BHARATI VIDYAPEETH DEEMED UNIVERSITY
COLLEGE OF ENGINEERING, PUNE - 43

DEPARTMENT OF COMPUTER ENGINEERING

Lab Manual

Programming Principals and Paradigms

B.Tech (Computer)Sem-I

VISION OF THE INSTITUTE

"To be World Class Institute for Social Transformation through Dynamic Education"

MISSION OF THE INSTITUTE

- To provide quality technical education with advanced equipment's, qualified faculty Members, infrastructure to meet needs of profession and society.
- To provide an environment conducive to innovation, creativity, research, and entrepreneurial leadership.
- To practice and promote professional ethics, transparency and accountability for social community, economic and environmental conditions.

VISION OF THE DEPARTMENT

"To pursue and excel in the endeavour for creating globally recognized Computer Engineers through Quality education".

MISSION OF THE DEPARTMENT

1. To impart engineering knowledge and skills confirming to a dynamic curriculum.

2. To develop professional, entrepreneurial & research competencies encompassing continuous intellectual growth.

3. To produce qualified graduates exhibiting societal and ethical responsibilities in working environment.

PROGRAM EDUCATIONAL OBJECTIVES

1. Demonstrate technical and professional competencies by applying engineering fundamentals, computing principles and technologies.

2. Learn, Practice, and grow as skilled professionals/ entrepreneur/researchers adapting to the evolving computing landscape.

3. Demonstrate professional attitude, ethics, understanding of social context and interpersonal skills leading to a successful career.

PROGRAM SPECIFIC OUTCOMES

1. To apply fundamental knowledge and technical skills towards solving Engineering problems.
2. To employ expertise and ethical practise through continuing intellectual growth and adapting to the working environment.

PROGRAM OUTCOMES

1. To apply knowledge of computing and mathematics appropriate to the domain.
2. To logically define, analyse and solve real world problems.
3. To apply design principles in developing hardware/software systems of varying complexity that meet the specified needs.
4. To interpret and analyse data for providing solutions to complex engineering problems.
5. To use and practise engineering and IT tools for professional growth.
6. To understand and respond to legal and ethical issues involving the use of technology for societal benefits.
7. To develop societal relevant projects using available resources.
8. To exhibit professional and ethical responsibilities.
9. To work effectively as an individual and a team member within the professional environment.
10. To prepare and present technical documents using effective communication skills.
11. To demonstrate effective leadership skills throughout the project management life cycle.
12. To understand the significance of lifelong learning for professional development.

GENERAL INSTUCTIONS:

- Equipment in the lab is meant for the use of students. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care.
- Students are required to carry their reference materials, files and records with completed assignment while entering the lab.
- Students are supposed to occupy the systems allotted to them and are not supposed to talk or make noise in the lab.

- All the students should perform the given assignment individually.
- Lab can be used in free time/lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- All the Students are instructed to carry their identity cards when entering the lab.
- Lab files need to be submitted on or before date of submission.
- Students are not supposed to use pen drives, compact drives or any other storage devices in the lab.
- For Laboratory related updates and assignments students should refer to the notice board in the Lab.

COURSE NAME: PROGRAMMING PRINCIPALS AND PARADIGMS

WEEKLY PLAN:

Week No.	Practical/Assignment Name	Problem Definition
1	Programming Principles paradigm & Basic Concepts of c	Study Programming paradigm &cConcepts
2	Study Concept of conditional statements	Implement program using Conditional statements(If, if-else, nested if)
3	Study the concept of switch case statement	Implement Programs using Switch case statement
4	Study of loop statements(while, do-while, for)	Implement program using loop statements(while, do-while, for)
5	Study the concept of Array	Implement program using Array
6	Study the concept of functions	Implement program using Functions
7	Study the concept of Structures	Implement program using Structures
8	Study the concept of Pointers	Implement program using pointers

EXAMINATION SCHEME

Term Work: 50 Marks
Total: 50Marks
Minimum Marks required: 20 Marks

PROCEDURE OF EVALUATION

Each practical/assignment shall be assessed continuously on the scale of 25 marks. The distribution of marks as follows.

Sr. No	Evaluation Criteria	Marks for each Criteria	Rubrics
1	**Timely Submission**	**14**	➢ Punctuality reflects the work ethics. Students should reflect that work ethics by completing the lab assignments and reports in a timely manner without being reminded or warned.
2	**Presentation**	**12**	➢ Student are expected to write the technical document (lab report) in their own words. The presentation of the contents in the lab report should be complete, unambiguous, clear, and understandable. The report should document approach/algorithm/design and code with proper explanation.
3	**Understanding**	**24**	➢ Correctness and Robustness of the code is expected. The Learners should have an in-depth knowledge of the practical assignment performed. The learner should be able to explain methodology used for designing and developing the program/solution. He/she should clearly understand the purpose of the assignment and its outcome.

LABORATORY USAGE

Students use computers for executing the lab experiments, document the results and to prepare the technical documents for making the lab reports.

OBJECTIVE

The objective of this lab is to make students aware and practice implementation of various data structures by designing algorithms and implementing programs in C.

PRACTICAL PRE-REQUISITE

The Students should have

1. Prior programming experience not required, but understanding of computational approaches to problem solving and logical aptitude is required.
2. Basic mathematical ability.

SOFTWARE REQUIREMENTS

- C compiler.

COURSE OUTCOMES

1. Identify and express the four programming paradigms**.**
2. Discuss various Operating Systems and Software Tools**.**
3. Define Data types and use them in data processing programs.
4. Implement different kinds of Array data structures
5. Implement Structures and Pointers
6. Implement File Handling in C Handling and Exception handling

HOW OUTCOMES ARE ASSESSED?

