

PROGRAMMING IN C AND DATA STRUCTURES

Code: 14PCD13/23
Hrs/Week: 04
Total Hrs: 52

IA Marks: 25
Exam Hrs: 03
Exam Marks:100

Objectives:

The objectives of this course is to make students to learn basic principles of Problem solving, implementing through C programming language and to design & develop programming skills.

MODULE I

INTRODUCTION TO C LANGUAGE: Pseudocode solution to problem, Basic concepts of a C program, Declaration, Assignment & Print statement, Types of operators and expressions, Programming examples and exercise.

Text 1: Chapter 2

Text 2: 1.1, 1.2, 1.3

10 Hours

MODULE II

BRANCHING AND LOOPING: Two way selection (if, if-else, nested if-else, cascaded if-else), switch statement, ternary operator? Go to, Loops (For, do-while, while) in C, break and continue, programming examples and exercises.

Text 1: Chapter 3

Text 2: 4.4

10 Hours

MODULE II

BRANCHING AND LOOPING: Two way selection (if, if-else, nested if-else, cascaded if-else), switch statement, ternary operator? Go to, Loops (For, do-while, while) in C, break and continue, programming examples and exercises.

Text 1: Chapter 3

Text 2: 4.4

10 Hours

MODULE III

ARRAYS, STRINGS AND FUNCTIONS:

ARRAYS AND STRINGS: Using an array, Using arrays with Functions, Multi-Dimensional arrays. String: Declaring, Initializing, Printing and reading strings, strings manipulation functions, strings input and output functions, arrays of strings, programming examples and Exercises.

Text 1: 5.7

Text 2: 7.3, 7.4 Chapter 9

FUNCTIONS: Functions in C, Argument Passing – call by value, Functions and program structure, location of functions, void and parameter less Functions, Recursion, programming examples and exercises.

Text 1: 1.7, 1.8, Chapter 4

Text 2: 5.1 to 5.4

10 Hours

MODULE IV

STRUCTURES AND FILE MANAGEMENT: Basic of structures, structures and Functions, Arrays of structures, structure Data types, type definition, Defining, opening and closing of files, Input and output operations, programming examples and exercises.

Text 1: 6.1 to 6.3

Text 2: 10.1 to 10.4, Chapter 11

10 Hours

MODULE V

POINTERS AND PREPROCESSORS: Pointers and address, pointers and functions arguments, pointers and arrays, address arithmetic, character pointer and functions, pointers to pointer ,Initialization of pointers arrays, Dynamic allocations methods, Introduction to Preprocessors, Compiler control Directives, programming examples and exercises.

Text 1: 5.1 to 5.6, 5.8

Text 2: 12.2, 12.3, 13.1 to 13.7

Introduction to Data Structures: Primitive and non primitive data types, Definition and applications of Stacks, Queues, Linked Lists and Trees.

Text 2: 14.1, 14.2, 14.11, 14.12, 14.13, 14.15, 14.16, 14.17, 15.1

08 Hours + 04 Hours

TEXT BOOK:

1. Brain W. Kernighan and Dennis M. Richie: The C programming Language, 2nd Edition, PHI, 2012.
2. Jacqueline Jones & Keith Harrow: Problem Solving with C, 1st Edition, Pearson 2011.

TABLE OF CONTENTS

MODULE I: INTRODUCTION TO C LANGUAGE	1-19
MODULE II: BRANCHING & LOOPING	20-34
MODULE III: ARRAYS & STRINGS	35-48
FUNCTIONS	49-57
MODULE IV: STRUCTURES & FILE MANAGEMENT	58-64
MODULE V: POINTERS & PREPROCESSORS	65-76
INTRODUCTION TO DATA STRUCTURES	77-89

WORKED EXAMPLES

BASIC C PROGRAMS	90-93
C PROGRAMS BASED ON BRANCHING	94-97
C PROGRAMS BASED ON LOOPING	98-106
C PROGRAMS BASED ON ARRAYS	107-116
C PROGRAMS BASED ON FUNCTIONS	117-120
C PROGRAMS BASED ON STRUCTURES & FILE MGMT.	121-124



MODULE I: INTRODUCTION TO C LANGUAGE

INTRODUCTION

- A computer is an electronic-device capable of manipulating numbers and symbols under the control of a program.
- A program is a sequence of instructions written using a computer programming language to perform a specified task.

A PSEUDOCODE SOLUTION TO PROBLEM

Algorithm

- An algorithm is a step by step procedure to solve a given problem in finite number of steps by
 - accepting a set of inputs and
 - producing the desired output for the given problem
- Example: Write an algorithm to add 2 numbers entered by user.
 - Step 1: Start
 - Step 2: Declare variables num1, num2 & sum.
 - Step 3: Read values of num1 and num2.
 - Step 4: Add num1 & num2 and assign the result to sum.
 - sum ← num1 + num2
 - Step 5: Display sum
 - Step 6: Stop

Pseudocode

- Pseudocode is a method of describing the algorithm using a combination of
 - natural language (English like words) and
 - programming language
- This is essentially an intermediate-step towards the development of the actual code.
- Although pseudocode is frequently used, there are no set of rules for its exact writing.
- Psuedocode is used in textbooks and scientific publications to describe various algorithms.
- For Example:
 - Problem 1: Input two numbers and display their sum.
 - 1) read num1, num2
 - 2) find sum = num1 + num2
 - 3) display sum
 - Problem 2: Input the marks and display message "passed" or "failed" based on the marks.
 - 1) read marks
 - 2) if student's marks is greater than or equal to 35
 - print "passed"
 - else
 - print "failed"
 - endif



PROGRAMMING IN C AND DATA STRUCTURES

BASIC CONCEPTS OF A C PROGRAM

- The structure of a C program is shown below:

```
preprocessor directives
void main()
{
    declaration section

    statement-1    // Executable section starts
    statement-2
    statement-3
    statement-4    // Executable section ends
}
```

Preprocessor Directives

- The *preprocessor* accepts the source program and prepare the source program for compilation.
- The preprocessor-statements start with symbol #.
- The normal preprocessor used in all programs is include.
- The #include directive instructs the preprocessor to include the specified file-contents in the beginning of the program.
- For ex:

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

main()

- Every C program should have a function called as main().
- This the first function to be executed always.
- The statements enclosed within left and right brace is called body of the function. The main() function is divided into 2 parts:

1) Declaration Section

- The variables that are used within the function main() should be declared in the declaration-section only.
- The variables declared inside a function are called local-variables.

Ex:

```
int p, t, r;
```

2) Executable Section

- This contains the instructions given to the computer to perform a specific task.
- The task may be to
 - display a message
 - read data or
 - add 2 numbers etc
- Comments are portions of the code ignored by the compiler. The comments allow the user to make simple notes in the source-code.

```
// this is an example for single line comment
```

```
/* this is an example
```

```
for multiple line comment */
```

Example: Program to display a message on the screen.

```
#include<stdio.h>
void main()
{
    printf("Welcome to C");
}

Output:
Welcome to C
```



PROGRAMMING IN C AND DATA STRUCTURES

HOW TO LEARN C LANGUAGE?

- English is a universal language used to communicate with others.
- In the same way, C is a language used to communicate with computer. In other words, C is used to instruct computer to perform particular task.
- The task can be
 - simple task like adding 2 numbers or
 - complex task like building a railway reservation system
- Before you play the game, you should learn rules of the game. So that you can play better and win easily. In the same way, to write C programs, you should learn rules of C language.

STEPS TO LEARN C LANGUAGE

Step 1: Before speaking any language, you should first learn alphabets. In the same way, to learn C language, you should first learn alphabets in C.

Step 2: Then, you should learn how to group alphabets in particular sequence to form a meaningful word. In the same way, in C language, you should learn tokens (i.e. words).

Step 3: Then, you should learn how to group the words in particular sequence to form a meaningful sentence. In the same way, in C language, you should learn instruction (i.e. sentence).

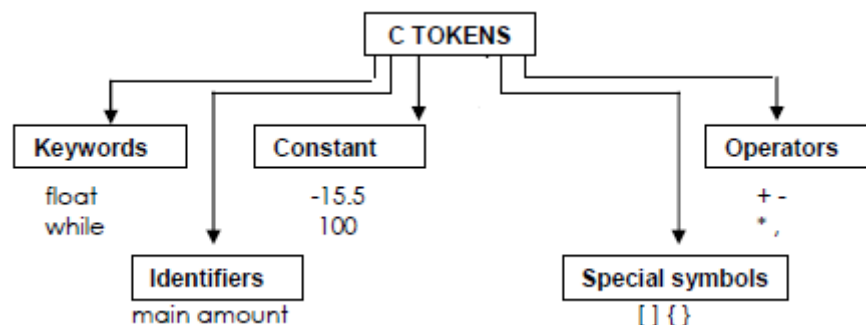
Step 4: Then, you should learn how to group the sentences in particular sequence to form a meaningful paragraph. In the same way, in C language, you should learn program (i.e. paragraph).

CHARACTER SET

- Character-set refers to the set of alphabets, letters and some special characters that are valid in C language.
- For example, the characters in C are:
 - Letters A-X, a-z, both upper and lower
 - Digits 0-9
 - Symbols such as + - * / %
 - White spaces

TOKENS

- A token is a smallest element of a C program.
- One or more characters are grouped in sequence to form meaningful words. These meaningful words are called *tokens*.
- The tokens are broadly classified as follows
 - Keywords ex: if, for, while
 - Identifiers ex: sum, length
 - Constants ex: 10, 10.5, 'a', "sri"
 - Operators ex: + - * /
 - Special symbols ex: [], (), {}





PROGRAMMING IN C AND DATA STRUCTURES

KEYWORDS

- Keywords are tokens which are used for their intended purpose only.
- Each keyword has fixed meaning and that cannot be changed by user. Hence, they are also called reserved-words.

Rules for using keywords

- Keywords cannot be used as a variable or function.
- All keywords should be written in lower letters.
- Some keywords are as listed below

break	case	char	const	continue	default	do
double	else	float	for	if	int	long
register	return	short	signed	sizeof	struct	switch
typedef	unsigned	void	while			

IDENTIFIER

- As the name indicates, identifier is used to identify various entities of program such as variables, constants, functions etc.
- In other words, an identifier is a word consisting of sequence of
 - Letters
 - Digits or
 - "_"(underscore)
- For ex:
area, length, breadth



PROGRAMMING IN C AND DATA STRUCTURES

CONSTANTS

- A constant is an identifier whose value remains fixed throughout the execution of the program.
- The constants cannot be modified in the program.
- For example:

1, 3.14512, 'z', "vtunotesbysri"

- Different types of constants are:

1) Integer Constant

- An integer is a whole number without any fraction part.
- There are 3 types of integer constants:
 - i) Decimal constants (0 1 2 3 4 5 6 7 8 9)
For ex: 0, -9, 22
 - ii) Octal constants (0 1 2 3 4 5 6 7)
For ex: 021, 077, 033
 - iii) Hexadecimal constants (0 1 2 3 4 5 6 7 8 9 A B C D E F)
For ex: 0x7f, 0x2a, 0x521

2) Floating Point Constant

- The floating point constant is a real number.
- The floating point constants can be represented using 2 forms:
 - i) **Fractional Form**
 - A floating point number represented using fractional form has an integer part followed by a dot and a fractional part.
 - For ex:
0.5, -0.99
 - ii) **Scientific Notation (Exponent Form)**
 - The floating point number represented using scientific notation has three parts namely: mantissa, E and exponent.
 - For ex:
9.86E3 imply 9.86×10^3

3) Character Constant

- A symbol enclosed within a pair of single quotes(') is called a character constant.
- Each character is associated with a unique value called an ASCII (American Standard Code for Information Interchange) code.
- For ex:
'9', 'a', '\n'

4) String Constant

- A sequence of characters enclosed within a pair of double quotes("") is called a string constant.
- The string always ends with NULL (denoted by \0) character.
- For ex:
"9" "a" "sri" "\n"

5) Escape Sequence Characters

- An escape sequence character begins with a backslash and is followed by one character.
- A backslash (\) along with some characters give rise to special print effects by changing (escaping) the meaning of some characters.
- The complete set of escape sequences are:

Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character



PROGRAMMING IN C AND DATA STRUCTURES

BASIC DATA TYPES

- The data type defines the type of data stored in a memory-location.
- C supports 3 classes of data types:
 - 1) Primary data type
For ex: int, float
 - 2) Derived data types
For ex: array
 - 3) User defined data types
For ex: structure
- C supports 5 primary data types:
 - 1) **int**
 - An int is a keyword which is used to define integers.
 - Using int keyword, the programmer can inform the compiler that the data associated with this keyword should be treated as integer.
 - C supports 3 different sizes of integer:
 - short int
 - int
 - long int
 - 2) **float**
 - A float is a keyword which is used to define floating point numbers.
 - 3) **double**
 - A double is a keyword used to define long floating point numbers.
 - 4) **char**
 - A char is a keyword which is used to define single character.
 - 5) **void**
 - void is an empty data type. Since no value is associated with this data type, it does not occupy any space in the memory.
 - This is normally used in functions to indicate that the function does not return any value.

Range of data types

Data type	Bytes	Range of data type
char	1 bytes	-128 to 127
int	2 bytes	-32, 768 to 32,767
float	4 bytes	3.4E-38 to 3.4E38
double	8 bytes	1.7E-308 to 1.7E308

Qualifiers

- Qualifiers alter the meaning of primary data types to yield a new data type.

Size Qualifiers

- Size qualifiers alter the size of primary data type.
- The keywords long and short are 2 size qualifiers.

For example:

```
long int i; //The size of int is 2 bytes but, when long keyword is
            //used, that variable will be of 4 bytes
short int i; //The size of int is 2 bytes but, when short keyword is
            //used, that variable will be of 1 byte
```

Sign Qualifiers

- Whether a variable can hold positive value, negative value or both values is specified by sign qualifiers.
- Keywords signed and unsigned are used for sign qualifiers.

```
unsigned int a; //unsigned variable can hold zero & positive values only
signed int b; //signed variable can hold zero , positive and negative values
```

Constant Qualifiers

- Constant qualifiers can be declared with keyword const.
- An object declared by const cannot be modified.

```
const int p=20; //the value of p cannot be changed in the program.
```




PROGRAMMING IN C AND DATA STRUCTURES

VARIABLE

- A variable is an identifier whose value can be changed during execution of the program.
In other words, a variable is a name given to a memory-location where the data can be stored.
- Using the variable-name, the data can be
 - stored in a memory-location and
 - accessed or manipulated

Rules for defining a variable

- 1) The first character in the variable should be a letter or an underscore
 - 2) The first character can be followed by letters or digits or underscore
 - 3) No extra symbols are allowed (other than letters, digits and underscore)
 - 4) Length of a variable can be up to a maximum of 31 characters
 - 5) Keywords should not be used as variable-names
- Valid variables:
a, principle_amount, sum_of_digits
 - Invalid variables:
 - 3fact //violates rule 1
 - sum= sum-of-digits 62\$ //violates rule 3
 - for int if //violates rule 5

Declaration of Variable

- The declaration tells the compiler
 - what is the name of the variable used
 - what type of data is held by the variable
- The syntax is shown below:
data_type v₁,v₂,v₃;
where v₁,v₂,v₃ are variable-names
data_type can be int, float or char
- For ex:
int a, b, c;
float x, y, z;

Initialization of Variable

- The variables are not initialized when they are declared. Hence, variables normally contain garbage values and hence they have to be initialized with valid data.
- Syntax is shown below:
data_type var_name=data;
where data_type can be int, float or char
var_name is a name of the variable
= is assignment operator
data is the value to be stored in variable
- For ex:
int a=10;
float pi=3.1416;
char c='z';



PROGRAMMING IN C AND DATA STRUCTURES

DATA INPUT/OUTPUT FUNCTIONS

- There are many library functions for input and output in C language.
- For ex:
getch(), putchar(), scanf(), printf()
- For using these functions in a C-program there should be preprocessor statement #include<stdio.h>.

Input Function

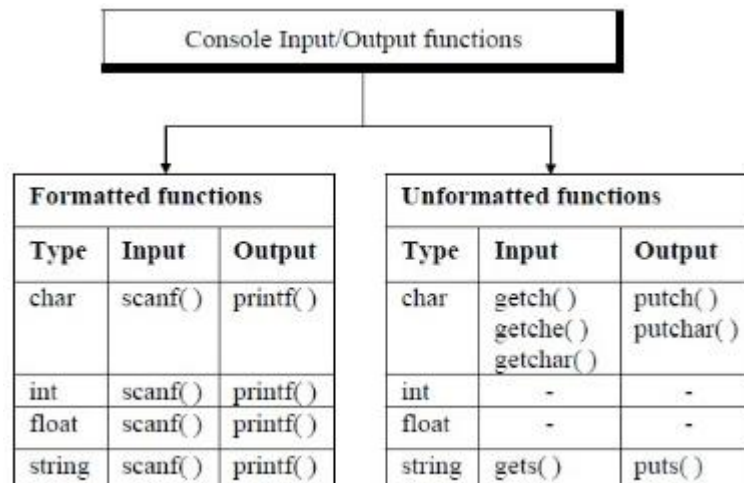
- The input functions are used to read the data from the keyboard and store in memory-location.
- For ex:
scanf(), getchar(), getch(), getche(), gets()

Output Functions

- The output functions are used to receive the data from memory-locations and display on the monitor.
- For ex:
printf(), putchar(), putch(), puts()

Types of I/O Functions

- There are 2 types of I/O Functions as shown below:



UNFORMATTED I/O

getchar() and putchar()

- getchar() is used to
 - read a character from the keyboard and
 - store this character into a memory-location
- You have to press ENTER key after typing a character.
- The syntax is shown below:
char variable_name = getchar();
- For ex:
char z;
z = getchar();
- putchar() is used to display a character stored in the memory-location on the screen.

```
#include<stdio.h>
main()
{
    char x;
    char y='n';
    printf("enter one letter terminated by ENTER key \n");
    x = getchar();
    putchar(y);    // same as printf("%c", z);
}
```

Output:

```
enter one letter terminated by ENTER key
m
n
```



PROGRAMMING IN C AND DATA STRUCTURES

getch() and putch()

- getch() is used to read a character from the keyboard without echo(i.e. typed character will not be visible on the screen). The character thus entered will be stored in the memory location.
- putch() is used to display a character stored in memory-location on the screen.

```
#include<stdio.h>
void main()
{
    int z;
    printf("enter one letter \n ");
    z = getch();
    putch(z);    // same as printf("%c", z)
}
```

Output:

```
enter one letter
m           //m is not visible
m
```

getche()

- getche() is used to read a character from the keyboard with echo(i.e. typed character will be visible on the screen). The character thus entered will be stored in the memory-location.

```
#include<stdio.h>
void main()
{
    int z;
    printf("enter one letter \n");
    z = getche();
    putch( z);    // same as printf("%c", z)
}
```

Output:

```
enter one letter
m           //m is visible
m
```

gets() and puts()

- gets() is used to
 - read a string from the keyboard and
 - store the string in memory-locations
- puts() is used to display a string stored in memory-locations on the screen.

```
#include<stdio.h>
void main()
{
    char string[20];
    printf("enter a string \n");
    gets(string);
    puts(string);    // same as printf("%s", string);
}
```

Output:

```
enter a string
vtunotesbysri
vtunotesbysri
```

Disadvantage of Unformatted I/O

- It is not possible to read/print any other data except characters i.e. it is not possible to read/print integer numbers, floating point numbers etc.



