&lim);
```

```
    //function declaration of read and display
```

```
    accept(a,lim);
```

```
    display(a,lim);
```

```
    inverse(a,lim);
```

```
}//END main()
```

```
    //function definition of read and display
```

```
void accept(int a[3][3],int lim)
```

```
{
```

```
    int i,j;
```

```
    printf("\nEnter the elements ="); //accept the elements from user
```

```
    for(i=0;i<lim;i++)
```

```
    {
```

```

for(j=0;j<lim;j++)
{
    scanf("%d",&a[i][j]);
} //End of j loop
} //End of i loop
} //End of accept()

void display(int a[3][3],int lim)
{
    int i,j;
    printf("\nArray elements are ="); //display the elements with index no.
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            printf("\na[%d][%d] = %d ",i,j,a[i][j]);
        } //End of j loop
    } //End of i loop

    printf("\nArray elements are =\n"); //display the elements in tabular format
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            printf("%d ",a[i][j]);
        } //End of j loop
        printf("\n");
    } //End of i loop
} //End of display()

void inverse(int a[3][3],int lim)
{
    int i,j;
    printf("\nArray elements are ="); //display the elements with index no.
    for(i=0;i<lim;i++)
    {
        for(j=0;j<lim;j++)
        {
            printf("%d ",a[j][i]);
        } //End of j loop
        printf("\n");
    } //End of i loop
} //End of inverse()

```

/*OUTPUT

```
[root@localhost C]# cc arr2dinverse.c
```

```
[root@localhost C]# ./a.out
```

```
Enter the limit = 3
```

```
Enter the elements =
```

```
11
```

```
22
```

```
33
```

```
44
```

```
55
```

```
66
```

```
77
```

```
88
```

```
99
```

```
Array elements are =
```

```
a[0][0] = 11
```

```
a[0][1] = 22
```

```
a[0][2] = 33
```

```
a[1][0] = 44
```

```
a[1][1] = 55
```

```
a[1][2] = 66
```

```
a[2][0] = 77
```

```
a[2][1] = 88
```

```
a[2][2] = 99
```

```
Array elements are =
```

```
11 22 33
```

```
44 55 66
```

```
77 88 99
```

```
Array elements are =
```

```
11 44 77
```

```
22 55 88
```

```
33 66 99
```

```
*/
```

//Ex. 3 : Accept the two matrices from user and display addition of Matrices

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

```
int main()
```

```
{
```

```
    int i, j, rows, columns, a[10][10], b[10][10];
```

```
    int arr[10][10];
```

```
    printf("\n Please Enter Number of rows and columns : ");
```

```
    scanf("%d %d", &i, &j);
```

```
    printf("\n Please Enter the First Elements\n");
```

```
    for(rows = 0; rows < i; rows++)
```

```
    {
```

```
        for(columns = 0; columns < j; columns++)
```

```

        {
            scanf("%d", &a[rows][columns]);
        }
    }

    printf("\n Please Enter the Second Elements\n");
    for(rows = 0; rows < i; rows++)
    {
        for(columns = 0; columns < j; columns++)
        {
            scanf("%d", &b[rows][columns]);
        }
    }