Outcome	Assignment Number	Level	Proficiency evaluated by
Identify and express the four programming paradigms**.**	**1,2,3,4,5,6,7,8**	**3,2,3,2,3,1,3,3**	Performing Practical and reporting results
Discuss various Operating Systems and Software Tools**.**	**1,2,3,4,5,6,7,8,**	**3,3,2,2,2,2,2**	Problem definition&Performing Practical and reporting results
Define Data types and use them in data processing programs.	**1,2,3,4**	**3,3,3,3**	Performing experiments and reporting results
Implement different kinds of Array data structures	**5,7**	**3,3**	Performing experiments and reporting results
Implement Structures and Pointers	**7,8**	**3,3**	Performing experiments and reporting results
Implement File Handling in C Handling and Exception handling	**7,8**	**3,3**	Performing experiments and reporting results

CONTRIBUTION TO PROGRAM OUTCOMES

COs		Program Outcomes												PSOs	
		PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2
CO 1	Identify and Express the four programming paradigms		2	1		1	1								1
CO 2	Discuss various Operating Systems and Software Tools		2	3	1			1		1	2		1		2
CO 3	Define Data types and use them in data processing programs	2	2	2	3	1			1			1	1	3	
CO 4	Implement different kinds of Array data structures	2	3	3	2	1			1			1	1	2	
CO 5	Implement Structures and Pointers	2	3	3	2	1			1			1	1	2	
CO 6	Implement File Handling in C Handling and Exception handling	2	3	3	2	1			1			1	1	2	2

DESIGN EXPERIENCE GAINED

The students gain moderate design experience by creating algorithms using data structures and convert that algorithm to executable code.

LEARNING EXPERIENCE GAINED

The students learn both soft skills and technical skills while they are undergoing the practical sessions. The soft skills gained in terms of communication, presentation and behavior. While technical skills they gained in terms of programming.

LIST OF PRACTICAL ASSIGNMENTS:

1. Study the basic concepts of C Programming. A)Write a program to print Hello world B)Write a program to program Sum and Difference of two numbers c) Write A program to print area of circle and Square
2. Study and Implement program using If, If-else and nested If statement. A) Implement program To Find greatest number among 3 numbers Using If-else statement B) Implement a program to print the result of student using nested IF statement
3. Study and implement program using switch case statement. A) Implement menu driven Program to find out the entered day by user. B) Implement menu driven program to find weather the entered character is vowel or constant.
4. Study and implement program using loop statement(do-while, while, for) A)Implement program to find sum of positive integers using for loop B)Implement program to display even numbers from 1 to 50 using while loop C) Implement program to display odd numbers from 1 to 50 using do-while loop
5. Study and implement program using function. A) Implement menu driven program to find Arithmetic operation using functions B) Implement program to calculate factorial of number using function recursion
6. Study and implement program using array A) Implement program to find average of marks using array B) Implement program to calculate addition of 2 matrices using 2D- array
7. Study and implement program using structure A) Implement program to print record of students using Structure
8. Study and implement program using pointer A) Implement program to swap 2 numbers using pointers

ASSIGNMENT NO.1

Aim: Study the basic concepts of C Programming.

A) Write a program to print Hello world
B) Write a program to program Sum and Difference of two numbers
C) Write a program to print area of circle and Square

Theory:

Programming paradigms are a way to classify programming languages based on their features. Languages can be classified into multiple paradigms.

Common programming paradigms include:

- **Imperative which allows side effects,**
  - In computer science, **imperative programming** is a programming paradigm that uses statements that change a program's state. In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform. Imperative programming focuses on describing *how* a program operates.
  - The term is often used in contrast to declarative programming, which focuses on *what* the program should accomplish without specifying *how* the program should achieve the result.
- **Functional which disallows side effects,**
  - In computer science, **functional programming** is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions[1] or declarations[2] instead of statements. In functional code, the output value of a function depends only on the arguments that are passed to the function, so calling a function *f* twice with the same value for an argument *x* will produce the same result *f(x)* each time; this is in contrast to procedures depending on a local or global state, which may produce different results at different times when called with the same arguments but a different program state. Eliminating side effects, i.e. changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behaviour of a program, which is one of the key motivations for the development of functional programming.
  - Functional programming has its origins in lambda calculus, a formal system developed in the 1930s to investigate computability, the Entscheidungs problem, function definition, function application, and recursion. Many functional programming languages can be viewed as elaborations on the lambda calculus
- **D which does not state the order in which operations execute,**

- **declarative programming** is a programming paradigm—a style of building the structure and elements of computer programs—that expresses the logic of a computation without describing its control flow.[1]
- Many languages that apply this style attempt to minimize or eliminate side effects by describing *what* the program must accomplish in terms of the problem domain, rather than describe *how* to accomplish it as a sequence of the programming language primitives . This is in contrast with imperative programming, which implements algorithms in explicit steps.
- Declarative programming often considers programs as theories of a formal logic, and computations as deductions in that logic space. Declarative programming may greatly simplify writing parallel programs.[3]
- Common declarative languages include those of database query languages (e.g., SQL, XQuery), regular expressions, logic programming, functional programming, and configuration management systems.

- **Object-oriented which groups code together with the state the code modifies,**
  - OO is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as *attributes;* and code, in the form of procedures, often known as *methods.* A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another.[1][2] There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type.
  - Many of the most widely used programming languages (such as C++, Object Pascal, Java, Python etc.) are multi-paradigm programming languages that support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming. Significant object-oriented languages include Java, C++, C#, Python, PHP, Ruby, Perl, Object Pascal, Objective-C, Dart, Swift, Scala, Common Lisp, and Smalltalk.
- **procedural which groups code into functions,**
  - **Procedural programming** is a programming paradigm, derived from structured programming, based upon the concept of the *procedure call*. Procedures, also known as routines, subroutines, or functions (not to be confused with mathematical functions, but similar to those used in functional programming), simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or itself. The first major procedural programming languages first appeared circa 1960, including FORTRAN, ALGOL, COBOL and BASIC.Pascal and C were published closer to the 1970s, while Ada was released in 1980.[1] Go is an example of a more modern procedural language, first published in 2009.
  - Computer processors provide hardware support for procedural programming through a stack register and instructions for calling procedures and returning from them.

Hardware support for other types of programming is possible, but no attempt was commercially successful (for example Lisp machines or Java processors).