PROGRAMMING IN C AND DATA STRUCTURES

FORMATTED I/O

scanf()

- The scanf function does following tasks:
 - Scan a series of input fields one character at a time
 - Format each field according to a corresponding format-specifier passed in format-string (format means convert).
 - Store the formatted input at an address passed as an argument i.e. address-list
- Syntax is shown below:
n=scanf("format-string", address-list);
where format-string contains one or more format-specifiers
address-list is a list of variables. Each variable name must be preceded by &

- For ex:

```
n=scanf("%d %f %c", &x, &y, &z);
```

Format specifiers Meaning

%d	an int argument in decimal
%ld	a long int argument in decimal
%c	a character
%s	a string
%f	a float or double argument
%e	same as %f, but use exponential notation
%o	an int argument in octal (base 8)
%x	an int argument in hexadecimal (base 16)

printf

The printf function does following tasks:

- Accept a series of arguments
- Apply to each argument a format-specifier contained in the format-string
- Output the formatted data to the screen
- The syntax is shown below:
n=printf("format-string", variable-list);
where format-string contains one or more format-specifiers
variable-list contains names of variables
- For ex:
n=printf("%d %f %c", x, y, z);
- Example: Program to read your age and display the same on the screen.

```
#include<stdio.h>
void main()
{
    int age;
    printf("enter your age: \n");
    scanf("%d", age);
    printf("your age is = %d years ", age);
}
```

Output:

```
enter your age:
21
your age is = 21 years
```



PROGRAMMING IN C AND DATA STRUCTURES

VARIATIONS IN OUTPUT FUNCTION FOR INTEGER AND FLOATS

- Integer and floating-points can be displayed in different formats in C as shown below:

```
#include<stdio.h>
void main()
{
    printf("Case 1:%6d \n",9876);
    // Prints the number right justified within 6 columns
    printf("Case 2:%3d \n",9876);
    // Prints the number to be right justified to 3 columns but, there are 4
    // digits so number is not right justified
    printf("Case 3:%.2f \n",987.6543);
    // Prints the number rounded to two decimal places
    printf("Case 4:%.f \n",987.6543);
    // Prints the number rounded to 0 decimal place, i.e., rounded to integer
    printf("Case 5:%e ",987.6543);
    // Prints the number in exponential notation(scientific notation)
}
```

Output:

```
Case 1:  9876
Case 2:9876
Case 3:987.65
Case 4:988
Case 5:9.876543e+002
```

OPERATOR

- An operator can be any symbol like + - * / that specifies what operation need to be performed on the data.
- For ex:
 - + indicates add operation
 - * indicates multiplication operation

Operand

- An operand can be a constant or a variable.

Expression

- An expression is combination of operands and operator that reduces to a single value.
- For ex:
 - Consider the following expression a+b
 - here a and b are operands
 - while + is an operator

CLASSIFICATION OF OPERATORS

Operator Name	For Example
Arithmetic operators	+ - * / %
Increment/decrement operators	++ --
Assignment operators	=
Relational operators	< > ==
Logical operators	&& ~
Conditional operator	?:
Bitwise operators	& ^
Special operators	[]



PROGRAMMING IN C AND DATA STRUCTURES

ARITHMETIC OPERATORS

- These operators are used to perform arithmetic operations such as addition, subtraction,
- There are 5 arithmetic operators:

Operator	Meaning of Operator
+	addition
-	subtraction
*	multiplication
/	division
%	modulos

- Division symbol (/)
 - divides the first operand by second operand and
 - returns the quotient.
 - Quotient is the result obtained after division operation.
- Modulus symbol (%)
 - divides the first operand by second operand and
 - returns the remainder.
 - Remainder is the result obtained after modulus operation.
- To perform modulus operation, both operands must be integers.
- Program to demonstrate the working of arithmetic operators.

```
#include<stdio.h>
void main()
{
    int a=9,b=4,c;
    c=a+b;
    printf("a+b=%d \n", c);
    c=a-b;
    printf("a-b=%d \n", c);
    c=a*b;
    printf("a*b=%d \n", c);
    c=a/b;
    printf("a/b=%d \n", c);
    c=a%b;
    printf("Remainder when a divided by b=%d ", c);
}
```

Output:

```
a+b=13
a-b=5
a*b=36
a/b=2
Remainder when a divided by b=1
```



PROGRAMMING IN C AND DATA STRUCTURES

INCREMENT OPERATOR

- ++ is an increment operator.
- As the name indicates, increment means increase, i.e. this operator is used to increase the value of a variable by 1.
- For example:
If b=5
then b++ or ++b; // b becomes 6
- The increment operator is classified into 2 categories:
 - 1) Post increment Ex: b++
 - 2) Pre increment Ex: ++b
- As the name indicates, post-increment means first use the value of variable and then increase the value of variable by 1.
- As the name indicates, pre-increment means first increase the value of variable by 1 and then use the updated value of variable.
- For ex:
If x is 10,
then z = x++; sets z to 10
but z = ++x; sets z to 11

Example: Program to illustrate the use of increment operators.

```
#include<stdio.h>
void main()
{
    int x=10,y = 10, z ;
    z= x++;
    printf(" z=%d x= %d\n", z, x);
    z = ++y;
    printf(" z=%d y= %d", z, y);
}
```

Output:

z=10 x=11
z=11 y=11

DECREMENT OPERATOR

- -- is a decrement operator.
- As the name indicates, decrement means decrease, i.e. this operator is used to decrease the value of a variable by 1.
- For example:
If b=5
then b-- or --b; // b becomes 4
- Similar to increment operator, the decrement operator is classified into 2 categories:
 - 1) Post decrement Ex: b--
 - 2) Pre decrement Ex: --b
- For ex:
If x is 10,
then z = x--; sets z to 10,
but z = --x; sets z to 9.

Example: Program to illustrate the use of decrement operators.

```
void main()
{
    int x=10,y = 10, z ;
    z= x--;
    printf(" z=%d x= %d\n", z, x);
    z = --y;
    printf(" z=%d y= %d", z, y);
}
```

Output:

z=10 x=9
z=9 y=9



PROGRAMMING IN C AND DATA STRUCTURES

ASSIGNMENT OPERATOR

- The most common assignment operator is =.
- This operator assigns the value in right side to the left side.
- The syntax is shown below:
variable=expression;
- For ex:
c=5; //5 is assigned to c
b=c; //value of c is assigned to b
5=c; // Error! 5 is a constant.
- The operators such as +=, *= are called shorthand assignment operators.
- For ex,
a=a+10; can be written as a+=10;
- In the same way, we have:

Operator	Example	Same as
-=	a-=b	a=a-b
=	a=b	a=a*b
/=	a/=b	a=a/b
%=	a%=b	a=a%b

RELATIONAL OPERATORS

- Relational operators are used to find the relationship between two operands.
- The output of relational expression is either true(1) or false(0).
- For example
a>b //If a is greater than b, then a>b returns 1 else a>b returns 0.
- The 2 operands may be constants, variables or expressions.
- There are 6 relational operators:

Operator	Meaning of Operator	Example
>	Greater than	5>3 returns true (1)
<	Less than	5<3 returns false (0)
>=	Greater than or equal to	5>=3 returns true (1)
<=	Less than or equal to	5<=3 return false (0)
==	Equal to	5==3 returns false (0)
!=	Not equal to	5!=3 returns true(1)

- For ex:

Condition	Return values
2>1	1 (or true)
2>3	0 (or false)
3+2<6	1 (or true)
- Example: Program to illustrate the use of all relational operators.

```
#include<stdio.h>
void main()
{
    printf("4>5 : %d \n", 4>5);
    printf("4>=5 : %d \n", 4>=5);
    printf("4<5 : %d \n", 4<5);
    printf("4<=5 : %d \n", 4<=5);
    printf("4==5 : %d \n", 4==5);
    printf("4!=5 : %d ", 4!=5);
}
```

Output:

```
4>5 : 0
4>=5 : 0
4<5 : 1
4<=5 : 1
4==5 : 0
4!=5 : 1
```




PROGRAMMING IN C AND DATA STRUCTURES

LOGICAL OPERATORS

- These operators are used to perform logical operations like negation, conjunction and disjunction.
- The output of logical expression is either true(1) or false(0).
- There are 3 logical operators:

Operator	Meaning	Example
&&	Logical AND	If c=5 and d=2 then ((c==5) && (d>5)) returns false.
	Logical OR	If c=5 and d=2 then ((c==5) (d>5)) returns true.
!	Logical NOT	If c=5 then !(c==5) returns false.

- All non zero values(i.e. 1, -1, 2, -2) will be treated as true.
While zero value(i.e. 0) will be treated as false.

Truth table

A	B	A&&B	A B	!A
0	0	0	0	1
0	1	0	1	
1	0	0	1	0
1	1	1	1	

- Example: Program to illustrate the use of all logical operators.

```
#include<stdio.h>
void main()
{
    clrscr ( );
    printf("7 && 0 : %d \n", 7 && 0 );
    printf("7 || 0 : %d \n", 7 || 0 );
    printf(" !0 : %d", !0 );
}
```

Output:

```
7 && 0 : 1
7 || 0 : 1
!0 : 1
```

- Example: Program to illustrate the use of both relational & Logical operators.

```
#include<stdio.h>
void main()
{
    printf("5>3 && 5<10 : %d \n", 5>3 && 5<10);
    printf(" 8<5 || 5= =5 : %d \n", 8<5 || 5==5);
    printf("!(8 = =8) : %d ", !(8==8) );
}
```

Output:

```
5>3 && 5<10 : 1
8<5 || 5= =5 : 1
!(8 = =8) : 0
```



PROGRAMMING IN C AND DATA STRUCTURES

CONDITIONAL OPERATOR

- The conditional operator is also called ternary operator it takes three operands.
- Conditional operators are used for decision making in C.
- The syntax is shown below:

(exp1)? exp2: exp3;

where exp1 is an expression evaluated to true or false;

If exp1 is evaluated to true, exp2 is executed;

If exp1 is evaluated to false, exp3 is executed.

Example: Program to find largest of 2 numbers using conditional operator.

```
#include<stdio.h>
void main()
{
    int a,b, max ;
    printf(" enter 2 distinct numbers \n");
    scanf("%d %d", &a, &b);
    max=(a>b)? a : b;
    printf(" largest number =%d ", max);
}
```

Output:

enter 2 distinct numbers

3 4

largest number = 4

BITWISE OPERATORS

- These operators are used to perform logical operation (and, or, not) on individual bits of a binary number.
- There are 6 bitwise operators:

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Truth Table

A	B	A&B	A B	A^B	~A
0	0	0	0	0	1
0	1	0	1	1	
1	0	0	1	1	0
1	1	1	1	0	

- Ex for ~ (bitwise complement)

a = 13 0000 1101
~a= 11110010

- Ex for & (bitwise AND)

a = 13 0000 1101
b = 6 0000 0110
a & b 0000 0100

- Ex for || (bitwise OR)

a = 13 0000 1101
b = 6 0000 0110
a | b 0000 1111

- Ex for ^ (bitwise xor)

a = 13 0000 1101
b = 6 0000 0110
a ^ b 0000 1011

- The operator that is used to shift the data by a specified number of bit positions towards left or right is called shift operator.

- The syntax is shown below for <<

b=a << num;

where a is value to be shifted

num is number of bits to be shifted

- The syntax is shown below for >>

b=a >> num;

- Ex for <<(left shift):

a = 13 0000 1101
b=a<<1 0001 1010

- Ex for >>(right shift):

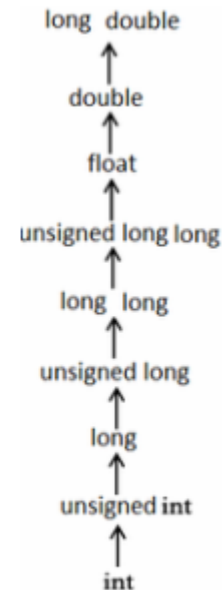
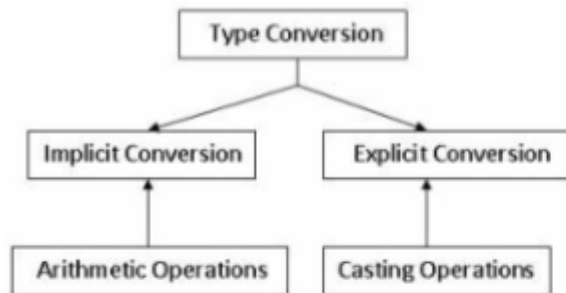
a = 13 0000 1101
b=a<<1 0000 0110



PROGRAMMING IN C AND DATA STRUCTURES

TYPE CONVERSION

- Type conversion is used to convert data of one type to data of another type.
- Type conversion is of 2 types as shown in below figure:



datatype hierarchy

IMPLICIT CONVERSION

- If a compiler converts one type of data into another type of data automatically, it is known as implicit conversions.
- There is no data loss in implicit conversion.
- The conversion always takes place from lower rank to higher rank.
For ex, int to float as shown in the above datatype hierarchy.
- For ex:

```
int a = 22, b=11;
float c = a; //c becomes 21.000000
float d=b/c=11/22.000000=11.000000/22.000000=0.500000
```
- If one operand type is same as other operand type, no conversion takes place and type of result remains same as the operands i.e. $\text{int} + \text{int} = \text{int}$
 $\text{float} + \text{float} = \text{float}$
- Conversion rules are as follows:
 - If either operand is long double, convert the other to long double.
 - Otherwise, if either operand is double, convert the other to double.
 - Otherwise, if either operand is float, convert the other to float.
 - Otherwise, convert char and short to int.
 - Then, if either operand is long, convert the other to long.
- Example: Program to illustrate implicit conversion.

```
#include<stdio.h>
void main()
{
    int a = 22, b=11;
    float d ;
    d=b/c;
    printf("d Value is : %f ", d );
}
```

Output:

d Value is : 0.500000



PROGRAMMING IN C AND DATA STRUCTURES

EXPLICIT CONVERSION

- When the data of one type is converted explicitly to another type with the help of some pre-defined functions, it is called as explicit conversion.
- There may be data loss in this process because the conversion is forceful.
- The syntax is shown below:

```
data_type1 v1;  
data_type2 v2= (data_type2) v1;  
    where v1 can be expression or variable
```

- For ex:
 float b=11.000000;
 int c = 22;
 float d=b/(float)c=11.000000/22.000000=0.500000
- Example: Program to illustrate explicit conversion.

```
#include<stdio.h>  
void main()  
{  
    float b=11.000000;  
    int c = 22;  
    float d;  
    d=b/(float)c;  
    printf("d Value is : %f ", d );  
}
```

Output:

d Value is : 0.500000



PROGRAMMING IN C AND DATA STRUCTURES

THE PRECEDENCE OF OPERATORS

- The order in which different operators are used to evaluate an expression is called precedence of operators.

Precedence Table

	Description	Represented By
1	Parenthesis	() []
1	Structure Access	. ->
2	Unary	! ~ ++ -- - * &
3	Multiply, Divide, Modulus	* / %
4	Add, Subtract	+ -
5	Shift Right, Left	>> <<
6	Greater, Less Than, etc	> < =
7	Equal, Not Equal	== !=
8	Bitwise AND	&
9	Bitwise Exclusive OR	^
10	Bitwise OR	
11	Logical AND	&&
12	Logical OR	
13	Conditional Expression	?:
14	Assignment	= += -= etc
15	Comma	,

- Example to understand the operator precedence available in C programming language.

```
#include<stdio.h>
void main()
{
    int a = 20;
    int b = 10;
    int c = 15;
    int d = 5;
    int e;
    e = (a + b) * c / d;           // ( 30 * 15 ) / 5
    printf("Value of (a + b) * c / d is : %d \n", e);
    e = ((a + b) * c) / d;        // (30 * 15) / 5
    printf("Value of ((a + b) * c) / d is : %d \n", e);
    e = (a + b) * (c / d);        // (30) * (15/5)
    printf("Value of (a + b) * (c / d) is : %d \n", e);
    e = a + (b * c) / d;          // 20 + (150/5)
    printf("Value of a + (b * c) / d is : %d ", e);
}
```

Output:

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50
```



MODULE II: BRANCHING & LOOPING

STATEMENT

- A statement is also called instruction.
- As the name indicates, instruction is used to instruct computer to perform a particular task like
 - adding two numbers
 - reading data from keyboard or
- For ex:
 sum=a+b;
 scanf("%d ", &n);
- In C, the semicolon(;) is a statement terminator.

Compound Statement

- The sequence of statement enclosed within a pair of braces { and } is called a compound statement.
- For ex:
 {
 a=l*b;
 printf("area =%d", a);
 }

CONTROL STRUCTURES

- A program is nothing but the execution of sequence of one or more instructions.
- Quite often, it is desirable to change the order of execution of statements based on certain conditions or
 - This involves a kind of decision making to see whether a particular condition has occurred or not and direct the computer to execute certain statements accordingly.
- Based on application, it is necessary / essential
 - i) To alter the flow of a program
 - ii) Test the logical conditions
 - iii) Control the flow of execution as per the selection these conditions can be placed in the program using decision-making statements.

C SUPPORTS MAINLY THREE TYPES OF CONTROL STATEMENTS

- 1) Decision making statements
 - i) if statement
 - ii) if else statement
 - iii) nested if statement
 - iv) else if ladder
 - v) switch statement
- 2) Loop control statements
 - i) while loop
 - ii) for loop
 - iii) do-while loop
- 3) Unconditional control statements
 - i) break statement
 - ii) continue statement



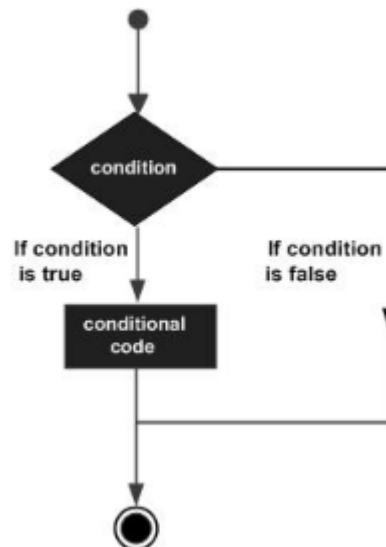
PROGRAMMING IN C AND DATA STRUCTURES

BASIC CONCEPT OF DECISION STATEMENTS

- Decision making is critical to computer programming.
- There will be many situations when you will be given 2 or more options and you will have to select an option based on the given conditions.
- For ex, we want to print a remark about a student based on secured marks and following is the situation:

1. Assume given marks are x for a student
2. If given marks are more than 95 then
3. Student is brilliant
4. If given marks are less than 30 then
5. Student is poor
6. If given marks are less than 95 and more than 30 then
7. Student is average

- Now, question is how to write programming code to handle such situation. C language provides conditional i.e., decision making statements which work based on the following flow diagram:



- There are 5 types of decision statements:
 - i) if statement
 - ii) if else statement
 - iii) nested if statement
 - iv) else if ladder
 - v) switch statement

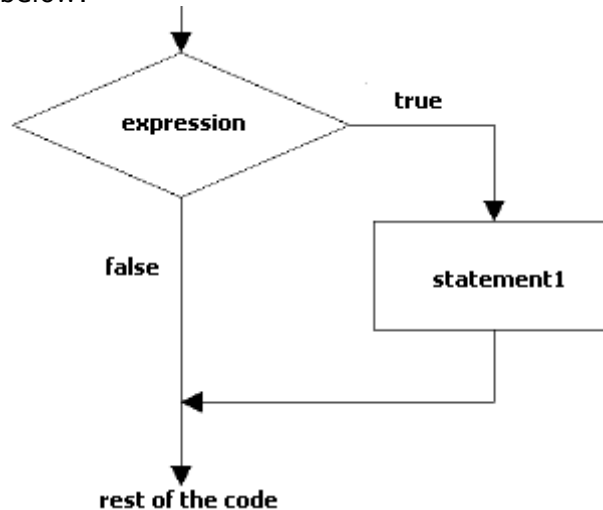


PROGRAMMING IN C AND DATA STRUCTURES

THE if STATEMENT

- This is basically a "one-way" decision statement.
- This is used when we have only one alternative.
- The syntax is shown below:

```
if(expression)
{
    statement1;
}
```
- Firstly, the expression is evaluated to true or false.
 - If the expression is evaluated to true, then statement1 is executed.
 - If the expression is evaluated to false, then statement1 is skipped.
- The flow diagram is shown below:



- Example: Program to illustrate the use of if statement.