    for(rows = 0; rows < i; rows++)
    {
        for(columns = 0; columns < j; columns++)
        {
            arr[rows][columns] = a[rows][columns] + b[rows][columns];
        }
    }
    printf("\n The Sum of Two a and b = a + b \n");
    for(rows = 0; rows < i; rows++)
    {
        for(columns = 0; columns < j; columns++)
        {
            printf("%d \t ", arr[rows][columns]);
        }
    }
    printf("\n");
}
return 0;
}

```

Set A . Write C programs for the following problems.

1. Write a program to accept a matrix A of size r x c and store its transpose in matrix B. Display matrix B also display sum of diagonal elements. Write separate functions.
2. Write a program to accept two matrices , write separate functions to accept, display, the matrices. check two matrices are identical or not
3. Write a menu driven program to perform the following operations on a square matrix.
 - Check if the matrix is symmetric.
 - Check if it is an identity matrix.

Signature of the Instructor

Date

Set B . Write C programs for the following problems.

1. Write a program to accept an $m \times n$ matrix and Find the sum of each row and each column of matrix of size $m \times n$ and display an $m+1 \times n+1$ matrix such that the $m+1^{\text{th}}$ row contains the sum of all elements of corresponding row and the $n+1^{\text{th}}$ column contains the sum of elements of the corresponding column.

Example:

A

10	2	3
4	15	6
7	8	19

B

10	2	3	15
4	15	6	25
7	8	19	34
21	25	28	74

2. Write a menu driven program to perform the following operations on a square matrix. Write separate functions for each option
 - Addition,
 - Subtraction
 - Multiplication
 - Exit

Signature of the Instructor

Date

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-Charge

Case Study or Mini Project

1. Accept the price and quantity of items as an input. Write a C function to calculate the sum of the prices. Write another C function to calculate the discount according to the following rules:

For total less than Rs.1000, discount is 5%.

For total greater than Rs.1000 but less than Rs.5000, discount is 10%.

For total greater than Rs.5000, discount is 15%.

Write another function to print the individual item prices, total, discount and the final price.(Hint : Use suitable array)

Example:

If the prices are as follows:

Item 1: 200

Item 2: 400

Item 3: 200

Item 4: 10

Item 5: 50

And the quantities are:

Item 1: 1

Item 2: 1

Item 3: 3

Item 4: 5

Item 5: 2

Then you should print:

Item	Price	Quantity	Subtotal
------	-------	----------	----------

Item 1	200	1	200
--------	-----	---	-----

Item 2	400	1	400
--------	-----	---	-----

Item 3	200	3	600
--------	-----	---	-----

Item 4	10	5	50
--------	----	---	----

Item 5	50	2	100
--------	----	---	-----

TOTAL			1350
--------------	--	--	-------------

Discount 10%			-135
---------------------	--	--	-------------

GRAND TOTAL			1215
--------------------	--	--	-------------

2. Compute taxes for a given income with tax slab rates as follows...

slab 1... 0-2500: 0%

slab 2... 2500-5000: 10%

slab 3... 5000-10000: 20%

slab 4... 10000+: 30%

(for each range of a given income, the tax rate is different)

e.g. input: $x = 5200$

divisions are 0-2500, 2500-5000, 5000-5200

calculation:

0-2500 of x :

0% of 2500 = 0

2500-5000 of x :

10% of 2500 = 250

5000-1000 of x :

20% of 200 = 40

output: 290

3. Project : Electricity Bill System

The technique for tracking power bills is specifically made to determine the overall amount of electricity used. The user must input the complete number of units used in this system before the total value is shown. The entire project was developed using the "C" programming language, utilizing a variety of variables and strings. Users find it straightforward to use and comprehend. The project contains no error or warning contents. The user won't have any trouble using and navigating the design because it is so straight forward.

4. Customer Billing System Project

The Customer Billing System Project is a simple console application designed to demonstrate the real-world applications and capabilities of the C programming language. It also aims to produce an application that can be used to bill customers in any department store, retail outlet, café, etc. customers' information such as name, address, phone number, paid amount, due amount, payment date, and so on.

The following are the main user defined functions utilized in this C project:

void inputdata()

void writedata()

void search()

void output()

5. Bank Management System Project

The project allows users to create new accounts, edit information in existing accounts, see and manage transactions, confirm account data, remove accounts, and peruse a customer list.

Functions in the Bank Management System

menu() : This function displays a welcome screen or menu that lets you do the different banking operations mentioned below.

new acc() : This function is used to create a new customer account. In addition to other personal and financial data, it demands the customer's name, date of birth, citizenship number, address, and phone number. A range of deposit accounts are available to you, such as current, savings, fixed for one, two, or three years.

view list() : Shows an itemised list. Through this tool, you can obtain the customer's financial details, such as the name, address, phone number, and account number that were provided at the time the account was opened.

edit() : This function can be used to change the phone number and address connected to a particular customer account.

transact() : This function lets you put money into and take money out of a particular client account.

remove() : This service allows the deletion of a client account.

The function see() : lets you take a peek at something. This function displays the following information: account number, name, citizenship number, date of birth, age, address, phone number, type of account, amount deposited, and date of deposit. Additionally, it displays the interest generated on a certain kind of account.