- **logic** which has a particular style of execution model coupled to a particular style of syntax and grammar,
  - which has a particular style of execution model coupled to a particular style of syntax and grammar, and
  - **Logic programming** is a type of programming paradigm which is largely based on formal logic. Any program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain. Major logic programming language families include Prolog, Answer set programming (ASP) and Datalog. In all of these languages, rules are written in the form of *clauses*:
  - H :- B_1, …, B_n.
  - and are read declaratively as logical implications:
  - H if B_1 and … and B_n.
  - H is called the *head* of the rule and B_1… B_n is called the *body*. Facts are rules that have no body, and are written in the simplified form:
- **Symbolic programming** which has a particular style of syntax and grammar.
  - Programming which has a particular style of syntax and grammar.[1][2][3]
  - In computer programming, **symbolic programming** is a programming paradigm in which the program can manipulate its own formulas and program components as if they were plain data.[1]
  - Through symbolic programming, complex processes can be developed that build other more intricate processes by combining smaller units of logic or functionality. Thus, such programs can effectively modify themselves and appear to "learn", which makes them better suited for applications such as artificial intelligence, expert systems, natural language processing, and computer games.
  - Languages that support symbolic programming include Wolfram Language,LISP and Prolog.

C – Programming Basics

This C programming basics section explains a simple “Hello World” C program. Also, it covers below basic topics as well, which are to be known by any C programmer before writing a C program.

- C programming basic commands to write a C program
- A simple C program with output and explanation
- Steps to write C programs and get the output
- Creation, Compilation and Execution of a C program
- How to install C compiler and IDE tool to run C programming codes
- Basic structure of a C program
- Example C program to compare all the sections
- Description for each section of the C program

C programs (Click here for more C programs) with definition and output – C program for Prime number, Factorial, Fibonacci series, Palindrome, Swapping 2 numbers with and without temp variable, sample calculator program and sample bank application program etc.

1. C PROGRAMMING BASICS TO WRITE A C PROGRAM:

Below are few commands and syntax used in C programming to write a simple C program. Let's see all the sections of a simple C program line by line.

C Basic commands	Explanation
#include <stdio.h>	This is a preprocessor command that includes standard input output header file(stdio.h) from the C library before compiling a C program
int main()	This is the main function from where execution of any C program begins.
{	This indicates the beginning of the main function.
/*_some_comments_*/	whatever is given inside the command "/* */" in any C program, won't be considered for compilation and execution.
printf("Hello_World! ");	printf command prints the output onto the screen.
getch();	This command waits for any character input from keyboard.
return 0;	This command terminates C program (main function) and returns 0.
}	This indicates the end of the main function.

A SIMPLE C PROGRAM:

Below C program is a very simple and basic program in C programming language. This C program displays "Hello World!" in the output window. And, all syntax and commands in C programming are case sensitive. Also, each statement should be ended with semicolon (;) which is a statement terminator.

A) Write a program to print Hello world

```
    #include <stdio.h>
intmain()
{
  /* Our first simple C basic program */
  printf("Hello World! ");
getch();
```

return0;
}

OUTPUT:

Hello World!

CREATION, COMPILATION AND EXECUTION OF A C PROGRAM:

Prerequisite:

- If you want to create, compile and execute C programs by your own, you have to install C compiler in your machine. Then, you can start to execute your own C programs in your machine.
- You can refer below link for how to install C compiler and compile and execute C programs in your machine.
- Once C compiler is installed in your machine, you can create, compile and execute C programs as shown in below link.
- If you don't want to install C/C++ compilers in your machine, you can refer online compilers which will compile and execute C/C++ and many other programming languages online and display outputs on the screen. Please search for online C/C++ compilers in Google for more details.

BASIC STRUCTURE OF A C PROGRAM:

Structure of C program is defined by set of rules called protocol, to be followed by programmer while writing C program. All C programs are having sections/parts which are mentioned below.

1. Documentation section
2. Link Section
3. Definition Section
4. Global declaration section
5. Function prototype declaration section
6. Main function
7. User defined function definition section

B) Write a program to program Sum and Difference of two numbers

Addition of two numbers in C: This C language program performs the basic arithmetic operation of addition of two numbers and then prints their sum on the screen. For example, if a user will input two numbers as 5, 6 then 11 (5 + 6) will be printed on the screen.

```
/*program for addition of 2 numbers */
#include<stdio.h>
int main()
{
  int a, b, c;
printf("Enter two numbers to add\n");
scanf("%d%d",&a,&b);
```

```
  c = a + b;
printf("Sum of entered numbers = %d\n",c);
  return 0;
}
```

For example, if a user will input two numbers as 10, 4 then 6 (10 + 4) will be printed on the screen

```
/*program for subtraction of 2 numbers */
#include<stdio.h>
int main()
{
  int a, b, c;
printf("Enter two numbers to add\n");
scanf("%d%d",&a,&b);
  c = a - b;
printf("Sum of entered numbers = %d\n",c);
  return 0;
}
```

C)Write A program to print area of circle and Square

Formula : Π * r * r

```
/* program to calculate area of circle*/
#include<stdio.h>
int main() {
  float radius, area;
printf("\nEnter the radius of Circle : ");
scanf("%d", &radius);
  area = 3.14 * radius * radius;
printf("\nArea of Circle : %f", area);
  return (0);
}
/* program to calculate area of square*/
```

Formula : side * side

```
#include<stdio.h>
int main() {
  int side, area;
printf("\nEnter the Length of Side : ");
scanf("%d", &side);
  area = side * side;
printf("\nArea of Square : %d", area);
  return (0);
}
```

ASSIGNMENT NO.2

Aim: Study and Implement program using If, If-else and nested If statement.