```
#include<stdio.h>
void main()
{
    int n;
    printf("Enter any non zero integer: \n") ;
    scanf("%d", &n)
    if(n>0)
        printf("Number is positive number ");
    if(n<0)
        printf("Number is negative number ");
}
```

Output:

```
Enter any non zero integer:
7
Number is positive number
```



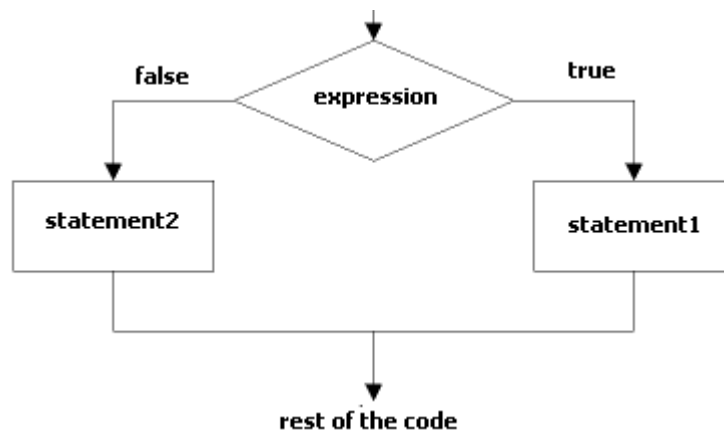

PROGRAMMING IN C AND DATA STRUCTURES

THE if else STATEMENT

- This is basically a "two-way" decision statement.
- This is used when we must choose between two alternatives.
- The syntax is shown below:

```
if(expression)
{
    statement1;
}
else
{
    statement2;
}
```

- Firstly, the expression is evaluated to true or false.
If the expression is evaluated to true, then statement1 is executed.
If the expression is evaluated to false, then statement2 is executed.
- The flow diagram is shown below:



- Example: Program to illustrate the use of if else statement.

```
#include<stdio.h>
void main()
{
    int n;
    printf("Enter any non-zero integer: \n") ;
    scanf("%d", &n)
    if(n>0)
        printf("Number is positive number");
    else
        printf("Number is negative number");
}
```

Output:

```
Enter any non-zero integer:
7
Number is positive number
```



PROGRAMMING IN C AND DATA STRUCTURES

THE nested if STATEMENT

- An if-else statement within another if-else statement is called nested if statement.
- This is used when an action has to be performed based on many decisions. Hence, it is called as multi-way decision statement.

- The syntax is shown below:

```
if(expr1)
{
    if(expr2)
        statement1
    else
        statement2
}
else
{
    if(expr3)
        statement3
    else
        statement4
}
```

- Here, firstly expr1 is evaluated to true or false.

If the expr1 is evaluated to true, then expr2 is evaluated to true or false.

If the expr2 is evaluated to true, then statement1 is executed.

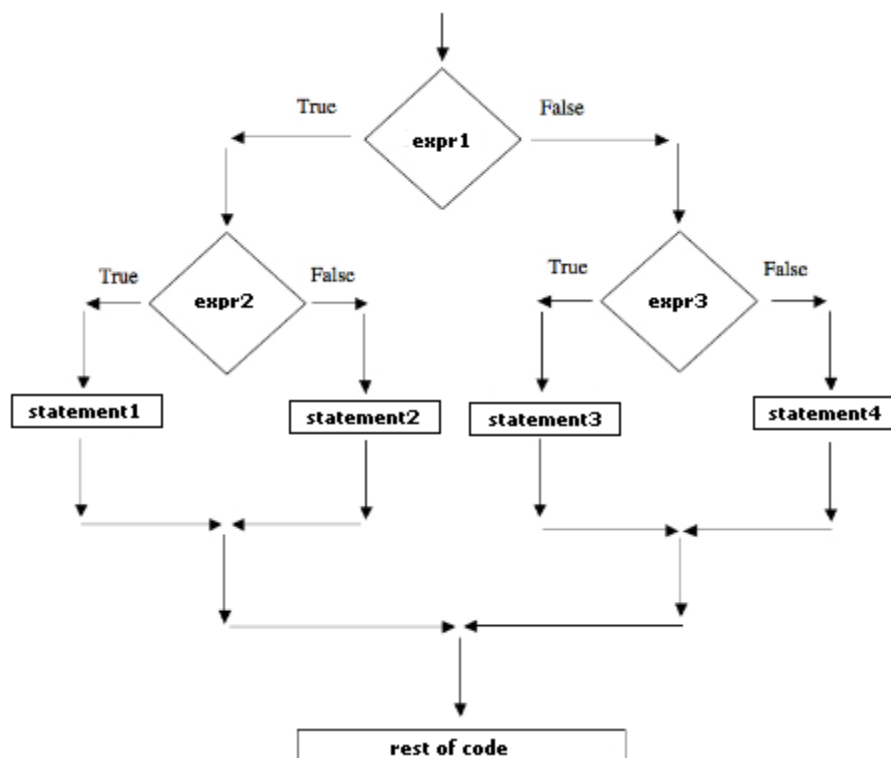
If the expr2 is evaluated to false, then statement2 is executed.

If the expr1 is evaluated to false, then expr3 is evaluated to true or false.

If the expr3 is evaluated to true, then statement3 is executed.

If the expr3 is evaluated to false, then statement4 is executed.

- The flow diagram is shown below:



**PROGRAMMING IN C AND DATA STRUCTURES**

- Example: Program to select and print the largest of the 3 numbers using nested "if-else" statements.

```
#include<stdio.h>
void main()
{
    int a,b,c;
    printf("Enter Three Values: \n");
    scanf("%d %d %d ", &a, &b, &c);
    printf("Largest Value is: ") ;
    if(a>b)
    {
        if(a>c)
            printf(" %d ", a);
        else
            printf(" %d ", c);
    }
    else
    {
        if(b>c)
            printf(" %d", b);
        else
            printf(" %d", c);
    }
}
```

Output:

```
Enter Three Values:
7 8 6
Largest Value is: 8
```

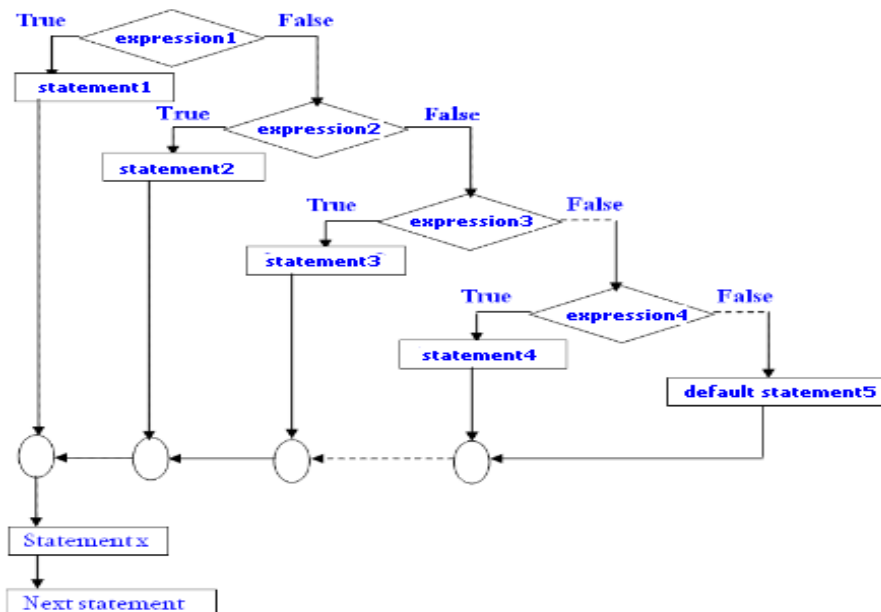


PROGRAMMING IN C AND DATA STRUCTURES

THE else if LADDER STATEMENT

- This is basically a "multi-way" decision statement.
- This is used when we must choose among many alternatives.
- The syntax is shown below:

```
if(expression1)
    statement1;
else if(expression2)
    statement2;
else if(expression3)
    statement3;
else if(expression4)
    statement4;
else
    default statement5;
```
- The expressions are evaluated in order (i.e. top to bottom).
- If an expression is evaluated to true, then
 - statement associated with the expression is executed &
 - control comes out of the entire else if ladder
- For ex, if expression1 is evaluated to true, then statement1 is executed.
If all the expressions are evaluated to false, the last statement4(default case) is executed.



Example: Program to illustrate the use of else if ladder statement.

```
void main ( )
{
    int n;
    printf("Enter any integer:") ;
    scanf("%d", &n)
    if(n>0)
        printf ("Number is Positive");
    else if(n< 0)
        printf ("Number is Negative");
    else if(n== 0)
        printf ("Number is Zero");
    else
        printf ("Invalid input");
}
```

Output:

Enter any integer: 7
Number is Positive

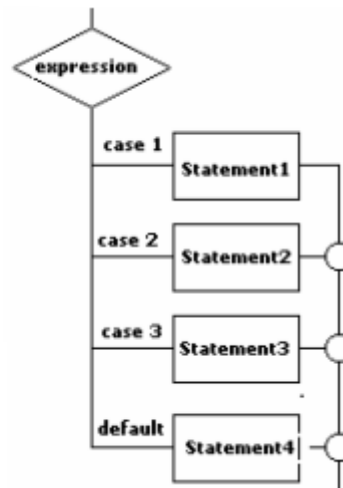


PROGRAMMING IN C AND DATA STRUCTURES

THE switch STATEMENT

- This is basically a "multi-way" decision statement.
- This is used when we must choose among many alternatives.
- The syntax & flow diagram is shown below:

```
switch(choice)
{
    case value1: statement1
                break;
    case value2: statement2
                break;
    case value3: statement3
                break;
    default: statement4;
}
```



- Here, choice can be either any integer value or a character.
- Based on this integer value, the control is transferred to a particular case-value where necessary statements are executed.
- During executing, if break statement is encountered, then the control comes out of the switch block.
- If the value of the choice does not match with any of the case values (i.e. value1, value2, value3) then control goes to default label.
- All case-values must be different.
- Example: Program to illustrate the use of switch statement.

```
void main()
{
    char grade;          // local variable definition
    printf("enter grade A to D: \n");
    scanf("%c",&grade);
    switch(grade)
    {
        case 'A': printf("Excellent! \n ");
                   break;
        case 'B': printf("Well done \n ");
                   break;
        case 'C': printf("You passed \n ");
                   break;
        case 'D': printf("Better try again\n ");
                   break;
        default:  printf("Invalid grade \n ");
                   return;
    }
    printf("Your grade is %c", grade);
    return 0;
}
```

Output:

```
enter grade A to D:
B
Well done
Your grade is B
```



PROGRAMMING IN C AND DATA STRUCTURES

BASIC CONCEPT OF LOOP

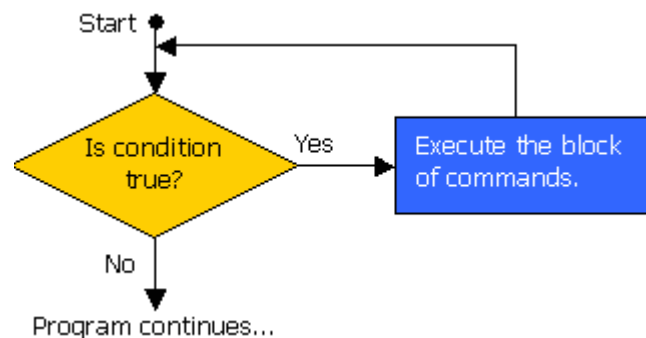
- Let's consider a situation when you want to write a message "Welcome to C language" five times. Here is a simple C program to do the same:

```
#include<stdio.h>
void main()
{
    printf( " Welcome to C language \n");
    printf( " Welcome to C language \n");
    printf( " Welcome to C language \n");
    printf( " Welcome to C language \n");
    printf( " Welcome to C language \n");
}
```

Output:

```
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
```

- When the program is executed, it produces the above result.
- It was simple, but again let's consider another situation when you want to write the same message thousand times, what you will do in such situation?
- Are we going to write printf() statement thousand times? No, not at all.
- C language provides a concept called loop, which helps in executing one or more statements up to desired number of times.
- Loops are used to execute one or more statements repeatedly.
- The flow diagram is shown below:



- There are 3 types of loops in C programming:
 - 1) while loop
 - 2) for loop
 - 3) do while loop



PROGRAMMING IN C AND DATA STRUCTURES

THE while LOOP

- A while loop statement can be used to execute a set of statements repeatedly as long as a given condition is true.

- The syntax is shown below:

```
while(expression)
{
    statement1;
    statement2;
}
```

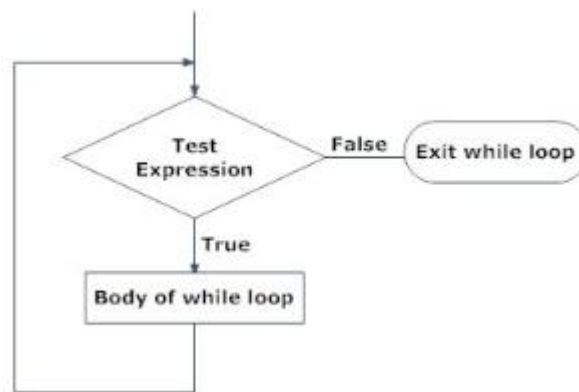
- Firstly, the expression is evaluated to true or false.

- If the expression is evaluated to false, the control comes out of the loop without executing the body of the loop.

- If the expression is evaluated to true, the body of the loop (i.e. statement1) is executed.

- After executing the body of the loop, control goes back to the beginning of the while statement and expression is again evaluated to true or false. This cycle continues until expression becomes false.

- The flow diagram is shown below:



Example: Program to display a message 5 times using while statement.

```
#include<stdio.h>
void main()
{
    int i=1;
    while(i<=5)
    {
        printf("Welcome to C language \n");
        i=i+1;
    }
}
```

Output:

```
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
```

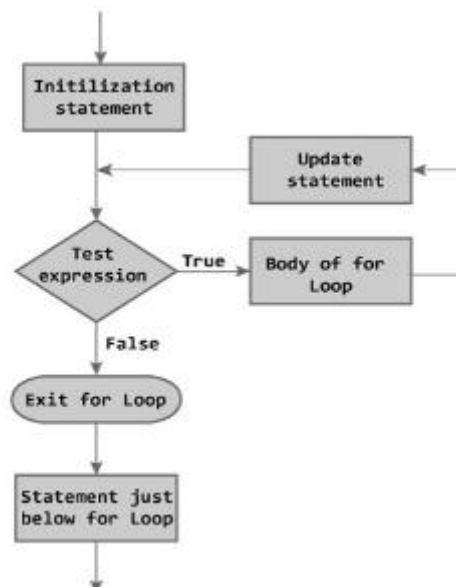


PROGRAMMING IN C AND DATA STRUCTURES

THE for LOOP

- A for loop statement can be used to execute a set of statements repeatedly as long as a given condition is true.
- The syntax is shown below:

```
for(expr1;expr2;expr3)
{
    statement1;
}
```
- Here, expr1 contains initialization statement
expr2 contains limit test expression
expr3 contains updating expression
- Firstly, expr1 is evaluated. It is executed only once.
- Then, expr2 is evaluated to true or false.
- If expr2 is evaluated to false, the control comes out of the loop without executing the body of the loop.
- If expr2 is evaluated to true, the body of the loop (i.e. statement1) is executed.
- After executing the body of the loop, expr3 is evaluated.
- Then expr2 is again evaluated to true or false. This cycle continues until expression becomes false.
- The flow diagram is shown below:



Example: Program to display a message 5 times using for statement.

```
#include<stdio.h>
void main()
{
    int i;
    for(i=1;i<=5;i++)
    {
        printf("Welcome to C language \n");
    }
}
```

Output:

```
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
```




PROGRAMMING IN C AND DATA STRUCTURES

THE do while STATEMENT

- When we do not know exactly how many times a set of statements have to be repeated, do-while statement can be used.

- The syntax is shown below:

```
do
{
    statement1;
}while(expression);
```

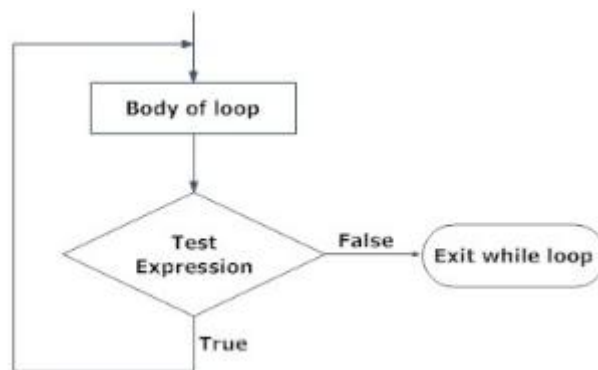
- Firstly, the body of the loop is executed. i.e. the body of the loop is executed at least once.

- Then, the expression is evaluated to true or false.

- If the expression is evaluated to true, the body of the loop (i.e. statement1) is executed

- After executing the body of the loop, the expression is again evaluated to true or false. This cycle continues until expression becomes false.

- The flow diagram is shown below:



Example: Program to display a message 5 times using do while statement.

```
#include<stdio.h>
void main()
{
    int i=1;
    do
    {
        printf("Welcome to C language \n");
        i=i+1;
    }while(i<=5);
}
```

Output:

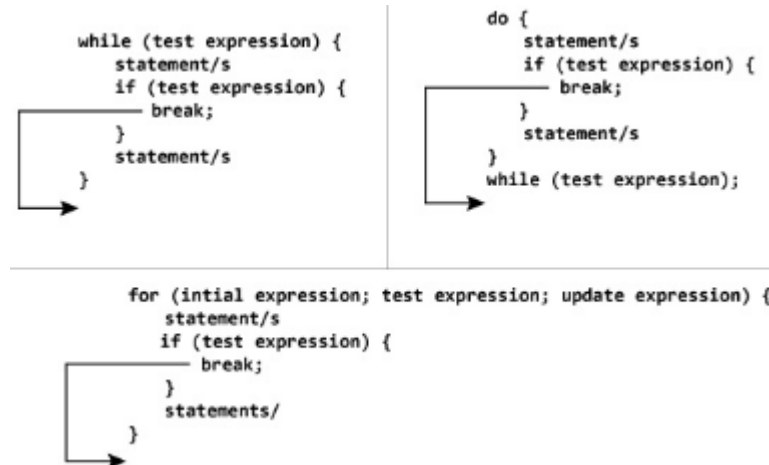
```
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
Welcome to C language
```



PROGRAMMING IN C AND DATA STRUCTURES

THE break STATEMENT

- The break statement is jump statement which can be used in switch statement and loops.
- The break statement works as shown below:
 - 1) If break is executed in a switch block, the control comes out of the switch block and the statement following the switch block will be executed.
 - 2) If break is executed in a loop, the control comes out of the loop and the statement following the loop will be executed.
- The syntax is shown below:



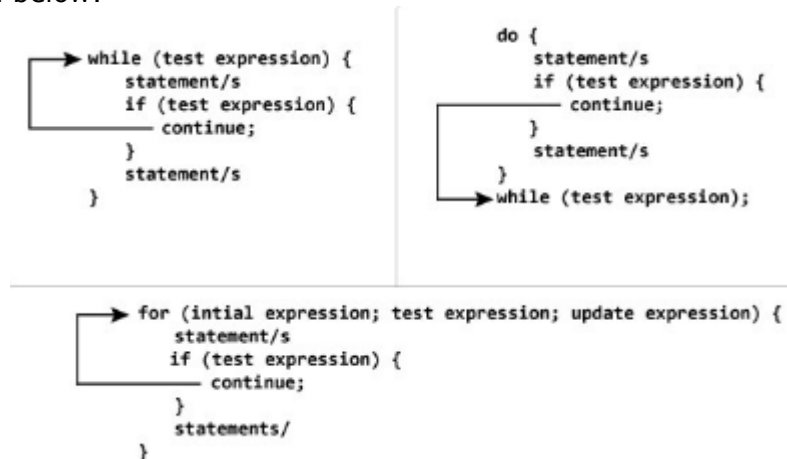
Example: Write the program given in switch statement.



PROGRAMMING IN C AND DATA STRUCTURES

THE continue STATEMENT

- During execution of a loop, it may be necessary to skip a part of the loop based on some condition. In such cases, we use continue statement.
- The continue statement is used only in the loop to terminate the current iteration.
- The syntax is shown below:



Example: Program to read and add only positive numbers using continue statement.

```
#include<stdio.h>
void main()
{
    int i=1, num, sum =0;
    for(i=0; i < 5; i ++)
    {
        printf(" Enter an integer:");
        scanf( "%d", &num);
        if(num < 0)
        {
            printf("you have entered a negative number \n");
            continue ;    // skip the remaining part of loop
        }
        sum=sum+num;
    }
    printf("The sum of the Positive Integers Entered = %d", sum);
}
```

Output:

```
Enter an integer: 10
Enter an integer:-15
You have entered a negative number
Enter an integer: 15
Enter an integer: -100
You have entered a negative number
Enter an integer: 30
The sum of the positive integers entered = 55
```



PROGRAMMING IN C AND DATA STRUCTURES

THE goto AND labels

- goto statement can be used to branch unconditionally from one point to another in the program.
- The goto requires a label in order to identify the place where the branch is to be made.
- A label is any valid variable name and must be followed by a colon (:).
- The label is placed immediately before the statement where the control is to be transferred.
- The syntax is shown below:

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

- Example: Program to detect the entered number as to whether it is even or odd. Use goto statement.

```
#include<stdio.h>  
void main()  
{  
    int x;  
    printf("Enter a Number: \n");  
    scanf("%d", &x);  
    if(x % 2 == 0)  
        goto even;  
    else  
        goto odd;  
    even: printf("%d is Even Number");  
    return;  
    odd: printf(" %d is Odd Number");  
}
```

Output:

```
Enter a Number:  
5  
5 is Odd Number.
```



MODULE III: ARRAYS & STRINGS

BASIC CONCEPT OF ARRAYS

- Consider a situation, where we need to store 5 integer numbers.
- If we use simple variable and data type concepts, then we need 5 variables of int data type and program will be something as follows:

```
#include<stdio.h>
void main()
{
    int number1;
    int number2;
    int number3;
    int number4;
    int number5;

    number1 = 10;
    number2 = 20;
    number3 = 30;
    number4 = 40;
    number5 = 50;

    printf( "number1: %d \n", number1);
    printf( "number2: %d \n", number2);
    printf( "number3: %d \n", number3);
    printf( "number4: %d \n", number4);
    printf( "number5: %d ", number5);
}
```

Output:

```
number1: 10
number2: 20
number3: 30
number4: 40
number5: 50
```

- It was simple, because we had to store just 5 integer numbers. Now let's assume we have to store 5000 integer numbers, so what is next? Are we going to use 5000 variables?
- To handle such situation, C language provides a concept called the array.
- Some examples where arrays can be used are
 - 1) List of temperatures recorded every hour in a day, or a month, or a year
 - 2) List of employees in an organization
 - 3) List of products and their cost sold by a store
 - 4) Test scores of a class of students

ARRAY

- Array is a collection of elements of same data type.
- The elements are stored sequentially one after the other in memory.
- Any element can be accessed by using
 - name of the array
 - position of element in the array
- Arrays are of 2 types:
 - 1) Single dimensional array
 - 2) Multi dimensional array



PROGRAMMING IN C AND DATA STRUCTURES

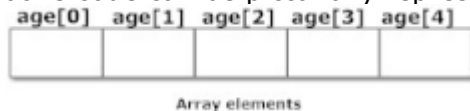
SINGLE DIMENSIONAL ARRAY

- A single dimensional array is a linear list consisting of related elements of same type.
- In memory, all the elements are stored in continuous memory-location one after the other.

Declaration of Single Dimensional Arrays

- The syntax is shown below:
 data_type array_name[array_size]
 where data_type can be int, float or char
 array_name is name of the array
 array_size indicates number of elements in the array

- For ex:
 int age[5];
- The above code can be pictorially represented as shown below:



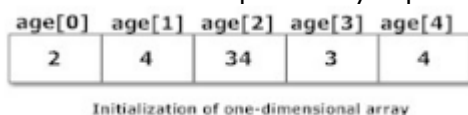
- Note that, the first element is numbered 0 and so on.
- Here, the size of array 'age' is 5 times the size of int because there are 5 elements.

Storing Values in Arrays

- The values can be stored in array using following three methods:
 - 1) Initialization
 - 2) Assigning values
 - 3) Input values from keyboard

Initialization of One-Dimensional Array

- The syntax is shown below:
 data_type array_name[array_size]={v₁,v₂,v₃};
 where v₁, v₂, v₃ are values
- For ex:
 int age[5]={2,4,34,3,4};
- The above code can be pictorially represented as shown below:



- Example: Program to illustrate the initialization of one-dimensional array.

```
#include<stdio.h>
void main()
{
    int age[5]={2,4,34,3,4};
    printf("Value in array age[0] : %d \n", age[0]);
    printf("Value in array age[1] : %d \n", age[1]);
    printf("Value in array age[2] : %d \n", age[2]);
    printf("Value in array age[3] : %d \n", age[3]);
    printf("Value in array age[4] : %d ",age[4]);
}
```

Output:

```
Value in array age[0] :2
Value in array age[1] :4
Value in array age[2] :34
Value in array age[3] :3
Value in array age[4] :4
```



PROGRAMMING IN C AND DATA STRUCTURES

Assigning values to One-Dimensional Array

- For ex:

```
int age[5];
age[0]=2;    //value 2 stored in array 'age' at position 0
age[1]=4;    //value 4 stored in array 'age' at position 1
age[2]=34;   //value 34 stored in array 'age' at position 2
age[3]=3;    //value 3 stored in array 'age' at position 3
age[4]=4;    //value 4 stored in array 'age' at position 4
```

- Example: Program to illustrate assigning values to one-dimensional array.

```
#include<stdio.h>
void main()
{
    int age[5];
    age[0]=2;
    age[1]=4;
    age[2]=34;
    age[3]=3;
    age[4]=4;

    printf("Value in array age[0] : %d \n", age[0]);
    printf("Value in array age[1] : %d \n", age[1]);
    printf("Value in array age[2] : %d \n", age[2]);
    printf("Value in array age[3] : %d \n", age[3]);
    printf("Value in array age[4] : %d ", age[4]);
}
```

Output:

```
Value in array age[0] :2
Value in array age[1] :4
Value in array age[2] :34
Value in array age[3] :3
Value in array age[4] :4
```



PROGRAMMING IN C AND DATA STRUCTURES

Reading/Writing to One-Dimensional Array

- For ex:

```
int age[5]
scanf("%d", &age[0]);    //read data from keyboard into array age at position 0.
scanf("%d", &age[1]);    //read data from keyboard into array age at position 1.
scanf("%d", &age[2]);    //read data from keyboard into array age at position 2.
scanf("%d", &age[3]);    //read data from keyboard into array age at position 3.
scanf("%d", &age[4]);    //read data from keyboard into array age at position 4.
```

- In general, to read 5 values, we can write as follows:

```
for(i=0;i<5;i++)
{
    scanf("%d", &age[i]);
}
```

- Similarly, to display 5 elements stored in the array, we can write:

```
for(i=0;i<5;i++)
{
    printf("%d", age[i]);
}
```

- Example: Program to illustrate reading/writing to one-dimensional array.

```
#include<stdio.h>
void main()
{
    int age[5], i;

    printf("enter 5 numbers:\n");
    for(i=0;i<5;i++)
    {
        scanf("%d", &age[i]);
    }

    for(i=0;i<5;i++)
    {
        printf("Value in array age[%d] : %d \n ", i, age[i]);
    }
}
```

Output:

```
enter 5 numbers:
2 4 34 3 4
Value in array age[0] :2
Value in array age[1] :4
Value in array age[2] :34
Value in array age[3] :3
Value in array age[4] :4
```




PROGRAMMING IN C AND DATA STRUCTURES

MULTI DIMENSIONAL ARRAYS

- Arrays with two or more dimensions are called multi dimensional arrays.
- For ex:
`int matrix[2][3];`
- The above code can be pictorially represented as shown below:

	col 1	col 2	col 3
row 1	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>
row 2	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>

Figure: Multidimensional Arrays

- A two dimensional array is used when elements are arranged in a tabular fashion.
- Here, to identify a particular element, we have to specify 2 indices:
 - First index identifies the row number of the element and
 - Second index identifies the column number of the element

Declaration of Two Dimensional Arrays

- The syntax is shown below:
`data_type array_name[array_size][array_size];`
- For ex:
`int matrix[2][3];`
- The above code can be pictorially represented as shown below:

	col 1	col 2	col 3
row 1			
row 2			

Initialization of Two Dimensional Arrays

- For ex:
`int matrix[2][3]= {1, 23, 11, 44, 5, 6};`
- The above code can be pictorially represented as shown below:

	col 1	col 2	col 3
row 1	1	23	11
row 2	44	5	67



PROGRAMMING IN C AND DATA STRUCTURES

Reading and Writing Two Dimensional Arrays

- For ex:

```
int matrix[2][3]
scanf("%d", &matrix[0][0]); //read data from keyboard into matrix at row=0 col=0.
scanf("%d", &matrix[0][1]); //read data from keyboard into matrix at row=0 col=1.
scanf("%d", &matrix[0][2]); //read data from keyboard into matrix at row=0 col=2.
scanf("%d", &matrix[1][0]); //read data from keyboard into matrix at row=1 col=0.
scanf("%d", &matrix[1][1]); //read data from keyboard into matrix at row=1 col=1.
scanf("%d", &matrix[1][2]); //read data from keyboard into matrix at row=1 col=2.
```

- In general, to read 6 values, we can write:

```
for(i=0;i<2;i++)
{
    for(j=0;j<3;j++)
    {
        scanf("%d", &matrix[i][j]);
    }
}
```

- Similarly to display 6 elements stored in the matrix, we can write:

```
for(i=0;i<2;i++)
{
    for(j=0;j<3;j++)
    {
        printf("%d ", matrix[i][j]);
    }
}
```

- Example: Program to illustrate reading/writing to two-dimensional array.

```
#include<stdio.h>
void main()
{
    int matrix[2][3], i, j;
    printf("enter elements of 2*3 matrix: \n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("elements of the matrix are: \n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d \t", matrix[i][j]);
        }
        printf("\n");
    }
}
```

Output:

enter elements of 2*3 matrix:

1 23 11

44 5 6

elements of the matrix are:

1 23 11

44 5 6



PROGRAMMING IN C AND DATA STRUCTURES

ARRAYS AND FUNCTIONS

- The arrays can be passed to functions using two methods:

1) Passing Individual Elements of Array

- All array-elements can be passed as individual elements to a function like any other variable.
- The array-element is passed as a value parameter i.e. any change in the formal-parameter will not affect the actual-parameter i.e. array-element.
- Example: Program to illustrate passing individual elements of an array to a function.

```
#include<stdio.h>
void display(int a)
{
    printf("%d", a);
}

void main()
{
    int c[3]={2,3,4};
    display(c[2]);    //Passing array-element c[2] only.
}
```

Output:

4

2) Passing the Whole Array

- Here, the parameter passing is similar to pass by reference.
- In pass by reference, the formal parameters are treated as aliases for actual parameters.
- So, any changes in formal parameter imply there is a change in actual parameter.
- Example: Program to find average of 6 marks using pass by reference.

```
#include <stdio.h>
void average(int m[])
{
    int i ,avg;
    int avg, sum=0;
    for(i=0;i<6;i++)
    {
        sum= sum+ m[i];
    }
    avg =(sum/6);
    printf("aggregate marks= %d ", avg);
}

void main()
{
    int m[6]={60, 50, 70, 80, 40, 80, 70};
    average(m);    // Only name of array is passed as argument
}
```

Output:

aggregate marks= 70



PROGRAMMING IN C AND DATA STRUCTURES

STRINGS

- Array of character are called strings.
- A string is terminated by null character /0.
- For ex:
"c string tutorial"
- The above string can be pictorially represented as shown below:

c		s	t	r	i	n	g		t	u	t	o	r	i	a	l	\0
---	--	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	----

STRING VARIABLE

- There is no separate data type for strings in C.
- They are treated as arrays of type 'char'.
- So, a variable which is used to store an array of characters is called a string variable.

Declaration of String

- Strings are declared in C in similar manner as arrays.
Only difference is that, strings are of 'char' type.
- For ex:
char s[5]; //string variable name can hold maximum of 5 characters including NULL character
- The above code can be pictorially represented as shown below:

s[0]	s[1]	s[2]	s[3]	s[4]
				\0

Initialization of String

- For ex:
char s[5]={'r', 'a', 'm', 'a' };
char s[9]="rama";
- The above code can be pictorially represented as shown below:

s[0]	s[1]	s[2]	s[3]	s[4]
r	a	m	a	\0

- Example: Program to illustrate the initialization of string.

```
#include<stdio.h>
void main()
{
    char s[5]={'r','a','m','a'};    //or char s[5]="rama";
    printf("Value in array s[0] : %c \n", s[0]);
    printf("Value in array s[1] : %c \n", s[1]);
    printf("Value in array s[2] : %c \n", s[2]);
    printf("Value in array s[3] : %c ", s[3]);
}
```

Output:

```
Value in array s[0] : r
Value in array s[1] : a
Value in array s[2] : m
Value in array s[3] : a
```



PROGRAMMING IN C AND DATA STRUCTURES

Reading & Printing Strings

- The strings can be read from the keyboard and can be displayed onto the monitor using following 2 formatted functions:

Formatted input function: scanf()

Formatted output function: printf()

- Example: Program to illustrate the use of scanf() and printf().

```
#include<stdio.h>
void main()
{
    char name[10];
    printf("enter your name: \n");
    scanf("%s", name);
    printf("welcome: ");
    printf("%s", name);
}
```

Output:

```
enter your name:
rama
welcome: rama
```

STRING INPUT/OUTPUT FUNCTIONS: gets, puts

- The strings can be read from the keyboard and can be displayed onto the monitor using following 2 unformatted functions:

Unformatted input function: gets()

Unformatted output function: puts()

- Example: Program to illustrate the use of gets() and puts().

```
#include<stdio.h>
void main()
{
    char name[10];
    printf("enter your name: \n");
    gets(name);           //same as scanf("%s", name);
    printf("welcome: ");
    puts(name);           //same as printf("%s", name);
}
```

Output:

```
enter your name:
rama
welcome: rama
```



PROGRAMMING IN C AND DATA STRUCTURES

STRING MANIPULATION FUNCTIONS FROM THE STANDARD LIBRARY

- Strings are often needed to be manipulated by programmer according to the need of a problem.
- All string manipulation can be done manually by the programmer but, this makes programming complex and large.
- To solve this, the C supports a large number of string handling functions.
- There are numerous functions defined in <string.h> header file.

S.N.	Function & Purpose
1	<code>strcpy(s1, s2);</code> Copies string s2 into string s1.
2	<code>strcat(s1, s2);</code> Concatenates string s2 onto the end of string s1.
3	<code>strlen(s1);</code> Returns the length of string s1.
4	<code>strcmp(s1, s2);</code> Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	<code>strchr(s1, ch);</code> Returns a pointer to the first occurrence of character ch in string s1.
6	<code>strstr(s1, s2);</code> Returns a pointer to the first occurrence of string s2 in string s1.

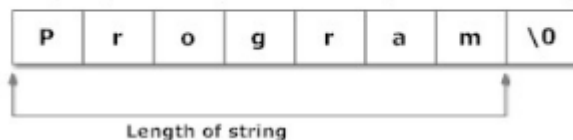
strlen()

- This function calculates the length of string. It takes only one argument, i.e., string-name.
- The syntax is shown below:

temp_variable = strlen(string_name);

```
char c[]={'P','r','o','g','r','a','m','\0'};
temp=strlen(c);
```

Then, temp will be equal to 7 because, null character '\0' is not counted.



- Example: Program to illustrate the use of strlen().

```
#include<string.h>
#include<stdio.h>
void main()
{
    char c[20];
    int len;
    printf("Enter string whose length is to be found:");
    gets(c);
    len=strlen(c);
    printf("\n Length of the string is %d ", len);
}
```

Output:

Enter string whose length is to be found: program
Length of the string is 7



PROGRAMMING IN C AND DATA STRUCTURES

strcpy()

- This function copies the content of one string to the content of another string. It takes 2 arguments.
- The syntax is shown below:
 strcpy(destination,source);
 where source and destination are both the name of the string.
- Example: Program to illustrate the use of strcpy().

```
#include<string.h>
#include<stdio.h>
void main()
{
    char src[20],dest[20];
    printf("Enter string: ");
    gets(src);
    strcpy(dest, src); //Content of string src is copied to string dest
    printf("Copied string: ");
    puts(dest);
}
```

Output:

Enter string: vtunotesbysri
Copied string: vtunotesbysri

strcat()

- This function joins 2 strings. It takes two arguments, i.e., 2 strings and resultant string is stored in the first string specified in the argument.
- The syntax is shown below:
 strcat(first_string,second_string);
- Example: Program to illustrate the use of strcat().

```
#include <stdio.h>
#include <string.h>
void main()
{
    char str1[10], str2[10];
    printf("Enter First String:");
    gets(str1);
    printf("\n Enter Second String:");
    gets(str2);
    strcat(str1,str2); //concatenates str1 and str2 and
    printf("\n Concatenated String is ");
    puts(str1);      //resultant string is stored in str1
}
```

Output:

Enter First String: rama
Enter Second String: krishna
Concatenated String is ramakrishna



PROGRAMMING IN C AND DATA STRUCTURES

strcmp()

- This function compares 2 string and returns value 0, if the 2 strings are equal. It takes 2 arguments, i.e., name of two string to compare.
- The syntax is shown below:

temp_variable=strcmp(string1,string2);

- Example: Program to illustrate the use of strcmp().

```
#include <string.h>
#include<stdio.h>
void main()
{
    char str1[30],str2[30];
    printf("Enter first string: ");
    gets(str1);
    printf("Enter second string: ");
    gets(str2);
    if(strcmp(str1,str2)==0)
        printf("Both strings are equal");
    else
        printf("Strings are unequal");
}
```

Output:

```
Enter first string: rama
Enter second string: rama
Both strings are equal
```




PROGRAMMING IN C AND DATA STRUCTURES

WRITING SOME USEFUL STRING FUNCTIONS OF OUR OWN

- Example: Program to find length of a string without using strlen().

```
#include<stdio.h>
void main(void)
{
    char str1[25];
    int len=0;
    printf("Enter string whose length is to be found:");
    gets(str1);

    while(str1[len]!='\0')
        len++;           //here the length of string is calculated.

    printf("Length of the string is %d", len);
}
```

Output:

Enter string whose length is to be found: vtunotesbysri
Length of the string is 13

- Example: Program to concatenate two strings without using strcat().

```
void main()
{
    char str1[25],str2[25];
    int i=0,j=0;
    printf(" Enter First String:");
    gets(str1);
    printf("\n Enter Second String:");
    gets(str2);
    while(str1[i]!='\0')
        i++;
    while(str2[j]!='\0')
    {
        str1[i]=str2[j];
        j++;
        i++;
    }
    str1[i]='\0';
    printf("\n Concatenated String is ");
    puts(str1);
}
```

Output:

Enter First String: rama
Enter Second String: krishna
Concatenated String is ramakrishna

**PROGRAMMING IN C AND DATA STRUCTURES**

- Example: Program to copy one string into other without using strcpy().