A) Implement program To Find greatest number among 3 numbers Using If-else statement

B) Implement a program using nested IF

C – Decision Control statement

- In decision control statements (if-else and nested if), group of statements are executed when condition is true. If condition is false, then else part statements are executed.
- There are 3 types of decision making control statements in C language. They are,

1. if statements
2. if else statements
3. nested if statements

"IF", "ELSE" AND "NESTED IF" DECISION CONTROL STATEMENTS IN C:

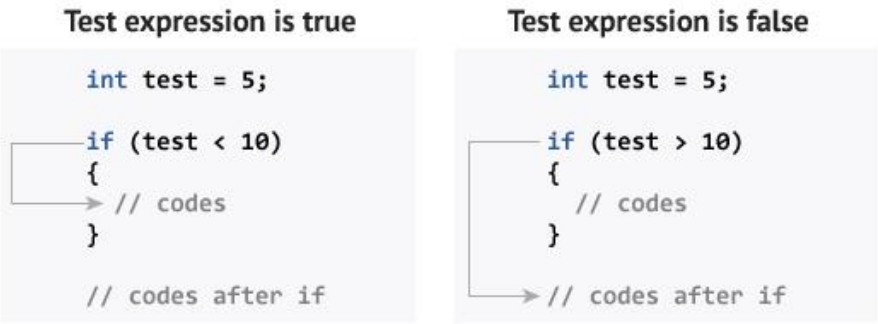
if statement

Syntax of If statement

```
if (testExpression)
{
  // statements
}
```

The if statement evaluates the test expression inside parenthesis.If test expression is evaluated to true, statements inside the body of if is executed.If test expression is evaluated to false, statements inside the body of if is skipped.

How if statement works?



Flowchart of if Statement

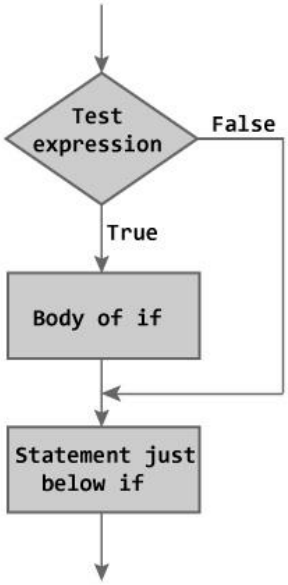


Figure: Flowchart of if Statement

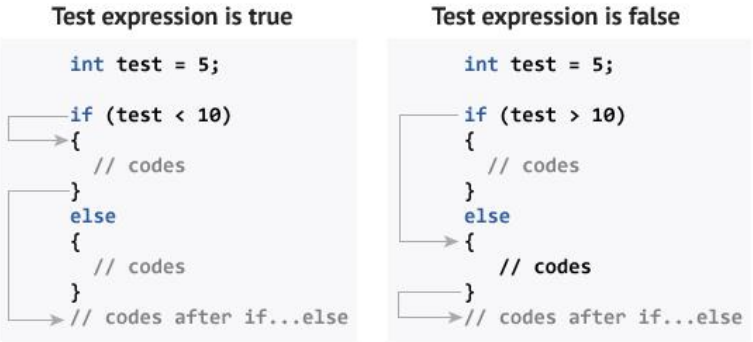
```
// Program to print positive number entered by the user
// If user enters negative number, it is skipped
#include <stdio.h>
int main()
{
   int number;
printf("Enter an integer: ");
scanf("%d",& number);
   // checks if the number is positive
   if ( number> 0)
   {
Printf("You entered a positive integer: %d", number);
   }
Printf( "This statement is always executed.");
   return 0;
}
```

A) Implement program To Find greatest number among 3 numbers Using If-else statement

if...else

The if else executes the codes inside the body of if statement if the test expression is true and skips the codes inside the body of else. If the test expression is false, it executes the codes inside the body of else statement and skips the codes inside the body of if.

How if...else statement works?



Flowchart of if...else

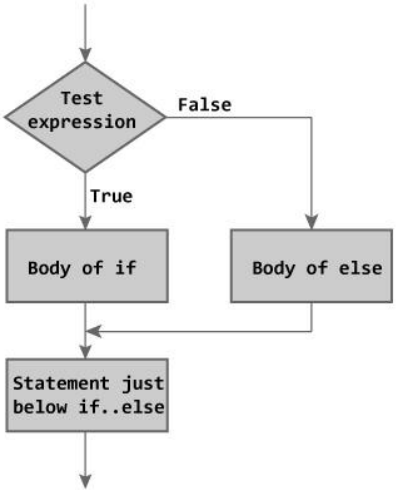


Figure: Flowchart of if...else Statement

```
#include <stdio.h>
int main()
{
   double n1, n2, n3;
printf("Enter three numbers: ");
scanf("%lf %lf %lf", &n1, &n2, &n3);
if( n1>=n2 && n1>=n3)
printf("%.2lf is the largest number.", n1);
   else if (n2>=n1 && n2>=n3)
printf("%.2lf is the largest number.", n2);
   else
printf("%.2lf is the largest number.", n3);
   return 0;
```

}

B) Implement a program using nested IF

Nested if...else

The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.
The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.

Syntax of Nested if...else

```
if (testExpression1)
{
  // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
  // statements to be executed if testExpression1 is false and testExpression2 is true
}
else if (testExpression 3)
{
  // statements to be executed if testExpression1 and testExpression2 is false and testExpression3
is true
}
.
.
else
{
  // statements to be executed if all test expressions are false
}
```

```
/* Example for Nested If in C Programming */
#include <stdio.h>
 int main()
{
 int age;
  printf("Please Enter Your Age Here:\n");
scanf("%d",&age);
  if ( age< 18 )
 {
  printf("You are Minor.\n");
  printf("Not Eligible to Work");
 }
 else
 {
```

```
 if (age >= 18 && age <= 60 )
  {
   printf("You are Eligible to Work \n");
   printf("Please fill in your details and apply\n");
  }
 else
  {
   printf("You are too old to work as per the Government rules\n");
   printf("Please Collect your pension! \n");
  }
}

 return 0;
}
```

ASSIGNMENT NO.3

Aim: Study and implement program using switch case statement.

A) Implement menu driven Program to find out the entered day by user.
B) Implement menu driven program to find weather the entered character is vowel or constant.

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

Syntax

The syntax for a **switch** statement in C programming language is as follows −

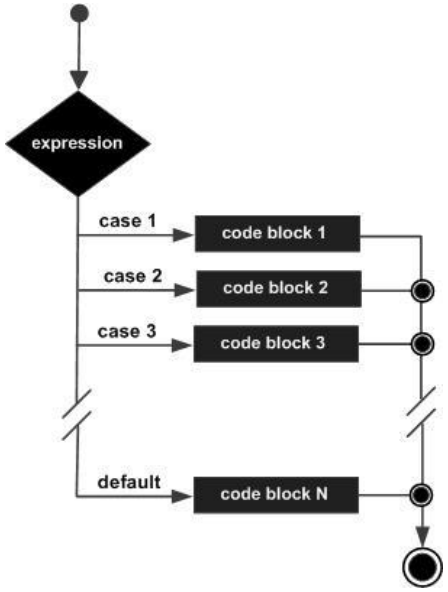
```
switch(expression){
case constant-expression  :
   statement(s);
break;/* optional */
case constant-expression  :
   statement(s);
break;/* optional */
/* you can have any number of case statements */
default:/* Optional */
  statement(s);
}
```

The following rules apply to a **switch** statement −

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.

- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Flow Diagram



A) Implement menu driven Program to find out the entered day by user.

Step by step descriptive logic to print day name of week.

1. Input day number from user. Store it in some variable say week.
2. Switch the value of week i.e. use switch (week) and match with cases.
3. There can be 7 possible values (choices) of week i.e. 1 to 7. Therefore write 7 case inside switch. In addition, add default case as an else block.
4. For case 1: print "MONDAY", for case 2: print "TUESDAY" and so on. Print "SUNDAY" for case 7:.
5. If any case does not matches then, for default: case print "Invalid week number".

```
/*C program to read weekday number and print weekday name using switch.*/
#include <stdio.h>
intmain()
{
intwDay;
printf("Enter weekday number (0-6): ");
```

```
scanf("%d",&wDay);
switch(wDay)
   {
case 0:
printf("Sunday");
break;
case 1:
printf("Monday");
break;
case 2:
printf("Tuesday");
break;
case 3:
printf("Wednesday");
break;
case 4:
printf("Thursday");
break;
case 5:
printf("Friday");
break;
case 6:
printf("Saturday");
break;
      default:
printf("Invalid weekday number.");
   }
printf("\n");
return0;
}
```

B) Implement menu driven program to find weather the entered character is vowel or constant.

English alphabets 'a', 'e', 'i', 'o', 'u' both lowercase and uppercase are known as vowels. Alphabets other than vowels are known as consonants.

Step by step descriptive logic to check vowel or consonant.

1. Input an alphabet from user. Store it in some variable say ch.
2. Switch the value of ch.
3. For ch, there are 10 possibilities for vowel we need to check i.e. a, e, i, o, u, A, E, I, O and U.
4. Write all 10 possible cases for vowels and print "Vowel" for each case.
5. If alphabet is not vowel then add a default case and print "Consonant".

```
/**
 * C program to check vowel or consonant using switch case
 */
```

```
#include<stdio.h>
intmain()
{
charch;
/* Input an alphabet from user */
printf("Enter any alphabet: ");
scanf("%c",&ch);
/* Switch value of ch */
switch(ch)
{
case'a':
printf("Vowel");
break;
case'e':
printf("Vowel");
break;
case'i':
printf("Vowel");
break;
case'o':
printf("Vowel");
break;
case'u':
printf("Vowel");
break;
case'A':
printf("Vowel");
break;
case'E':
printf("Vowel");
break;
case'I':
printf("Vowel");
break;
case'O':
printf("Vowel");
break;
case'U':
printf("Vowel");
break;
default:
printf("Consonant");
}

return0;
}
```

ASSIGNMENT NO.4

Aim: Study and implement program using loop statement (do-while, while, for)

A) Implement program to find sum of positive integers using for loop

B) Implement program to display even numbers from 1 to 50 using while loop

C) Implement program to display odd numbers from 1 to 50 using do-while loop

Loops are used in programming to repeat a specific block until some end condition ismet. There are three loops in C programming:

1. for loop
2. while loop
3. do...while loop

A) Implement program to find sum of positive integers using for loop

for Loop

The syntax of for loop is:

```
for (initialization Statement; test Expression; update Statement)
{
    // codes
}
```

How for loop works?

The initialization statement is executed only once.

Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and the update expression is updated. This process repeats until the test expression is false. The for loop is commonly used when the number of iterations is known. To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out relational and logical operators.

for loop Flowchart

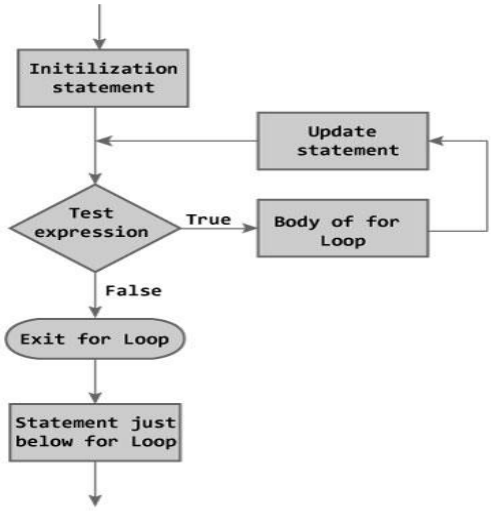


Figure: Flowchart of for Loop

```
// Program to calculate the sum of first n natural numbers
// Positive integers 1,2,3...n are known as natural numbers
#include <stdio.h>
int main()
{
   int num, count, sum = 0;
printf("Enter a positive integer: ");
scanf("%d", &num);
   // for loop terminates when n is less than count
for(count = 1; count <= num; ++count)
   {
      sum += count;
   }
printf("Sum = %d", sum);
  return 0;
}
```

B) Implement program to display even numbers from 1 to 50 using while loop

while loop

The syntax of a while loop is:

```
while (testExpression)
{
   //codes
}
```

where, testExpression checks the condition is true or false before each loop.

How while loop works?

The while loop evaluates the test expression.

If the test expression is true (nonzero), codes inside the body of while loop are exectued. The test expression is evaluated again. The process goes on until the test expression is false.

When the test expression is false, the while loop is terminated.