```
#include<stdio.h>
void main()
{
    char src[100], dest[100];
    int i;
    printf("Enter string:");
    gets(src);
    i = 0;
    while (src[i] != '\0')
    {
        dest[i] = src[i];
        i++;
    }
    dest[i] = '\0';
    printf("Copied String ; %s ", dest);
}
```

Output:

Enter string : vtunotesbysri
Copied String : vtunotesbysri

- Example: Program to compare 2 strings without using strcmp().

```
#include<stdio.h>
void main()
{
    char str1[100],str2[100];
    int i=0,flag=0;
    printf("Enter first string: ");
    scanf("%s",str1);
    printf("Enter second string: ");
    scanf("%s",str2);
    while(str1[i]!='\0' && str2[i]!='\0')
    {
        if(str1[i]!=str2[i])
        {
            flag=1;
            break;
        }
        i++;
    }
    if (flag==0 && str1[i]=='\0' && str2[i]=='\0')
        flag=1;
    else
        flag=0;
    if(flag == 1)
        printf("Both strings are equal");
    else
        printf("Both strings are not equal");
}
```

Output:

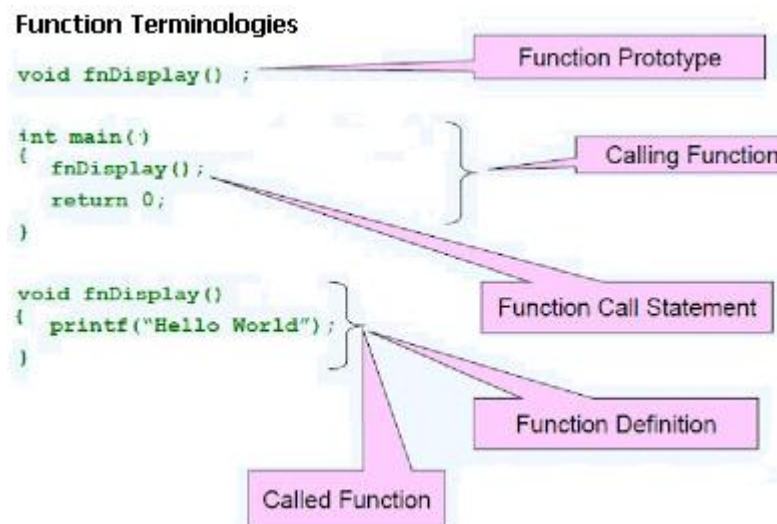
Enter first string: rama
Enter second string: rama
Both strings are equal



MODULE III(COND.): FUNCTIONS

FUNCTION IN C

- A function is a block of code to perform a specific task.
- Every C program has at least one function `main()`. Without `main()` function, there is technically no C program.
- Program to print a sentence using function is shown below.



TYPES OF C FUNCTIONS

- There are 2 types of functions in C programming:
 - Library function
 - User defined function

Library Function

- Library functions are the in-built function in C compiler.

For example:

- `main()` //The execution of every C program starts from this `main()` function
- `printf()` //printf() is used for displaying output in C
- `scanf()` //scanf() is used for taking input in C

User Defined Function

- C allows programmer to define their own function according to their requirement. These types of functions are known as user-defined functions.
- Suppose, a programmer wants to find factorial of a number and check whether it is prime or not in same program. Then, he can create two separate user-defined functions in that program:
 - one for finding factorial and
 - other for checking whether it is prime or not

ADVANTAGES OF USER DEFINED FUNCTIONS

- 1) User defined functions helps to decompose the large program into small segments which makes programmer easy to understand, maintain and debug.
- 2) If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
- 3) Programmer working on large project can divide the workload by making different functions.



PROGRAMMING IN C AND DATA STRUCTURES

FUNCTIONS AND PROGRAM STRUCTURE

- Basic structure of C program with function is shown below:

```
#include <stdio.h>
void function_name();      //function declaration
int main()
{
    .....
    .....
    function_name();        // function call
    .....
    .....
}

void function_name()      //function definition
{
    .....
    .....
}
```

- As mentioned earlier, every C program begins from main() and program starts executing the codes inside main() function.
- When the control of program reaches to function_name() inside main() function, the control of program jumps to void function_name() and executes the codes inside it.
- When all the codes inside that user-defined function are executed, control of the program jumps to the statement just after function_name() from where it is called.

Function Declaration

- Every function in C program should be declared before they are used.
- Function declaration gives compiler information about
 - function name
 - type of arguments to be passed and
 - return type
- The syntax is shown below:


```
return_type function_name(type(1) argument(1),...,type(n) argument(n));
```

Function Call

- Control of the program cannot be transferred to user-defined function unless it is called invoked.
- The syntax is shown below:


```
function_name(argument(1),....argument(n));
```

Function Definition

- Function definition contains programming codes to perform specific task.
- The syntax is shown below:


```
return_type function_name(type(1) argument(1),...,type(n) argument(n))
{
    //body of function
}
```

- Example: Program to Print a sentence using function.

```
#include<stdio.h>
void display();      //function declaration

void main()
{
    display();        //function call
}

void display()        //function definition
{
    printf("C Programming");
    return;
}
```

Output:

C Programming



PROGRAMMING IN C AND DATA STRUCTURES

ACTUAL AND FORMAL ARGUMENTS

- Argument (or parameter) refers to data that is passed to function (function definition) while calling function.
- Arguments listed in function calling statements are referred to as actual arguments.

These actual values are passed to a function to compute a value or to perform a task.

- The arguments used in the function declaration are referred as formal arguments.

They are simply formal variables that accept or receive the values supplied by the calling function.

- The number of actual and formal arguments and their data types should be same.
- Example: Program to add two integers. Make a function add integers and display sum in main() function.

```
#include <stdio.h>
int add(int a, int b);
int main()
{
    int a,b,sum;
    printf("Enters two number to add \n");
    scanf("%d %d", &a,&b);          //actual arguments
    sum=add(a,b);
    printf("\n sum=%d", sum);
    return 0;
}
int add(int a,int b)                //formal arguments
{
    int sum;
    sum=a+b;
    return sum;
}
```

Output:

```
Enters two number to add
2 3
sum=5
```



PROGRAMMING IN C AND DATA STRUCTURES

ARGUMENT PASSING – CALL BY VALUE

- In this type, value of actual arguments are passed to the formal arguments and the operation is done on the formal arguments.
- Any changes made in the formal arguments does not effect the actual arguments because formal arguments are photocopy of actual arguments.
- Changes made in the formal arguments are local to the block of called-function.
- Once control returns back to the calling-function the changes made vanish.
- Example: Program to send values using call-by-value method.

```
#include <stdio.h>
void main()
{
    int x,y,
    printf("enter values of x & y : ");
    scanf("%d %d ", &x, &y);
    printf("\n old values x=%d y =%d", x, y);
    change(x,y) ;
    printf("\n new values x=%d y =%d", x, y);
}
void change(int a, int b)
{
    k=a;
    a=b;
    b=k;
    return;
}
```

Output:

```
enter values of x & y : 2 3
old values x=2 y =3
new values x=2 y =3
```



PROGRAMMING IN C AND DATA STRUCTURES

CLASSIFICATION OF USER-DEFINED FUNCTIONS

1. Function with no arguments and no return value
2. Function with no arguments and return value
3. Function with arguments but no return value
4. Function with arguments and return value

Example: Program to illustrate function with no arguments and no return value.

```
#include <stdio.h>
void add();
void main()
{
    add();
}
void add()
{
    int a, b, sum;
    printf("Enters two number to add : ");
    scanf("%d %d", &a, &b);
    sum=a+b;
    printf("\n sum=%d", sum);
    return;
}
```

Output:

Enters two number to add : 2 3
sum=5

Example: Program to illustrate function with no arguments and return value.

```
#include <stdio.h>
int add();
void main()
{
    int sum;
    sum=add();
    printf("\n sum=%d", sum);
}
int add()
{
    int a, b, sum;
    printf("Enters two number to add \n");
    scanf("%d %d", &a, &b);
    sum=a+b;
    return sum;
}
```

Output:

Enters two number to add
2 3
sum=5

**PROGRAMMING IN C AND DATA STRUCTURES****Example: Program to illustrate function with arguments but no return value**

```
#include <stdio.h>
void add(int a, int b);
void main()
{
    int a, b, sum;
    printf("Enters two number to add \n");
    scanf("%d %d", &a, &b);
    add(a, b);
}
void add(int a, int b)
{
    sum= a+ b;
    printf("\n sum=%d", sum);
    return;
}
```

Output:

```
Enters two number to add
2 3
sum=5
```

Example: Program to illustrate function with arguments and return value

```
#include <stdio.h>
int add(int a, int b);
int main()
{
    int a, b, sum;
    printf("Enters two number to add \n");
    scanf("%d %d", &a, &b);
    sum=add(a,b);
    printf("\n sum=%d", sum);
    return 0;
}

int add(int a, int b)
{
    int sum;
    sum=a+b;
    return sum;           //return statement of function
}
```

Output:

```
Enters two number to add
2 3
sum=5
```




PROGRAMMING IN C AND DATA STRUCTURES

RECURSION

- A function that calls itself is known as recursive function.
- Example: Program to find sum of first n natural numbers using recursion.

```
#include <stdio.h>

int add(int n)
{
    if(n==0)
        return n;
    else
        return n+add(n-1); /*self call to function add() */
}

void main()
{
    int num, sum;
    printf("Enter a positive integer: ");
    scanf("%d",&num);
    sum=add(num);
    printf("sum=%d", sum);
}
```

Output:

```
Enter a positive integer: 5
15
```

Explanation:

- Here, add() function is invoked from the same function.
- If n is not equal to 0 then, the function calls itself passing argument 1 less than the previous argument it was called with.
- Suppose, n is 5 initially. Then, during next function calls, 4 is passed to function and the value of argument decreases by 1 in each recursive call.
- When, n becomes equal to 0, the value of n is returned which is the sum numbers from 5 to 1.
- Observe the following for better visualization of recursion in this example:

```
add(5)
=5+ add(4)
=5+4+ add(3)
=5+4+3+ add(2)
=5+4+3+2+ add(1)
=5+4+3+2+1+ add(0)
=5+4+3+2+1+0
=5+4+3+2+1
=5+4+3+3
=5+4+6
=5+10
=15
```

- Every recursive function must be provided with a way to end the recursion. In this example when, n is equal to 0, there is no recursive call and recursion ends.



PROGRAMMING IN C AND DATA STRUCTURES

STORAGE CLASS

- Every variable in C program has two properties: type and storage class.
- Type refers to the data type of variable whether it is character or integer or floating-point value etc.
- And storage class determines how long it stays in existence.
- There are 4 types of storage class:
 - 1) automatic(or local)
 - 2) external(or global)
 - 3) static
 - 4) register

AUTOMATIC VARIABLE

- Variables declared inside the function body are automatic by default.
- These variables are also known as local variables as they are local to the function and doesn't have meaning outside that function.
- For example:
 int a,b; or auto int a,b;
- Since, variable inside a function is automatic by default, keyword auto are rarely used.

EXTERNAL VARIABLES

- External variable can be accessed by any function. They are also known as global variables.
- Variables declared outside every function are external variables.
- In case of large program, containing more than one file, if the global variable is declared in file 1 and that variable is used in file 2 then, compiler will show error.
- To solve this problem, keyword extern is used in file 2 to indicate that, the variable specified is global variable and declared in another file.
- Example: Program to demonstrate working local and global variables.

```
#include <stdio.h>
int z;                // z is a global variable

void display()
{
    printf("\n sum=%d", z);
}

void main()
{
    // a and b are local variables
    auto int a, b;    //or int a, b;
    printf("Enters two number to add: ");
    scanf("%d %d", &a, &b);
    z=a+b;
    display();
}
```

Output:

```
Enters two number to add: 2 3
sum=5
```

REGISTER VARIABLES

- Register variables are similar to automatic variable and exists inside that particular function only.
- For example:
 register int z;
 where z is register variable
- If the compiler encounters register variable, it tries to store variable in microprocessor's register rather than memory.
- Value stored in register are much faster to access than that of memory.
- In case of larger program, variables that are used in loops and function parameters are declared register variables.



PROGRAMMING IN C AND DATA STRUCTURES

Example: Program to illustrate use of register variable.

```
#include <stdio.h>
void main()
{
    int a, b;
    register int z;
    printf("Enter two number to add \n");
    scanf("%d %d", &a, &b);
    z=a+b;
    printf("\n sum=%d", z);
}
```

Output:

```
Enter two number to add
2 3
sum=5
```

STATIC VARIABLE

- The value of static variable persists until the end of the program.
- For example:

```
static int c;
```

where c is a static variable

- Example: Program to demonstrate working of static variable.

```
#include <stdio.h>
void Check()
{
    static int c=0;
    printf("%d \t",c);
    c=c+5;
}
```

```
int main()
{
    Check();
    Check();
    Check();
}
```

Output:

```
0    5    10
```

Explanation:

- During first function call, it will display 0.
- Then, during second function call, variable c will not be initialized to 0 again, as it is static variable.
- So, 5 is displayed in second function call and 10 in third call.



MODULE IV: STRUCTURES

INTRODUCTION

- Suppose, you want to store the information about person about his name, citizenship number and salary.
- You can create these information separately but, better approach will be collection of these information under single name because all these information are related to person.
- For this purpose, structure can be used.

STRUCTURES

- Structure is a collection of elements of different data type.
- The syntax is shown below:

```
struct structure_name
{
    data_type member1;
    data_type member2;
    data_type member3;
};
```
- The variables that are used to store the data are called members of the structure.
- We can create the structure for a person as shown below:

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

- This declaration above creates the derived data type struct person.



PROGRAMMING IN C AND DATA STRUCTURES

Structure Variable Declaration

- A structure can be declared using 2 methods:

1) Tagged Structure

- When a structure is defined, it creates a user-defined type but, no storage is allocated. For the above structure of person, variable can be declared as:

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

Inside main function:

```
struct person p1, p2;
```

- Here the above statement declares that the variables p1 and p2 are variables of type struct person.
- Once the structure variable is declared, the compiler allocates memory for the structure variables.
- The size of the memory allocated is the sum of size of individual members.

2) Type Defined Structure

- Another way of creating structure variable using the keyword typedef is:

```
typedef struct person
{
    char name[50];
    int cit_no;
    float salary;
}EMP;
```

Inside main function:

```
EMP p1 ,p2 ;
```

- Here the above statement declares that the variables p1 and p2 are variables of type EMP(which is of type as struct person)..

Structure Variable Initialization

- For ex:

```
struct person
{
    char name[50];
    int cit_no;
    float salary;
};
struct person p1={"ram", 24, 23000};
```

Accessing Members of a Structure

- Member operator(.) can be used for accessing members of a structure.
- Any member of a structure can be accessed as:
structure_variable_name.member_name
- By specifying p1.name, we can access the name 'ram'.
By specifying p1.salary, we can access the value '23000'.
- The various values can be accessed and printed as shown below:
printf("%s", p1.name);
printf("%f", p1.salary);
- We can read the members of a structure as shown below:
gets(p1.name); or scanf("%s", p1.name);
scanf("%f",&p1.salary);.



PROGRAMMING IN C AND DATA STRUCTURES

STRUCTURES WITHIN STRUCTURES

- For ex:

```
struct dob          //dob = date of birth
{
    int day;
    int month;
    int year;
};

struct person
{
    char name[50];
    int cit_no;
    float salary;
    struct dob d1;
};
```

Inside main:

```
struct person p1 ,p2 ;
```

- Suppose you want to access month for p1 structure-variable then, structure-member p1.d1.month is used.

SELF REFERENTIAL STRUCTURES

- A self-referential structure is one in which one or more of its components is a pointer to itself.
- These require dynamic storage management functions (malloc & free) to explicitly obtain and release memory.

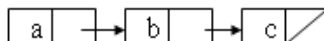
```
typedef struct
{
    char data;
    struct list *link;    //list is a pointer to a list structure
}list;
```

- Consider three structures and values assigned to their respective fields:

```
list item1,item2,item3;
item1.data='a';
item2.data='b';
item3.data='c';
item1.link=item2.link=item3.link=NULL;
```

- We can attach these structures together as follows:

```
item1.link=&item2;
item2.link=&item3;
```





PROGRAMMING IN C AND DATA STRUCTURES

STRUCTURES AND FUNCTIONS

- In C, structure can be passed to functions by 2 methods:
 - 1) Passing by value (passing actual value as argument)
 - 2) Passing by reference (passing address of an argument)

Passing Structure by Value

- A structure-variable can be passed to the function as an argument as normal variable.
- If structure is passed by value, change made in structure-variable in function-definition does not reflect in original structure variable in calling-function.
- Example: Write a C program to create a structure student, containing name and roll. Ask user the name and roll of a student in main function. Pass this structure to a function and display the information in that function.

```
#include<stdio.h>
struct student
{
    char name[50];
    int roll;
};

void Display(struct student stu)
{
    printf("Your Name: %s \n",stu.name);
    printf("Your Roll: %d",stu.roll);
}

void main()
{
    struct student s1;
    printf("Enter student's name: ");
    scanf("%s", &s1.name);
    printf("Enter roll number: ");
    scanf("%d", &s1.roll);
    Display(s1);      //passing structure variable s1 as argument
}
```

Output:

```
Enter student's name: rama
Enter roll number: 149
Your Name: rama
Your Roll: 149
```



PROGRAMMING IN C AND DATA STRUCTURES

Passing Structure by Reference

- The address location of structure-variable is passed to function while passing it by reference.
- If structure is passed by reference, change made in structure-variable in function-definition reflects in original structure variable in the calling-function.
- Write a C program to add two complex numbers entered by user. To solve this program, make a structure. Pass two structure variable (containing distance in real and imaginary part) to add function by reference and display the result in main function without returning it.

```
#include <stdio.h>
struct comp
{
    int real;
    float img;
};

void Add(struct comp c1, struct comp c2, struct comp *c3)
{
    // Adding complex numbers c1 and d2 and storing it in c3
    c3->real=c1.real+c2.real;
    c3->img=c1.img+c2.img;
}

void main()
{
    struct comp c1, c2, c3;
    printf("First complex number \n");
    printf("Enter real part: ");
    scanf("%d",&c1.real);
    printf("Enter imaginary part: ");
    scanf("%f",&c1.img);
    printf("Second complex number \n");
    printf("Enter real part: ");
    scanf("%d",&c2.real);
    printf("Enter imaginary part: ");
    scanf("%f",&c2.img);
    Add(c1, c2, &c3);

    //passing structure variables c1 and c2 by value whereas passing
    // structure variable c3 by reference
    printf("Sum of 2 complex numbers = %d + i %d",c3.real, c3.img);
}
```

Output:

```
First complex number
Enter real part: 3
Enter imaginary part: 4
Second complex number
Enter real part: 2
Enter imaginary part: 5
Sum of 2 complex numbers = 5 + i 9
```




MODULE IV(CONT.): FILE MANAGEMENT

INTRODUCTION

- A computer file is used to store data in digital format like plain text, image data or any other content.
- Computer files can be considered as the digital counterpart of paper documents, which traditionally are kept in office.
- While doing programming, you keep your source code in text files with different extensions.
- For example, C programming files have .c extension, Java programming files have .java extension.

File Operations

- 1) Creating a new file
- 2) Opening an existing file
- 3) Reading from and writing information to a file
- 4) Closing a file

DEFINING, OPENING AND CLOSING OF FILES

Defining a File

- While working with file, you need to declare a pointer of type 'FILE'. This declaration is needed for communication between file and program.

```
FILE *ptr;
```

Opening a File

- fopen() function can be used
 - to create a new file or
 - to open an existing file
- This function will initialize an object of the type FILE, which contains all the information necessary to control the stream.
- The syntax is shown below:

```
FILE *fopen( const char *filename, const char *access_mode );
```

where filename is string literal, which you will use to name your file.

access_mode can have one of the following values:

Mode Description

r	Opens an existing text file for reading purpose
w	Opens a text file for writing, if it does not exist then a new file is created
a	Opens a text file for writing in appending mode
r+	Opens a text file for reading and writing both

Closing a File

- The file should be closed after reading/writing of a file.
- fclose() function can be used to close a file.
- The syntax is shown below:

```
int fclose(FILE *fp);
```
- The fclose() function returns zero on success or
returns EOF(special character) if there is an error



PROGRAMMING IN C AND DATA STRUCTURES

INPUT AND OUTPUT OPERATIONS

Writing a File: fputc() fputs() & fprintf()

- You can use the following function to write individual characters to a stream:
`int fputc(int c, FILE *fp);`
- The function fputc() writes the character value of argument c to the output stream referenced by fp.
- This function returns the written character on success; otherwise returns EOF if there is an error.
- You can use the following function to write a null-terminated string to a stream:
`int fputs(const char *s, FILE *fp);`
- The function fputs() writes the string s into the file referenced by fp.
- This function returns a non-negative value on success; otherwise returns EOF if there is an error.
- You can use following function to write a string into a file:
`int fprintf(FILE *fp, const char *format, ...)`

```
#include <stdio.h>
void main()
{
    FILE *fp;
    fp = fopen("/tc/bin/test.txt", "w+");
    fprintf(fp, "This is testing for fprintf...\n");
    fputs("This is testing for fputs...\n", fp);
    fclose(fp);
}
```

- When the above code is compiled and executed, it creates a new file test.txt in /tc/bin directory and writes two lines using two different functions.

Reading a File: fgetc() fgets() & fscanf()

- You can use the following function to read a text file character by character:
`int fgetc(FILE * fp);`
- The fgetc() function reads a character from the input file referenced by fp.
- This function returns the character being read on success; otherwise returns EOF if there is an error.
- You can use the following function to read a string from a stream:
`char *fgets(char *buf, int n, FILE *fp);`
- The functions fgets() reads up to n-1 characters from the input stream referenced by fp.
- It copies the read string into the buffer buf, appending a null character to terminate the string.
- If this function encounters a newline character '\n', then it returns only the characters read up to that point including new line character.
- You can also use `int fscanf(FILE *fp, const char *format, ...)` function to read strings from a file but it stops reading after the first space character encounters.

```
#include <stdio.h>
void main()
{
    FILE *fp;
    char buff[255];
    fp = fopen("/tc/bin/test.txt", "r");
    fscanf(fp, "%s", buff);
    printf("1 : %s \n", buff );
    fgets(buff, 255, (FILE*)fp);
    printf("2: %s \n", buff );
    fgets(buff, 255, (FILE*)fp);
    printf("3: %s ", buff );
    fclose(fp);
}
```

Output:

```
1 : This
2: is testing for fprintf...
3: This is testing for fputs...
```

- When the code is compiled and executed, it reads the file 'test.txt' in /tc/bin/ directory and produces the above result.



MODULE V: POINTERS & PREPROCESSORS

INTRODUCTION

- 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 will print the address of the variables defined:

```
#include<stdio.h>
void main()
{
    int var1;
    char var2;
    printf("Address of var1 variable: %d \n", &var1 );
    printf("Address of var2 variable: %d", &var2 );
    return 0;
}
```

Output:

Address of var1 variable: 1266

Address of var2 variable: 1268

POINTER

- A pointer is a variable which holds address of another variable or a memory-location.
- For ex:

c=300;

pc=&c;

Here pc is a pointer; it can hold the address of variable c
& is called reference operator



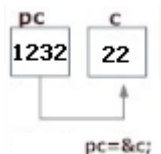
PROGRAMMING IN C AND DATA STRUCTURES

DECLARATION OF POINTER VARIABLE

- Dereference operator(*) are used for defining pointer-variable.
- The syntax is shown below:
data_type *ptr_var_name;
- For ex:
int *a; // a as pointer variable of type int
float *c; // c as pointer variable of type float
- Steps to access data through pointers:
 - 1) Declare a data-variable ex: int c;
 - 2) Declare a pointer-variable ex: int *pc;
 - 3) Initialize a pointer-variable ex: pc=&c;
 - 4) Access data using pointer-variable ex: printf("%d",*pc);
- Example: Program to illustrate working of pointers.

```
#include<stdio.h>
void main()
{
    int *pc;
    int c;
    c=22;
    printf("Address of c: %d \n", &c);
    printf("Value of c: %d \n", c);
    pc=&c;
    printf("Address of pointer pc: %d \n", pc);
    printf("Content of pointer pc: %d",*pc);
}
```

Output:
Address of c: 1232
Value of c: 22
Address of pointer pc: 1232
Content of pointer pc: 22



Explanation of Program and Figure

- Code int* pc; creates a pointer pc and code int c; creates normal variable c
- Code c=22; makes the value of c equal to 22, i.e., 22 is stored in the memory-location of variable c.
- Code pc=&c; makes pointer, point to address of c. Note that, &c is the address of variable c (because c is normal variable) and pc is the address of pc (because pc is the pointer-variable).
- Since the address of pc and address of c is same, *pc will be equal to the value of c.

NULL POINTER

- A NULL pointer is defined as a special pointer value that points to '\0'(nowhere) in the memory. In other words, NULL pointer does not point to any part of the memory.
- For ex:
int *p=NULL;



PROGRAMMING IN C AND DATA STRUCTURES

POINTERS AND ARRAYS

- Consider an array:
`int arr[4];`
- The above code can be pictorially represented as shown below:

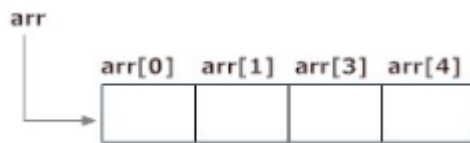


Figure: Array as Pointer

- The name of the array always points to the first element of an array.
- Here, address of first element of an array is `&arr[0]`.
- Also, `arr` represents the address of the pointer where it is pointing. Hence, `&arr[0]` is equivalent to `arr`.
- Also, value inside the address `&arr[0]` and address `arr` are equal. Value in address `&arr[0]` is `arr[0]` and value in address `arr` is `*arr`. Hence, `arr[0]` is equivalent to `*arr`.
- Similarly,
 - `&a[1]` is equivalent to `(a+1)` AND, `a[1]` is equivalent to `*(a+1)`.
 - `&a[2]` is equivalent to `(a+2)` AND, `a[2]` is equivalent to `*(a+2)`.
 - `&a[3]` is equivalent to `(a+1)` AND, `a[3]` is equivalent to `*(a+3)`.
 - .
 - .
 - `&a[i]` is equivalent to `(a+i)` AND, `a[i]` is equivalent to `*(a+i)`.
- You can declare an array and can use pointer to alter the data of an array.
- Example: Program to access elements of an array using pointer.

```
#include<stdio.h>
void main()
{
    int data[5], i;
    printf("Enter elements: ");
    for(i=0;i<5;++i)
        scanf("%d", data+i);
    printf("You entered: ");
    for(i=0;i<5;++i)
        printf("%d ",*(data+i) );
}
```

Output:

```
Enter elements: 1 2 3 5 4
You entered: 1 2 3 5 4
```

- Example: Program to find sum of all the elements of an array using pointers.

```
#include<stdio.h>
void main()
{
    int i, a[10], n, sum=0;
    printf("Enter the size of the array:");
    scanf("%d", &n);
    printf("Enter the elements into the array: ");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    for(i=0;i<n;i++)
        sum+=*(a+i);
    printf("Sum --> %d",sum);
}
```

Output:

```
Enter the size of the array:5
Enter the elements into the array: 2 4 6 10 15
Sum --> 37
```



PROGRAMMING IN C AND DATA STRUCTURES

POINTERS AND FUNCTIONS

- When, argument is passed using pointer, address of the memory-location is passed instead of value.
- Example: Program to swap 2 number using call by reference.

```
#include<stdio.h>

void swap(int *a,int *b)
{ // pointer a and b points to address of num1 and num2 respectively
  int temp;
  temp=*a;
  *a=*b;
  *b=temp;
}

void main()
{
  int num1=5,num2=10;
  swap(&num1, &num2); //address of num1 & num2 is passed to swap function
  printf("Number1 = %d \n",num1);
  printf("Number2 = %d",num2);
}
```

Output:

Number1 = 10
Number2 = 5

Explanation

- The address of memory-location num1 and num2 are passed to function and the pointers *a and *b accept those values.
- So, the pointer a and b points to address of num1 and num2 respectively.
- When, the value of pointer is changed, the value in memory-location also changed correspondingly.
- Hence, change made to *a and *b was reflected in num1 and num2 in main function.
- This technique is known as call by reference.



PROGRAMMING IN C AND DATA STRUCTURES

POINTER ARITHMETIC

- As you know, pointer is an address, which is a numeric value.
- Therefore, you can perform arithmetic operations on a pointer just as you can a numeric value.
- There are 4 arithmetic operators that can be used on pointers: ++, --, +, and -
- To understand pointer arithmetic, let us consider that ptr is an integer pointer which points to the address 1000.
- Assuming 16-bit integers, let us perform the following arithmetic operation on the pointer:
ptr++
- Now, after the above operation, the ptr will point to the location 1002 because each time ptr is incremented, it will point to the next integer location which is 2 bytes next to the current location.

Incrementing a Pointer

- We prefer using a pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because it is a constant pointer.
- Example: Program to increment the variable pointer to access each succeeding element of the array.

```
#include<stdio.h>
void main()
{
    int var[] = {10, 100, 200};
    int i, *ptr;
    ptr = var;
    for ( i = 0; i < 3; i++)
    {
        printf("Address of var[%d] = %x \n", i, ptr );
        printf("Value of var[%d] = %d \n", i, *ptr );
        ptr++;           //move to the next location
    }
}
```

Output:

```
Address of var[0] = 1130
Value of var[0] = 10
Address of var[1] = 1132
Value of var[1] = 100
Address of var[2] = 1134
Value of var[2] = 200
```



PROGRAMMING IN C AND DATA STRUCTURES

POINTERS TO POINTERS

- A variable which contains address of a pointer-variable is called pointer to a pointer.



- A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name.
- For example, following is the declaration to declare a pointer to a pointer of type int:
`int **var;`
- When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example:

```
#include<stdio.h>
void main()
{
    int var;
    int *ptr;
    int **pptr;
    var = 3000;           // take the address of var
    ptr = &var;           // take the address of ptr using address of operator &
    pptr = &ptr;          // take the value using pptr
    printf("Value of var = %d \n", var );
    printf("Value available at *ptr = %d \n", *ptr );
    printf("Value available at **pptr = %d ", **pptr);
    return 0;
}
```

Output:

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```




PROGRAMMING IN C AND DATA STRUCTURES

PREPROCESSOR

- A preprocessor is not part of the compiler, but is a separate step in the compilation process.
- In simplistic terms, a preprocessor is just a text substitution tool and they instruct compiler to do required pre-processing before actual compilation.
- All preprocessor commands begin with a pound symbol(#).
- List of pre-processor directives:

#include

This is used insert a particular header from another file.

#define, #undef

These are used to define and un-define conditional compilation symbols.

#if, #elif, #else, #endif

These are used to conditionally skip sections of source code.

#error

This is used to issue errors for the current program.

Use of #include

- Let us consider very common preprocessing directive as below:
#include <stdio.h>
- Here, "stdio.h" is a header file and the preprocessor replace the above line with the contents of header file.
- Example: Program to illustrate use of #include

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

Output:

VTUNOTESBYSRI

Use of #define

- Analyze following examples to understand this directive.
#define PI 3.1415
- The string 3.1415 is replaced in every occurrence of symbolic constant PI,
- Example: Program to find area of a circle using #define.

```
#include <stdio.h>
#define PI 3.1415
int main()
{
    int radius;
    float area;
    printf("Enter the radius: ");
    scanf("%d", &radius);
    area=PI*radius*radius;
    printf("Area=%.2f",area);
    return 0;
}
```

Output:

Enter the radius: 3
Area=28.27



PROGRAMMING IN C AND DATA STRUCTURES

Use of #if, #elif, #else and #endif

- The preprocessor directives #if, #elif, #else and #endif allows to conditionally compile a block of code based on predefined symbols.
- Example: Program to illustrate this concept.

```
#include<stdio.h>
#define MAX 100
void main()
{
    #if(MAX)
        printf("MAX is defined");
    #else
        printf ("MAX is not defined");
    #endif
}
```

Output:

MAX is defined

Use of #error

- The #error directives allow instructing the compiler to generate an error.
- For example, we wish to issue a warning when a symbol is defined in the current project.
- Example: Program to illustrate use of #error.

```
#include<stdio.h>
#define MAX 100
void main()
{
    #if(MAX)
        #error: MAX is defined by me!!!
        . .
        . .
        . .
    #endif
}
```

Output:

#error: MAX is defined by me!!!

Use of #undef

- Consider an example shown below:
#undef FILE_SIZE
#define FILE_SIZE 42
- This tells the preprocessor to undefine existing FILE_SIZE and define it as 42.



PROGRAMMING IN C AND DATA STRUCTURES

MEMORY ALLOCATION FUNCTIONS

- There are 2 types:

1) Static Memory Allocation:

- If memory-space to be allocated for various variables is decided during compilation-time itself, then the memory-space cannot be expanded to accommodate more data or cannot be reduced to accommodate less data.
- In this technique, once the size of the memory-space to be allocated is fixed, it cannot be altered during execution-time. This is called static memory allocation.
- For ex,
 int a[5];

2) Dynamic Memory Allocation

- Dynamic memory allocation is the process of allocating memory-space during execution-time i.e. run time.
- If there is an unpredictable storage requirement, then the dynamic allocation technique is used.
- This allocation technique uses predefined functions to allocate and release memory for data during execution-time.
- There are 4 library functions for dynamic memory allocation:
 - 1) malloc()
 - 2) calloc()
 - 3) free()
 - 4) realloc()
- These library functions are defined under "stdlib.h"



PROGRAMMING IN C AND DATA STRUCTURES

malloc()

- The name malloc stands for "memory allocation".
- This function is used to allocate the requirement memory-space during execution-time.
- The syntax is shown below:
 data_type *p;
 p=(data_type*)malloc(size);
 here p is pointer variable
 data_type can be int, float or char
 size is number of bytes to be allocated
- If memory is successfully allocated, then address of the first byte of allocated space is returned.
 If memory allocation fails, then NULL is returned.
- For ex:
 ptr=(int*)malloc(100*sizeof(int));
- The above statement will allocate 200 bytes assuming sizeof(int)=2 bytes.
- Example: Program to find sum of n elements entered by user. Allocate memory dynamically using malloc() function.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int n, i, *ptr, sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int)); //memory allocated using malloc
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d ",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
}
```

Output:

```
Enter number of elements: 3
Enter elements of array: 2 5 1
Sum= 8
```



PROGRAMMING IN C AND DATA STRUCTURES

calloc()

- The name calloc stands for "contiguous allocation".
- This function is used to allocate the required memory-size during execution-time and at the same time, automatically initialize memory with 0's.
- The syntax is shown below:
data_type *p;
p=(data_type*)calloc(n,size);
- If memory is successfully allocated, then address of the first byte of allocated space is returned.
If memory allocation fails, then NULL is returned.
- The allocated memory is initialized automatically to 0's.
- For ex:
ptr=(int*)calloc(25,sizeof(int));
- The above statement allocates contiguous space in memory for an array of 25 elements each of size of int, i.e., 2 bytes.
- Example: Program to find sum of n elements entered by user. Allocate memory dynamically using calloc() function.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d ",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
}
```

Output:

```
Enter number of elements: 3
Enter elements of array: 2 5 1
Sum= 8
```

free()

- Dynamically allocated memory with either calloc() or malloc() does not get return on its own. The programmer must use free() explicitly to release space.
- The syntax is shown below:
free(ptr);
- This statement causes the space in memory pointed by ptr to be deallocated.



PROGRAMMING IN C AND DATA STRUCTURES

realloc()

- If the previously allocated memory is insufficient or more than sufficient. Then, you can change memory-size previously allocated using realloc().
- The syntax is shown below:
ptr=(data_type*)realloc(ptr,newsize);
- Example: Program to illustrate working of realloc().

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int *ptr, i, n1, n2;
    printf("Enter size of array: ");
    scanf("%d",&n1);
    ptr=(int*)malloc(n1*sizeof(int));
    printf("Address of previously allocated memory: ");
    for(i=0;i<n1;i++)
        printf("%u \n", ptr+i);
    printf("\n Enter new size of array: ");
    scanf("%d",&n2);
    ptr=realloc(ptr,n2);
    printf("Address of newly allocated memory: ");
    for(i=0;i<n2;i++)
        printf("%u \n", ptr+i);
}
```

Output:

```
Enter size of array: 3
Address of previously allocated memory:
1022
1024
1026
Enter new size of array: 5
Address of newly allocated memory:
1022
1024
1026
1028
1030
```



MODULE V(CONT.): INTRODUCTION TO DATA STRUCTURES

PRIMITIVE AND NON-PRIMITIVE DATA TYPES

- Data type specifies the type of data stored in a variable. The data type can be classified into two types: 1) Primitive data type and 2) Non-Primitive data type

Primitive Data Type

- The primitive data types are the basic data types that are available in most of the programming languages.
- The primitive data types are used to represent single values.
 - Integer: This is used to represent a number without decimal point.
Eg: 12, 90
 - Float: This is used to represent a number with decimal point.
Eg: 45.1, 67.3
 - Character: This is used to represent single character
Eg: 'C', 'a'
 - String: This is used to represent group of characters.
Eg: "M.S.P.V.L Polytechnic College"
 - Boolean: This is used to represent logical values either true or false.

Non Primitive Data Type

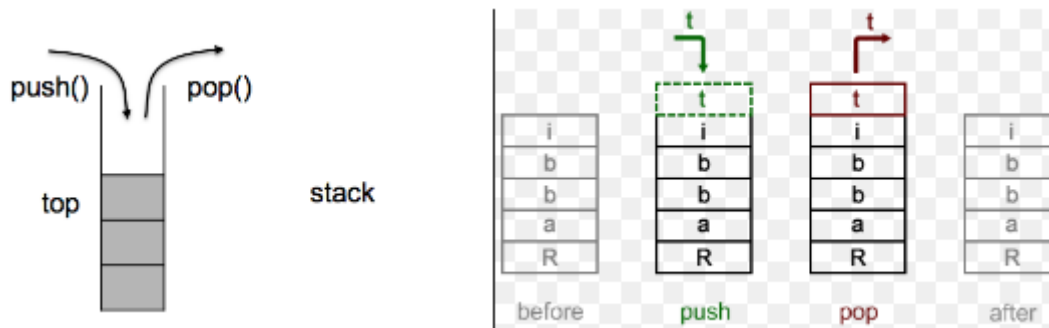
- The data types that are derived from primary data types are known as non-Primitive data types.
- These datatypes are used to store group of values.
- The non-primitive data types are
 - Arrays
 - Structure
 - Stacks
 - Linked list
 - Queue
 - Binary tree



PROGRAMMING IN C AND DATA STRUCTURES

STACKS

- A stack is a special type of data structure where elements are inserted from one end and elements are deleted from the same end.
- Using this approach, the Last element Inserted is the First element to be deleted Out, and hence, stack is also called LIFO data structure.
- The various operations performed on stack:
 - Insert: An element is inserted from top end. Insertion operation is called push operation.
 - Delete: An element is deleted from top end. Deletion operation is called pop operation.
 - Overflow: Check whether the stack is full or not.
 - Underflow: Check whether the stack is empty or not.
- This can be pictorially represented as shown below:



APPLICATIONS OF STACK

- 1) Conversion of expressions: The compiler converts the infix expressions into postfix expressions using stack.
- 2) Evaluation of expression: An arithmetic expression represented in the form of either postfix or prefix can be easily evaluated using stack.
- 3) Recursion: A function which calls itself is called recursive function.
- 4) Other applications: To find whether the string is a palindrome, to check whether a given expression is valid or not.