Flowchart of while loop

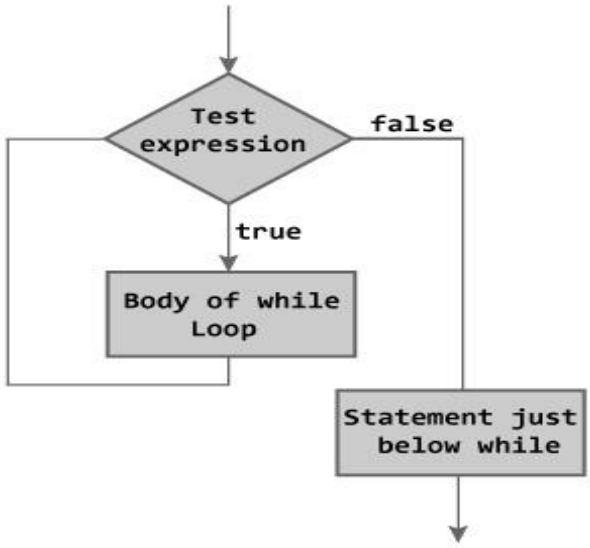


Figure: Flowchart of while Loop

```
**
 * C program to print all even numbers from 1 to n
 */
#include<stdio.h>
intmain()
{
inti,n;
// Input upper limit of even number from user
printf("Print all even numbers till: ");
scanf("%d",&n);
printf("All even numbers from 1 to %d are: \n", n);
i=1;
while(i<=n)
{
/* Check even condition before printing */
if(i%2==0)
{
printf("%d\n",i);
}
i++;
}
return0;
}
```

C) Implement program to display odd numbers from 1 to 50 using do-while loop

do...while loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once.

do...while loop Syntax

```
do
{
  // codes
}
while (testExpression);
```

How do...while loop works?

The code block (loop body) inside the braces is executed once.Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false).When the test expression is false (nonzero), the do...while loop is terminated.

Flowchart of do..while loop

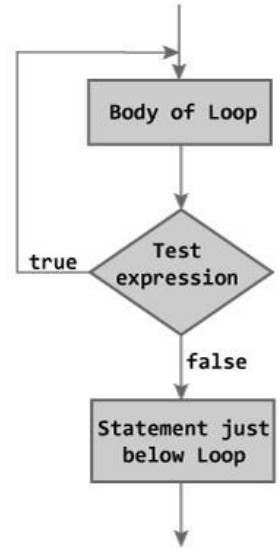


Figure: Flowchart of do...while Loop

```
/**
 * C program to print all Odd numbers from 1 to n
 */
#include<stdio.h>
intmain()
{
inti,n;
```

```
/* Input upper limit from user */
printf("Print odd numbers till: ");
scanf("%d",&n);
printf("All odd numbers from 1 to %d are: \n", n);

/* Start loop from 1 and increment it by 1 */
for(i=1;i<=n;i++)
{
/* If 'i' is odd then print it */
if(i%2!=0)
{
printf("%d\n",i);
}
}
return0;
}
```

ASSIGNMENT NO.5

Aim: Study and implement program using function.

A) Implement menu driven program to find Arithmetic operation using functions
B) Implement program to calculate factorial of number using function recursion

A function is a block of code that performs a specific task.
Suppose, a program related to graphics needs to create a circle and color it depending upon the radius and color from the user. You can create two functions to solve this problem:

- create a circle function
- color function

Dividing complex problem into small components makes program easy to understand and use.

Types of functions in C programming

Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming
There are two types of functions in C programming:

- Standard library functions
- User defined functions

Standard library functions
The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.
These functions are defined in the header file. When you include the header file, these functions are available for use. For example: The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in "stdio.h" header file. There are other numerous library functions defined under "stdio.h", such as scanf(), fprintf(), getchar() etc. Once you include "stdio.h" in your program, all these functions are available for use.

User-defined functions
As mentioned earlier, C allow programmers to define functions. Such functions created by the user are called user-defined functions.
Depending upon the complexity and requirement of the program, you can create as many user-defined functions as you want.
A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function −

- **Return Type** − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The

parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

- **Function Body** − The function body contains a collection of statements that define what the function does.

How user-defined function works?

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

int main()
{
   ... .. ...
   ... .. ...

functionName();

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

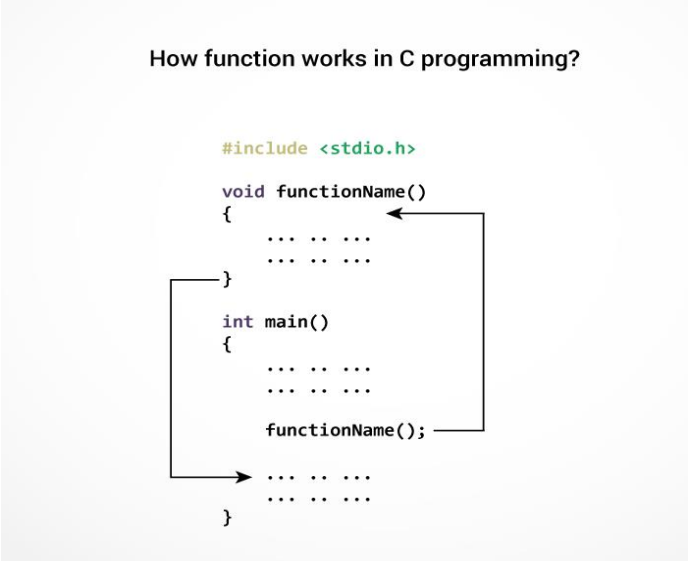
The execution of a C program begins from the main() function.

When the compiler encounters functionName(); inside the main function, control of the program jumps to

void functionName()

And, the compiler starts executing the codes inside the user-defined function.

The control of the program jumps to statement next to functionName(); once all the codes inside the function definition are executed.



Remember, function name is an identifier and should be unique.
This is just an overview on user-defined function. Visit these pages to learn more on:

- User-defined Function in C programming
- Types of user-defined Functions

Advantages of user-defined function

1. The program will be easier to understand, maintain and debug.
2. Reusable codes that can be used in other programs
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

A) Implement menu driven program to find Arithmetic operation using functions

Problem

Write a menu driven program using switch statement that has the following options:
Add numbers
Subtract numbers
Multiply numbers
Divide numbers
Exit
The prototype of the function for add, subtract, multiply and divide is as follows:

```
int add(int, int);
int sub(int, int);
int mul(int, int);
int div(int, int);
```

Solution - Code

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int add (int, int);
int sub (int, int);
int mul (int, int);
int divi (int, int);
int main()
{
   int i,x,y,c;
   printf("Enter two numbers : ");
   scanf ("%d %d",&x,&y);
   printf("1. addition\n");
   printf("2. subtraction\n");
   printf("3. multiplication\n");
   printf("4. division\n");
```

```
    printf("5. exit\n\n");
    printf("Which action you want to perform : ");
    scanf ("%d",&i);
    switch(i)
    {
        case 1:
            c=add(x , y);
            printf("Answer = %d\n",c);
            system("pause");
            break;
        case 2:
            c=sub(x , y);
            printf("Answer = %d\n",c);
            system("pause");
            break;
        case 3:
            c=mul(x , y);
            printf("Answer = %d\n",c);
            system("pause");
            break;
        case 4:
            c=divi(x,y);
            printf("Answer = %d\n",c);
            system("pause");
            break;
        case 5 :
            exit(0);
            break;
        default :
            printf("Enter valid number\n");
            system("pause");
    }
}
int add(int x, int y)
{
    int c;
    c = (x+y);
    return c ;
}
int sub(int x, int y)
```

```
{
   int c;
   c = (x-y);
   printf("Answer = %d\n",c);
}
int mul(int x, int y)
{
   int c;
   c = (x*y);
   printf("Answer = %d\n",c);
}
int divi(int x, int y)
{
   int c;
   c = (x/y);
   printf("Answer = %d\n",c);
}
```

B) Implement program to calculate factorial of number using function recursion

The factorial of a positive number n is given by:
factorial of n (n!) = 1*2*3*4....n
The factorial of a negative number doesn't exist. And the factorial of 0 is 1.

```
/*
 * C Program to find factorial of a given number using recursion
 */
#include <stdio.h>
 int factorial(int);
 int main()
{
   int num;
   int result;
printf("Enter a number to find it's Factorial: ");
scanf("%d", &num);
   if (num < 0)
   {
printf("Factorial of negative number not possible\n");
   }
   else
   {
```

```
        result = factorial(num);
printf("The Factorial of %d is %d.\n", num, result);
    }
    return 0;
}
int factorial(int num)
{
    if (num == 0 || num == 1)
    {
        return 1;
    }
    else
    {
return(num * factorial(num - 1));
    }
}
```

ASSIGNMENT NO.6

Aim: Study and implement program using array

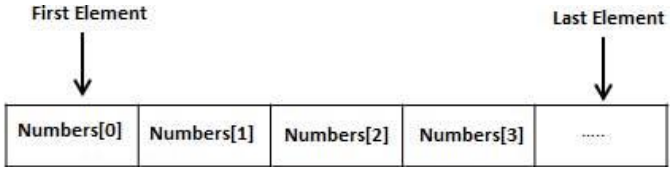
A) Implement program to find average of marks using array
B) Implement program to calculate addition of 2 matrices using 2D- array

Arrays

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows −

type arrayName [arraySize];

This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and type can be any valid C data type. For example, to declare a 10-element array called balance of type double, use this statement −

double balance[10];

Here balance is a variable array which is sufficient to hold up to 10 double numbers.

Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows −

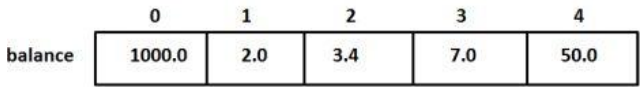
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array −

balance[4] = 50.0;
The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above −



Accessing Array Elements
An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

double salary = balance[9];
The above statement will take the 10th element from the array and assign the value to salary variable. The following example Shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays –

Few key notes:
Arrays have 0 as the first index not 1. In this example, mark[0]
If the size of an array is n, to access the last element, (n-1) index is used. In this example, mark[4]
Suppose the starting address of mark[0] is 2120d. Then, the next address, a[1], will be 2124d, address of a[2] will be 2128d and so on. It's because the size of a float is 4 bytes.
How to initialize an array in C programming?
It's possible to initialize an array during declaration. For example,

int mark[5] = {19, 10, 8, 17, 9};
Another method to initialize array during declaration:
int mark[] = {19, 10, 8, 17, 9};

Initialize an array in C programming
Here,

mark[0] is equal to 19
mark[1] is equal to 10
mark[2] is equal to 8
mark[3] is equal to 17
mark[4] is equal to 9

A) Implement program to find average of marks using array

```
// Program to find the average of n (n < 10) numbers using arrays
#include <stdio.h>
int main()
{
    int marks[10], i, n, sum = 0, average;
printf("Enter n: ");
scanf("%d", &n);
for(i=0; i<n; ++i)
    {
printf("Enter number%d: ",i+1);
scanf("%d", &marks[i]);
        sum += marks[i];
    }
    average = sum/n;
printf("Average = %d", average);
    return 0;
}
```

Multi-dimensional Arrays in C

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration −

```
type name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional integer array −

```
int threedim[5][10][4];
```

Two-dimensional Arrays

The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows −

```
type arrayName [ x ][ y ];
```

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows –

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array a is identified by an element name of the form a[i][j], where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */

};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example −

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example −

```
int val = a[2][3];
```

The above statement will take the 4th element from the 3rd row of the array. You can verify it in the above figure. Let us check the following program where we have used a nested loop to handle a two-dimensional array –

B) Implement program to calculate addition of 2 matrices using 2D- array

```
/*Program to Add Two Matrices*/
#include <stdio.h>
int main(){
   int r, c, a[100][100], b[100][100], sum[100][100], i, j;
printf("Enter number of rows (between 1 and 100): ");
scanf("%d", &r);
printf("Enter number of columns (between 1 and 100): ");
scanf("%d", &c);
printf("\nEnter elements of 1st matrix:\n");
for(i=0; i<r; ++i)
for(j=0; j<c; ++j)
      {
printf("Enter element a%d%d: ",i+1,j+1);
scanf("%d",&a[i][j]);
      }
printf("Enter elements of 2nd matrix:\n");
for(i=0; i<r; ++i)
for(j=0; j<c; ++j)
      {
printf("Enter element a%d%d: ",i+1, j+1);
scanf("%d", &b[i][j]);
      }
   for(i=0;i<r;++i)
      for(j=0;j<c;++j)
      {
         sum[i][j]=a[i][j]+b[i][j];
      }
printf("\nSum of two matrix is: \n\n");
   for(i=0;i<r;++i)
      for(j=0;j<c;++j)
      {
printf("%d   ",sum[i][j]);
         if(j==c-1)
         {
printf("\n\n");
```

```
        }
    }
    return 0;
}
```

ASSIGNMENT NO.7

Aim: Study and implement program using structure

A) Implement program to print record of students using Structure

Structure

Structure is a group of variables of different data types represented by a single name. Lets take an example to understand the need of a structure in C programming.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store these information separately.