**PROGRAMMING IN C AND DATA STRUCTURES****Program to implement a stack using array.**

```
#include <stdio.h>
#define MAXSIZE 5
int stack[MAXSIZE];
int top;

void push()          // Function to add an element to the stack
{
    int num;

    if (top==(MAXSIZE-1))
    {
        printf ("Error: Overflow ");
    }
    else
    {
        printf ("Enter the element to be pushed \n");
        scanf ("%d", &num);
        top = top + 1;
        stack[top] = num;
    }
}

void pop()           //Function to delete an element from the stack
{
    int num;
    if (top== -1)
    {
        printf ("Error: Stack Empty\n");
    }
    else
    {
        num = stack[top];
        printf ("popped element is = %d \n", num);
        top=top-1;
    }
}

void display()       //Function to display the status of the stack
{
    int i;
    if (top == -1)
    {
        printf ("Error: Stack Empty");
    }
    else
    {
        printf ("\n Items in Stack \n");
        for (i = top; i >= 0; i--)
        {
            printf ("%d \n", stack[i]);
        }
    }
}
```

**PROGRAMMING IN C AND DATA STRUCTURES**

```
void main()
{
    int element, choice;
    top = -1;
    while (1)
    {
        printf ("1. PUSH \n");
        printf ("2. POP \n");
        printf ("3. DISPLAY \n");
        printf ("4. EXIT \n");
        printf ("Enter your choice \n");
        scanf ("%d", &choice);
        switch (choice)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: return;
        }
    }
}
```

Output:

```
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice
1
Enter the element to be pushed
11
```

```
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice
1
Enter the element to be pushed
22
```

```
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice
1
Enter the element to be pushed
33
```

**PROGRAMMING IN C AND DATA STRUCTURES**

```
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice
3
Items in Stack
33
22
11

1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice
2
popped element is = 33

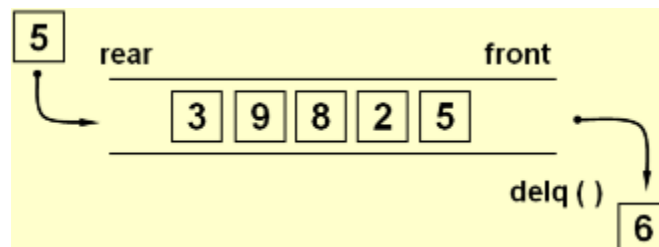
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice
2
popped element is = 22
```



PROGRAMMING IN C AND DATA STRUCTURES

QUEUES

- A queue is a special type of data structure where elements are inserted from one end and elements are deleted from the other end.
- The end at which new elements are added is called the rear and the end from which elements are deleted is called the front.
- The first element inserted is the first element to be deleted out, and hence queue is also called FIFO data structure.
- The various operations performed on queue are
 - 1) Insert: An element is inserted from rear end.
 - 2) Delete: An element is deleted from front end.
 - 3) Overflow: If queue is full and we try to insert an item, overflow condition occurs.
 - 4) Underflow: If queue is empty and try to delete an item, underflow condition occurs.
- This can be pictorially represented as shown below:



Program to implement a queue using an array.

```
#include <stdio.h>
#define MAX 50
int queue_array[MAX], rear = - 1, front = - 1;

insert()
{
    int add_item;
    if (rear == MAX - 1)
        printf("Queue Overflow ");
    else
    {
        if (front == - 1)        //If queue is initially empty
            front = 0;
        printf("Insert the element in queue : ");
        scanf("%d", &add_item);
        rear=rear+1;
        queue_array[rear]=add_item;
    }
}    //End of insert()

delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
}    //End of delete()
```

**PROGRAMMING IN C AND DATA STRUCTURES**

```
display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}

main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1: insert();
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
            case 4: exit(1);
                    break;
            default: printf("Wrong choice \n");
        }
    }
}
```

Output:

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Insert the element in queue : 10
```

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Insert the element in queue : 15
```

**PROGRAMMING IN C AND DATA STRUCTURES**

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 1

Inset the element in queue : 20

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 1

Inset the element in queue : 30

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 2

Element deleted from queue is : 10

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 3

Queue is :

15 20 30

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 4



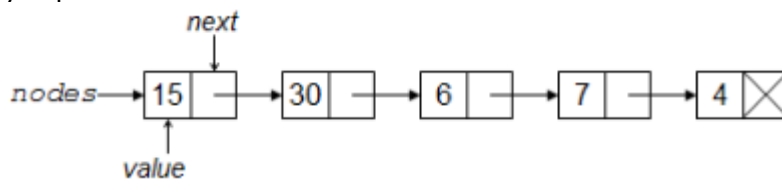
PROGRAMMING IN C AND DATA STRUCTURES

LINKED LIST

- A linked list is a data structure which is collection of zero or more nodes where each node has some information.
- Normally, node consists of 2 fields
 - 1) info field which is used to store the data or information to be manipulated
 - 2) link field which contains address of the next node.
- Different types of linked list are SLL & DLL
- Various operations of linked lists
 - 1) Inserting a node into the list
 - 2) Deleting a node from the list
 - 3) Search in a list
 - 4) Display the contents of list

SINGLY LINKED LIST

- This is a collection of zero or more nodes where each node has two or more fields and only one link field which contains address of the next node.
- This can be pictorially represented as shown below:



- In C, this can be represented as shown below:

```
struct node    // Node Declaration
{
    int value;
    struct node *next;
};
typedef struct node snode;
```
- A pointer variable can be declared as shown below

```
snode *first;
```

**PROGRAMMING IN C AND DATA STRUCTURES****• Program to implement singly linked list using dynamic memory allocation**

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

struct node    // Node Declaration
{
    int value;
    struct node *next;
};

typedef struct node snode;
snode *newnode, *ptr, *prev, *temp;
snode *first = NULL, *last = NULL;

snode* create_node(int val)    // Creating Node
{
    newnode = (snode *)malloc(sizeof(snode));
    if (newnode == NULL)
    {
        printf("\nMemory was not allocated");
        return 0;
    }
    else
    {
        newnode->value = val;
        newnode->next = NULL;
        return newnode;
    }
}

void insert_node_first()    // Inserting Node at First
{
    int val;
    printf("\nEnter the value for the node:");
    scanf("%d", &val);
    newnode = create_node(val);
    if (first == last && first == NULL)
    {
        first = last = newnode;
        first->next = NULL;
        last->next = NULL;
    }
    else
    {
        temp = first;
        first = newnode;
        first->next = temp;
    }
}
```


**PROGRAMMING IN C AND DATA STRUCTURES**

```
void display() // Displays non-empty List from Beginning to End
{
    if (first == NULL)
    {
        printf(":No nodes in the list to display\n");
    }
    else
    {
        for (ptr = first; ptr != NULL; ptr = ptr->next)
        {
            printf("%d\t", ptr->value);
        }
    }
}

int main()
{
    int ch;
    char ans = 'Y';

    while (1)
    {
        printf("\n Operations on singly linked list\n");
        printf("\n 1.Insert node at first");
        printf("\n 2.Display List from Beginning to end");
        printf("\n 3.Exit\n");
        printf("\n Enter your choice: ");
        scanf("%d", &ch);

        switch (ch)
        {
            case 1:    printf("\n...Inserting node at first...\n");
                       insert_node_first();
                       break;
            case 2:    printf("\n...Displaying List From Beginning to End...\n");
                       display();
                       break;
            case 3:    printf("\n...Exiting...\n");
                       return 0;
                       break;
            default:   printf("\n...Invalid Choice...\n");
                       break;
        }
    }
    return 0;
}
```

Output:

Operations on singly linked list
1.Insert node at first
2.Delete Node from any Position
3.Exit
Enter your choice: 1
...Inserting node at first...
Enter the value for the node: 100

**PROGRAMMING IN C AND DATA STRUCTURES**

Operations on singly linked list

- 1.Insert node at first
- 2.Delete Node from any Position
- 3.Exit

Enter your choice: 1

...Inserting node at first...

Enter the value for the node: 200

Operations on singly linked list

- 1.Insert node at first
- 2.Delete Node from any Position
- 3.Exit

Enter your choice: 1

...Inserting node at first...

Enter the value for the node: 300

Operations on singly linked list

- 1.Insert node at first
- 2.Delete Node from any Position
- 3.Exit

Enter your choice: 2

...Displaying List From Beginning to End...

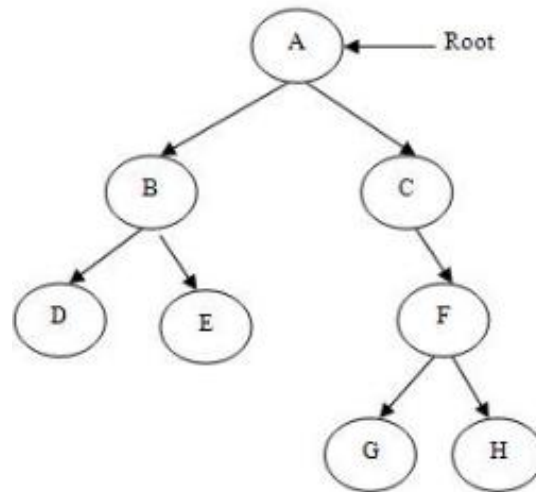
100 200 300



PROGRAMMING IN C AND DATA STRUCTURES

BINARY TREE

- A tree in which each node has either zero, one or two subtrees is called a binary tree.
- figure
- Each node consists of three fields
 - llink: contains address of left subtree
 - info: this field is used to store the actual data or information to be manipulated
 - rrlink: contains address of right subtree
- This can be pictorially represented as shown below:



BINARY TREE APPLICATIONS

- Tree traversal refers to process of visiting all nodes of a tree exactly once (Figure 5.16).
- There are 3 techniques, namely:
 - 1) Inorder traversal(LVR);
 - 2) Preorder traversal(VLR);
 - 3) Postorder traversal(LRV). (Let L= moving left, V= visiting node and R=moving right).
- In postorder, we visit a node after we have traversed its left and right subtrees.
 - In preorder, the visiting node is done before traversal of its left and right subtrees.
 - In inorder, firstly node's left subtrees is traversed, then node is visited and
- For above tree, tree traversal is as follows
 - Inorder traversal → DBEACGFH
 - Preorder traversal → ABDECFGH
 - Postorder traversal → DEBGHFCA



WORKED EXAMPLES

BASIC C PROGRAMS

PROGRAM TO PRINT A SENTENCE.

```
#include <stdio.h>
void main()
{
    printf("C Programming"); //displays the content inside quotation
    return;
}
```

Output:

C Programming

PROGRAM TO READ AND DISPLAY AN INTEGER.

```
#include <stdio.h>
void main()
{
    int num;
    printf("Enter a integer: ");
    scanf("%d",&num); // Storing a integer entered by user in variable num
    printf("You entered: %d", num);
    return;
}
```

Output:

Enter a integer: 25

You entered: 25

PROGRAM TO ADD TWO INTEGERS

```
#include <stdio.h>
void main( )
{
    int num1, num2, sum;
    printf("Enter two integers: ");
    scanf("%d %d",&num1,&num2); // Stores the two integer entered by user in
                                // variable num1 and num2
    sum=num1+num2; // Performs addition and stores it in variable sum
    printf("Sum: %d",sum); // Displays sum
    return;
}
```

Output:

Enter two integers: 12

11

Sum: 23



PROGRAMMING IN C AND DATA STRUCTURES

PROGRAM TO MULTIPLY TWO FLOATING POINT NUMBERS

```
#include <stdio.h>
void main( )
{
    float num1, num2, product;
    printf("Enter two numbers: ");
    scanf("%f %f",&num1,&num2); //Stores two floating point numbers entered
                                //by user in variable num1 and num2 respectively
    product = num1*num2; // Performs multiplication and stores it
    printf("Product: %f", product);
    return;
}
```

Output:

Enter two numbers: 2.4
1.1
Product: 2.640000

PROGRAM TO FIND QUOTIENT AND REMAINDER OF TWO INTEGERS ENTERED BY USER

```
#include <stdio.h>
void main()
{
    int dividend, divisor, quotient, remainder;
    printf("Enter dividend: ");
    scanf("%d",&dividend);
    printf("Enter divisor: ");
    scanf("%d",&divisor);
    quotient=dividend/divisor; // Computes quotient
    remainder=dividend%divisor; // Computes remainder
    printf("Quotient = %d\n",quotient);
    printf("Remainder = %d",remainder);
    return;
}
```

Output:

Enter dividend: 25
Enter divisor: 4
Quotient = 6
Remainder = 1

PROGRAM TO FIND THE AREA OF A RECTANGLE.

```
#include<stdio.h>
#include<conio.h>
main( )
{
    int l, b, area;
    printf("\nEnter length and breadth");
    scanf("%d %d", &l, &b);
    area=l*b;
    printf("\nArea of rectangle=%d", area);
}
```

Output:

Enter length and breadth
3 4
Area of rectangle=12



PROGRAMMING IN C AND DATA STRUCTURES

PROGRAM TO FIND THE AREA OF A CIRCLE.

```
#include<stdio.h>
#include<conio.h>
main( )
{
    float r, area;
    printf("enter radius");
    scanf("%f", &r);
    area=3.145*r*r;
    printf("area of circle=%f", area);
}
```

Output:

```
enter radius
4
area of circle=12.58
```

PROGRAM TO FIND THE SIMPLE INTEREST

```
#include<stdio.h>
void main()
{
    int amount, rate, time, si;

    printf("\nEnter Principal Amount : ");
    scanf("%d", &amount);

    printf("\nEnter Rate of Interest : ");
    scanf("%d", &rate);

    printf("\nEnter Period of Time : ");
    scanf("%d", &time);

    si = (amount * rate * time) / 100;
    printf("\nSimple Interest : %d", si);

    return(0);
}
```

Output:

```
Enter Principal Amount : 500
Enter Rate of interest : 5
Enter Period of Time : 2
Simple Interest : 50
```

PROGRAM TO FIND SQUARE OF A NUMBER.

```
#include<stdio.h>
main()
{
    int n, p;
    printf("enter a number");
    scanf("%d",&n);
    p=n*n;
    printf("The square of the number is %d",p);
}
```

Output:

```
enter a number
4
The square of the number is 16
```



PROGRAMMING IN C AND DATA STRUCTURES

PROGRAM TO FIND CUBE OF A NUMBER.

```
#include<stdio.h>
void main()
{
    int n, p;
    printf("enter a number");
    scanf("%d",&n);
    p=n*n*n;
    printf("The cube of the number is %d",p);
}
```

Output:

```
enter a number
4
The cube of the number is 64
```

PROGRAM TO SWAP TWO NUMBERS

```
#include <stdio.h>
void main()
{
    float a, b, temp;
    printf("Enter value of a: ");
    scanf("%f",&a);
    printf("Enter value of b: ");
    scanf("%f",&b);
    temp = a;    // Value of a is stored in variable temp
    a = b;       // Value of b is stored in variable a
    b = temp;    // Value of temp is stored in variable b
    printf("\nAfter swapping, value of a = %.2f\n", a);
    printf("After swapping, value of b = %.2f", b);
    return;
}
```

Output:

```
Enter value of a: 1.20
Enter value of b: 2.45
After swapping, value of a = 2.45
After swapping, value of b = 1.2
```

PROGRAM TO FIND RATIO OF NUMBER OF FEMALE AND MALE STUDENTS IN A CLASS USING EXPLICIT CONVERSION.

```
#include <stdio.h>
void main()
{
    int female_students = 35;
    int male_students = 25;
    float ratio;
    ratio = (float) female_students/ male_students
    printf("ratio : %f", sum );
}
```

Output:

```
Value of sum : 1.4
```



C PROGRAMS BASED ON BRANCHING

PROGRAM TO FIND THE LARGEST NUMBER OF TWO NUMBERS USING IF STATEMENT.

```
#include <stdio.h>
void main()
{
    float a, b;
    printf("Enter 2 numbers: \n ");
    scanf("%f %f", &a, &b);
    if(a>b)
        printf("Largest number = %.2f", a);
    if(b>a)
        printf("Largest number = %.2f", b);

    return;
}
```

Output:

```
Enter three numbers:
13.452
10.193
Largest number = 13.45
```

PROGRAM TO FIND THE LARGEST NUMBER AMONG THREE NUMBERS USING IF STATEMENT.

```
#include <stdio.h>
void main()
{
    float a, b, c;
    printf("Enter three numbers: ");
    scanf("%f %f %f", &a, &b, &c);
    if(a>=b && a>=c)
        printf("Largest number = %.2f", a);
    if(b>=a && b>=c)
        printf("Largest number = %.2f", b);
    if(c>=a && c>=b)
        printf("Largest number = %.2f", c);

    return;
}
```

Output:

```
Enter three numbers: 12.2
13.452
10.193
Largest number = 13.45
```


**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO CHECK WHETHER A NUMBER IS EVEN OR ODD USING IF ELSE STATEMENT.**

```
#include <stdio.h>
void main()
{
    int num;
    printf("Enter an integer you want to check: ");
    scanf("%d",&num);
    if((num%2)==0)    // Checking whether remainder is 0 or not.
        printf("%d is even.",num);
    else
        printf("%d is odd.",num);
}
```

Output:

Enter an integer you want to check: 25
25 is odd.

PROGRAM TO CHECK WHETHER A CHARACTER IS AN ALPHABET OR NOT USING IF ELSE STATEMENT.

```
#include <stdio.h>
void main()
{
    char c;
    printf("Enter a character: ");
    scanf("%c",&c);
    if( (c>='a' && c<='z') || (c>='A' && c<='Z'))
        printf("%c is an alphabet.",c);
    else
        printf("%c is not an alphabet.",c);
}
```

Output:

Enter a character: *
* is not an alphabet

PROGRAM TO CHECK VOWEL OR CONSONANT USING IF ELSE STATEMENT.

```
#include <stdio.h>
void main()
{
    char c;
    printf("Enter an alphabet: ");
    scanf("%c",&c);
    if(c=='a' || c=='e' || c=='i' || c=='o' || c=='u')
        printf("%c is a vowel.",c);
    else
        printf("%c is a consonant.",c);
    return;
}
```

Output:

Enter an alphabet: i
i is a vowel.

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO CHECK LEAP YEAR USING NESTED IF ELSE STATEMENT.**

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

Output:

Enter year: 1900
1900 is not a leap year.

PROGRAM TO CHECK WHETHER A NUMBER IS POSITIVE OR NEGATIVE USING NESTED IF ELSE STATEMENT.

```
#include <stdio.h>
void main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num<=0)
    {
        if (num==0)
            printf("You entered zero.");
        else
            printf("%d is negative.", num);
    }
    else
        printf("%d is positive.", num);
}
```

Output:

Enter a number: -7
-7 is negative

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO DISPLAY GRADE BASED ON THE MARKS USING ELSE IF LADDER.**

```
#include <stdio.h>
void main(void)
{
    int num;
    printf("Enter your mark: ");
    scanf("%d", &num);
    if(num>=80)
    {
        printf(" You got A grade");
    }
    else if(num>=60)
    {
        printf(" You got B grade");
    }
    else if(num>=40)
    {
        printf(" You got C grade");
    }
    else if(num<40)
    {
        printf(" You Failed in this exam");
    }
}
```

Output:

Enter your mark: 89
You got A grade

PROGRAM TO FIND ALL ROOTS OF A QUADRATIC EQUATION USING ELSE IF LADDER.