However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2

You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.

A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well.

This collection of all related information under a single name Person is a structure.

Structure Definition in C

Keyword struct is used for creating a structure.

Syntax of structure

```
struct structure_name
{
data_type member1;
data_type member2;
   .
   .
data_typememeber;
};
```

We can create the structure for a person as mentioned above as:

```
struct person
{
   char name[50];
   int citNo;
   float salary;
};
```

This declaration above creates the derived data type struct person.

Structure variable declaration

When a structure is defined, it creates a user-defined type but, no storage or memory is allocated.

For the above structure of a person, variable can be declared as:

```
struct person
```

```
{
   char name[50];
   int citNo;
   float salary;
};

int main()
{
   struct person person1, person2, person3[20];
   return 0;
}
```

Another way of creating a structure variable is:

```
struct person
{
   char name[50];
   int citNo;
   float salary;
} person1, person2, person3[20];
```

In both cases, two variables person1, person2 and an array person3 having 20 elements of type **struct person** are created.

Accessing members of a structure

There are two types of operators used for accessing members of a structure.

1. Member operator(.)
2. Structure pointer operator(->) (is discussed in structure and pointers tutorial)

Any member of a structure can be accessed as:

```
structure_variable_name.member_name
```

Suppose, we want to access salary for variable person2. Then, it can be accessed as:

```
person2.salary
```

A) Implement program to print record of students using Structure

```
#include<stdio.h>
/* Created a structure here. The name of the structure is
 * StudentData.
 */
structStudentData{
char*stu_name;
intstu_id;
intstu_age;
};
intmain()
{
/* student is the variable of structure StudentData*/
structStudentDatastudent;
```

```
/*Assigning the values of each struct member here*/
student.stu_name="Steve";
student.stu_id=1234;
student.stu_age=30;

/* Displaying the values of struct members */
printf("Student Name is: %s",student.stu_name);
printf("\nStudent Id is: %d",student.stu_id);
printf("\nStudent Age is: %d",student.stu_age);
return0;
}
```

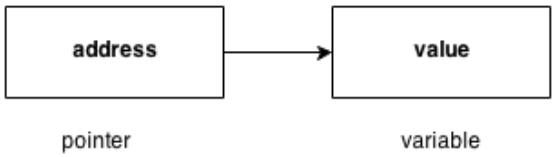
ASSIGNMENT NO.8

Aim: Study and implement program using pointer

A) Implement program to swap 2 numbers using pointers

Pointers

Pointers in C are easy and fun to learn. Some C programming tasks are performed more easily with pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect C programmer. Let's start learning them in simple and easy steps.
As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory. Consider the following example, which prints the address of the variables defined –



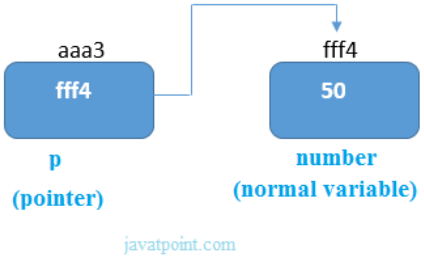
Declaring a pointer

The pointer in c language can be declared using * (asterisk symbol).

int *a;//pointer to int
char *c;//pointer to char

Pointer example

An example of using pointers printing the address and value is given below.



Demo

```
#include <stdio.h>
int main () {
int  var1;
  char var2[10];
```

```
printf("Address of var1 variable: %x\n", &var1  );
printf("Address of var2 variable: %x\n", &var2  );
  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

What are Pointers?

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

```
type *var-name;
```

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations −

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

How to Use Pointers?

There are a few important operations, which we will do with the help of pointers very frequently. **(a)** We define a pointer variable, **(b)** assign the address of a variable to a pointer and **(c)** finally access the value at the address available in the pointer variable. This is done by using unary operator ***** that returns the value of the variable located at the address specified by its operand. The following example makes use of these operations −

Demo

```
#include<stdio.h>
int main (){
intvar=20;/* actual variable declaration */
int*ip;/* pointer variable declaration */
```

```
ip=&var;/* store address of var in pointer variable*/
printf("Address of var variable: %x\n",&var);
/* address stored in pointer variable */
printf("Address stored in ip variable: %x\n",ip);
/* access the value using the pointer */
printf("Value of *ip variable: %d\n",*ip);
return0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

KEY POINTS TO REMEMBER ABOUT POINTERS IN C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- & symbol is used to get the address of the variable.
- * symbol is used to get the value of the variable that the pointer is pointing to.
- If a pointer in C is assigned to NULL, it means it is pointing to nothing.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- The size of any pointer is 2 byte (for 16 bit compiler).
-

Advantage of pointer

1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees etc. and used with arrays, structures and functions.

2) We can return multiple values from function using pointer.

3) It makes you able to access any memory location in the computer's memory.

Usage of pointer

There are many usage of pointers in c language.

A) Implement program to swap 2 numbers using pointers

```
/*C program to swap two numbers using pointers.*/
#include <stdio.h>
 // function : swap two numbers using pointers
void swap(int *a,int *b)
{
```

```
    int t;
     t   = *a;
    *a   = *b;
    *b   =  t;
}
 int main()
{
    int num1,num2;
printf("Enter value of num1: ");
scanf("%d",&num1);
printf("Enter value of num2: ");
scanf("%d",&num2);
        //print values before swapping
printf("Before Swapping: num1=%d, num2=%d\n",num1,num2);
        //call function by passing addresses of num1 and num2
    swap(&num1,&num2);
        //print values after swapping
printf("After  Swapping: num1=%d, num2=%d\n",num1,num2);
        return 0;
}
```