```
#include <stdio.h>
#include <math.h> // This is needed to use sqrt() function.
void main()
{
    float a, b, c, determinant, r1,r2, real, imag;
    printf("Enter coefficients a, b and c: ");
    scanf("%f%f%f",&a,&b,&c);
    determinant=b*b-4*a*c;
    if (determinant>0)
    {
        r1= (-b+sqrt(determinant))/(2*a);
        r2= (-b-sqrt(determinant))/(2*a);
        printf("Roots are: %.2f and %.2f",r1 , r2);
    }
    else if (determinant==0)
    {
        r1 = r2 = -b/(2*a);
        printf("Roots are: %.2f and %.2f", r1, r2);
    }
    else
    {
        real= -b/(2*a);
        imag = sqrt(-determinant)/(2*a);
        printf("Roots are: %.2f+%.2fi and %.2f-%.2fi",real,imag,real, imag);
    }
}
```

Output:

Enter coefficients a, b and c: 2.3 4 5.6
Roots are: -0.87+1.30i and -0.87-1.30i



C PROGRAMS BASED ON LOOPING

PROGRAM TO PRINT FIRST 10 NATURAL NUMBERS USING FOR LOOP.

```
#include <stdio.h>
void main()
{
    int num;
    for (num = 1; num <= 10; num++)
    {
        printf(" %d  ", num);
    }
}
```

Output:

1 2 3 4 5 6 7 8 9 10

PROGRAM TO PRINT FIRST 10 NATURAL NUMBERS USING WHILE LOOP.

```
#include <stdio.h>
void main()
{
    int num, sum = 0;
    num = 1
    while(num <= 10)
    {
        printf(" %d  ", num);
        num++
    }
}
```

Output:

1 2 3 4 5 6 7 8 9 10

PROGRAM TO PRINT FIRST 10 NATURAL NUMBERS USING DO-WHILE LOOP.

```
#include <stdio.h>
void main()
{
    int num, sum = 0;
    num = 1
    do
    {
        printf(" %d  ", num);
        num++
    }while(num <= 10);
}
```

Output:

1 2 3 4 5 6 7 8 9 10

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO PRINT SUM OF FIRST 50 NATURAL NUMBERS USING FOR LOOP.**

```
#include <stdio.h>
void main()
{
    int num, sum = 0;
    for (num = 1; num <= 10; num++)
    {
        sum = sum + num;
    }
    printf("Sum = %4d\n", sum);
}
```

Output:

Sum = 1275

PROGRAM TO PRINT SUM OF FIRST 50 NATURAL NUMBERS USING WHILE LOOP.

```
#include <stdio.h>
void main()
{
    int num, sum = 0;
    num = 1
    while(num <= 50)
    {
        sum = sum + num;
        num++;
    }
    printf("Sum = %4d\n", sum);
}
```

Output:

Sum = 1275

PROGRAM TO PRINT SUM OF FIRST 50 NATURAL NUMBERS USING DO-WHILE LOOP.

```
#include <stdio.h>
void main()
{
    int num, sum = 0;
    num = 1

    do
    {
        sum = sum + num;
        num++;
    }while(num <= 50);

    printf("Sum = %4d\n", sum);
}
```

Output:

Sum = 1275

**PROGRAMMING IN C AND DATA STRUCTURES**

PROGRAM TO FIND THE SUM OF FIRST N NATURAL NUMBERS WHERE N IS ENTERED BY USER. NOTE: 1,2,3... ARE CALLED NATURAL NUMBERS USING FOR LOOP.

```
#include <stdio.h>
void main()
{
    int n, count, sum=0;
    printf("Enter the value of n:");
    scanf("%d",&n);
    for(count=1;count<=n;++count) //for loop terminates if count>n
    {
        sum+=count; // this statement is equivalent to sum=sum+count
    }
    printf("Sum=%d",sum);
}
```

Output:

Enter the value of n: 19
Sum=190

PROGRAM TO FIND SUM OF SERIES $1+x+x^2+x^3+.....+x^n$

```
#include<stdio.h>
#include<math.h>
void main()
{
    int i,n;
    float x,sum=1;
    printf(" Enter the value of x and n:");
    scanf("%f %d", &x,&n);
    for(i=1;i<=n;++i)
        sum+=pow(x,i);
    printf(" Sum= %f ",sum);
}
```

Output:

Enter the value of x and n:
2 3
Sum= 15

PROGRAM TO FIND THE SUM OF THE SERIES $x+x^2/2+x^3/3+.....+x^n/n$

```
#include<stdio.h>
#include<math.h>
void main()
{
    int i,n;
    float x,sum=0;
    printf(" Enter the value of x and n:");
    scanf("%f %d", &x,&n);
    for(i=1;i<=n;++i)
        sum+=pow(x,i)/i;
    printf(" Sum= %f ",sum);
}
```

Output:

Enter the value of x and n:
3 5
Sum= 85.349998

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO FIND HCF AND LCM OF 2 NUMBERS.**

```
#include <stdio.h>
void main()
{
    int a, b, x, y, t, gcd, lcm;

    printf("Enter two integers\n");
    scanf("%d%d", &x, &y);

    a = x;
    b = y;

    while (b != 0)
    {
        t = b;
        b = a % b;
        a = t;
    }

    gcd = a;
    lcm = (x*y)/gcd;

    printf("Greatest common divisor of %d and %d = %d\n", x, y, gcd);
    printf("Least common multiple of %d and %d = %d\n", x, y, lcm);
}
```

Output:

Greatest common divisor of 9 and 24 = 3
Least common multiple of 9 and 24 = 72

PROGRAM TO COUNT NUMBER OF DIGITS OF AN INTEGER.

```
#include <stdio.h>
void main()
{
    int n, count=0;
    printf("Enter an integer: ");
    scanf("%d", &n);
    while(n!=0)
    {
        n/=10;          // n=n/10
        ++count;
    }
    printf("Number of digits: %d", count);
}
```

Output:

Enter an integer: 34523
Number of digits: 5

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO REVERSE A NUMBER.**

```
#include <stdio.h>
void main()
{
    int n, reverse=0, rem;
    printf("Enter an integer: ");
    scanf("%d", &n);
    while(n!=0)
    {
        rem=n%10;
        reverse=reverse*10+rem;
        n/=10;
    }
    printf("Reversed Number = %d",reverse);
    return;
}
```

Output:

Enter an integer: 2345
Reversed Number = 5432

PROGRAM TO CHECK WHETHER A NUMBER IS PALINDROME OR NOT.

```
#include <stdio.h>
void main()
{
    int n, reverse=0, rem,temp;
    printf("Enter an integer: ");
    scanf("%d", &n);
    temp=n;
    while(temp!=0)
    {
        rem=temp%10;
        reverse=reverse*10+rem;
        temp/=10;
    }

    if(reverse==n)
        printf("%d is a palindrome.",n);
    else
        printf("%d is not a palindrome.",n);
    return;
}
```

Output:

Enter an integer: 12321
12321 is a palindrome.

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO FIND FACTORIAL OF A NUMBER.**

```
#include <stdio.h>
void main()
{
    int n, count;
    unsigned long long int factorial=1;
    printf("Enter an integer: ");
    scanf("%d",&n);
    if ( n < 0)
        printf("Error!!! Factorial of negative number doesn't exist.");
    else
    {
        for(count=1;count<=n;++count)    // for loop terminates if count>n
        {
            factorial*=count;           // factorial=factorial*count
        }
        printf("Factorial = %lu",factorial);
    }
}
```

Output:

Enter an integer: 10
Factorial = 3628800

PROGRAM TO DISPLAY FIBONACCI SERIES.

```
#include <stdio.h>
void main()
{
    int count, n, t1=0, t2=1, display=0;
    printf("Enter number of terms: ");
    scanf("%d",&n);
    printf("Fibonacci Series: %d+%d+", t1, t2);
    count=2;
    while (count<n)
    {
        display=t1+t2;
        t1=t2;
        t2=display;
        ++count;
        printf("%d+",display);
    }
    return;
}
```

Output:

Enter number of terms: 10
Fibonacci Series: 0+1+1+2+3+5+8+13+21+34+

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM GENERATE FIRST N PRIME NUMBERS.**

```
#include<stdio.h>
void main()
{
    int n, i = 3, count, c;
    printf("Enter the number of prime numbers required\n");
    scanf("%d",&n);
    if ( n >= 1 )
    {
        printf("First %d prime numbers are :\n",n);
        printf("2\n");
    }
    for ( count = 2 ; count <= n ; )
    {
        for ( c = 2 ; c <= i - 1 ; c++ )
        {
            if ( i%c == 0 )
                break;
        }
        if ( c == i )
        {
            printf("%d\n",i);
            count++;
        }
        i++;
    }
    return;
}
```

Output:

```
Enter the number of prime numbers required
5
First %d prime numbers are :
2 3 5 7 11
```

PROGRAM TO CHECK WHETHER A NUMBER IS PRIME OR NOT.

```
#include <stdio.h>
void main()
{
    int n, i, flag=0;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    for(i=2;i<=n/2;++i)
    {
        if(n%i==0)
        {
            flag=1;
            break;
        }
    }
    if (flag==0)
        printf("%d is a prime number.",n);
    else
        printf("%d is not a prime number.",n);
}
```

Output:

```
Enter a positive integer: 29
29 is a prime number.
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO FIND THE SINE VALUE FOR A GIVEN VALUE**

```
#include<stdio.h>
#include<math.h>
void main()
{
    int i,j,deno,val,n;
    float x,y;
    printf("\n Enter the no. of term's\t");
    scanf("%d",&n);
    printf("\n Enter the value to calculate[Range -1 to 1]\t");
    scanf("%f",&y);
    x=y;
    deno=3;
    for(i=2;i<=n;i++)
    {
        val=1;
        for(j=1;j<=deno;j++)
            val=val*j;

        if((i%2)==1)
            x=x+(pow(x,deno))/val;
        else
            x=x-(pow(x,deno))/val;

        deno+=2;
    }
    printf("\nCalculated Sine value:- %.6f",x);
    printf("\nComputers Sine value:- %.6f",sin(y));
}
```

Output:

```
Enter the no. of term's 5
Enter the value to calculate[Range -1 to 1] 0.89
Calculated Sine value:- 0.774761
Computers Sine value:- 0.777072
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO MAKE A SIMPLE CALCULATOR TO ADD, SUBTRACT, MULTIPLY OR DIVIDE USING SWITCH...CASE.**

```
void main()
{
    char o;
    float num1,num2;
    printf("Enter operator either + or - or * or divide : ");
    scanf("%c",&o);
    printf("Enter two operands: ");
    scanf("%f%f",&num1,&num2);
    switch(o)
    {
        case '+': printf("%.1f + %.1f = %.1f",num1, num2, num1+num2);
                  break;
        case '-': printf("%.1f - %.1f = %.1f",num1, num2, num1-num2);
                  break;
        case '*': printf("%.1f * %.1f = %.1f",num1, num2, num1*num2);
                  break;
        case '/': printf("%.1f / %.1f = %.1f",num1, num2, num1/num2);
                  break;
        default: printf("Error! operator is not correct");
                  break;
    }
    return;
}
```

Output:

```
Enter operator either + or - or * or divide : -
Enter two operands: 3.4
8.4
3.4 - 8.4 = -5.0
```

PROGRAM TO PRINT A REMARKS BASED ON GRADE USING SWITCH...CASE

```
#include <stdio.h>
main()
{
    char grade ;
    printf("enter grade of a student: ")
    scanf("%c", &grade);
    switch(grade)
    {
        case 'A' : printf( "Excellent\n" );
                  break;
        case 'B' : printf( "Good\n" );
                  break;
        case 'C' : printf( "OK\n" );
                  break;
        case 'D' : printf( "Mmmmm....\n" );
                  break;
        case 'F' : printf( "You must do better than this\n" );
                  break;
        default  : printf( "What is your grade anyway?\n" );
                  break;
    }
}
```

Output:

```
enter grade of a student: B
Good
```



C PROGRAMS BASED ON ARRAYS

PROGRAM TO CALCULATE SUM & AVERAGE OF AN ARRAY.

```
#include <stdio.h>
void main()
{
    int array[MAXSIZE];
    int i, num, negative_sum = 0, positive_sum = 0;
    float total = 0.0, average;

    printf("Enter the value of N \n");
    scanf("%d", &num);
    printf("Enter %d numbers (-ve, +ve and zero) \n", num);
    for (i = 0; i < num; i++)
    {
        scanf("%d", &array[i]);
    }

    // Summation starts
    for (i = 0; i < num; i++)
    {
        if (array[i] < 0)
        {
            negative_sum = negative_sum + array[i];
        }
        else if (array[i] > 0)
        {
            positive_sum = positive_sum + array[i];
        }
        else if (array[i] == 0)
        {
            ;
        }
        total = total + array[i] ;
    }
    average = total / num;
    printf("Sum of all negative numbers = %d \n", negative_sum);
    printf("Sum of all positive numbers = %d \n", positive_sum);
    printf("Average of all input numbers = %.2f", average);
}
```

Output:

```
Enter the value of N
10
Enter 10 numbers (-ve, +ve and zero)
-8 9 -100 -80 90 45 -23 -1 0 16
Sum of all negative numbers = -212
Sum of all positive numbers = 160
Average of all input numbers = -5.20
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO FIND LARGEST ELEMENT OF AN ARRAY.**

```
#include <stdio.h>
void main()
{
    int i,n;
    float arr[100];
    printf("Enter total number of elements(1 to 100): ");
    scanf("%d",&n);
    printf("\n");
    for(i=0;i<n;++i) // Stores number entered by user.
    {
        printf("Enter Number %d: ",i+1);
        scanf("%f",&arr[i]);
    }
    for(i=1;i<n;++i) // Loop to store largest number to arr[0]
    {
        if(arr[0]<arr[i]) // Change < to > if you want to find smallest element
            arr[0]=arr[i];
    }
    printf("Largest element = %.2f",arr[0]);
}
```

Output:

```
Enter total number of elements(1 to 100): 8
Enter Number 1: 23.4
Enter Number 2: -34.5
Enter Number 3: 50
Enter Number 4: 33.5
Enter Number 5: 55.5
Enter Number 6: 43.7
Enter Number 7: 5.7
Enter Number 8: -66.5
Largest element = 55.5
```

PROGRAM TO GENERATE FIBONACCI SERIES USING AN ARRAY.

```
#include <stdio.h>
main()
{
    int fib[100];
    int i, n;
    fib[0] = 0;
    fib[1] = 1;
    printf("enter n value: ");
    scanf("%d", &n);
    for(i = 2; i < n; i++)
    {
        fib[i] = fib[i-1] + fib[i-2];
    }
    printf("fibonacci series is ..... \n");
    for(i = 0; i < n; i++)
    {
        printf("%d ", fib[i]);
    }
}
```

Output:

```
enter n value:7
fibonacci series is .....
0 1 1 2 3 5 8
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO FIND THE VALUE OF THE POLYNOMIAL $f(x)=a_4x^4+a_3x^3+a_2x^2+a_1x+a_0$ USING HORNER'S METHOD.**

```
#include "stdio.h"
void main()
{
    float a[100],sum=0,x;
    int n,i;
    printf("Enter the degree of the polynomial:");
    scanf("%d",&n);
    printf("Enter the coefficients into the array:");
    for(i=n;i>=0;i--)
    {
        scanf("%f",&a[i]);
    }
    printf("Enter the value of x:");
    scanf("%f",&x);
    for(i=n;i>0;i--)
    {
        sum=(sum+a[i])*x;
    }
    sum=sum+a[0];
    printf(" Value of the polynomial is =%f",sum);
}
```

Output:

```
Enter the degree of the polynomial:
4
Enter the coefficients into the array:
3 2 1 1 2
Enter the value of x:
2
Value of the polynomial is:72.000000
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO SORT N NUMBERS IN ASCENDING ORDER USING BUBBLE SORT.**

```
#include <stdio.h>
#define MAXSIZE 10
void main()
{
    int array[MAXSIZE];
    int i, j, num, temp;

    printf("Enter the value of num: \n");
    scanf("%d", &num);
    printf("Enter the elements one by one: \n");
    for (i = 0; i < num; i++)
    {
        scanf("%d", &array[i]);
    }
    printf("Input array is: \n");
    for (i = 0; i < num; i++)
    {
        printf("%d ", array[i]);
    }
    // Bubble sorting begins
    for (i = 0; i < num; i++)
    {
        for (j = 0; j < (num - i - 1); j++)
        {
            if (array[j] > array[j + 1])
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
    printf("Sorted array is :\n ");
    for (i = 0; i < num; i++)
    {
        printf("%d ", array[i]);
    }
}
```

Output:

```
Enter the value of num:
6
Enter the elements one by one:
23 45 67 89 12 34
Input array is:
23 45 67 89 12 34
Sorted array is.:
12 23 34 45 67 89
```


**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO IMPLEMENT LINEAR SEARCH.**

```
#include <stdio.h>
void main()
{
    int array[10];
    int i, num, keynum, found = 0;

    printf("Enter the value of num \n");
    scanf("%d", &num);
    printf("Enter the elements one by one \n");
    for (i = 0; i < num; i++)
    {
        scanf("%d", &array[i]);
    }

    printf("Enter the element to be searched \n");
    scanf("%d", &keynum);
    // Linear search begins
    for (i = 0; i < num ; i++)
    {
        if (keynum == array[i] )
        {
            found = 1;
            break;
        }
    }
    if (found == 1)
        printf("Element is present in the array\n");
    else
        printf("Element is not present in the array\n");
}
```

Output:

```
Enter the value of num
5
Enter the elements one by one
23 90 56 15 58
Enter the element to be searched
56
Element is present in the array
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO ACCEPT SORTED ARRAY AND DO SEARCH USING BINARY SEARCH.**

```
#include <stdio.h>
void main()
{
    int array[10];
    int i, j, num, temp, keynum;
    int low, mid, high;
    printf("Enter the value of num:");
    scanf("%d", &num);
    printf("Enter the elements one by one \n");
    for (i = 0; i < num; i++)
        scanf("%d", &array[i]);
    printf("Input array elements \n");
    for (i = 0; i < num; i++)
        printf("%d  ", array[i]);

    for (i = 0; i < num; i++)          // Bubble sorting begins
        for (j = 0; j < (num - i - 1); j++)
            if (array[j] > array[j + 1])
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }

    printf("Sorted array is...\n");
    for (i = 0; i < num; i++)
        printf("%d ", array[i]);
    printf("Enter the element to be searched:");
    scanf("%d", &keynum);

    low = 1;          // Binary searching begins
    high = num;
    do
    {
        mid = (low + high) / 2;
        if (keynum < array[mid])
            high = mid - 1;
        else if (keynum > array[mid])
            low = mid + 1;
    } while (keynum != array[mid] && low <= high);

    if (keynum == array[mid])
        printf("SEARCH SUCCESSFUL ");
    else
        printf("SEARCH FAILED ");
}
```

Output:

```
Enter the value of num: 5
Enter the elements one by one
23 90 56 15 58
Input array elements
23 90 56 15 58
Sorted array is...
15 23 56 58 90
Enter the element to be searched: 58
SEARCH SUCCESSFUL
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO CALCULATE THE ADDITION OR SUBTRACTION OF 2 MATRICES.**

```
#include <stdio.h>
void main()
{
    int array1[10][10], array2[10][10], arraysum[10][10],
    arraydiff[10][10];
    int i, j, m, n, option;

    printf("Enter the order of the matrix array1 and array2 \n");
    scanf("%d %d", &m, &n);
    printf("Enter the elements of matrix array1 \n");
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &array1[i][j]);
        }
    }
    printf("Enter the elements of matrix array2 \n");
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &array2[i][j]);
        }
    }
    printf("Enter your option: 1 for Addition and 2 for Subtraction \n");
    scanf("%d", &option);
    switch (option)
    {
        case 1:
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                {
                    arraysum[i][j] = array1[i][j] + array2[i][j];
                }
            }
            printf("Sum matrix is \n");
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                {
                    printf("%3d", arraysum[i][j]) ;
                }
                printf("\n");
            }
            break;
        case 2:
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                {
                    arraydiff[i][j] = array1[i][j] - array2[i][j];
                }
            }
    }
}
```

**PROGRAMMING IN C AND DATA STRUCTURES**

```
        printf("Difference matrix is \n");
        for (i = 0; i < m; i++)
        {
            for (j = 0; j < n; j++)
            {
                printf("%3d", arraydiff[i][j]) ;
            }
            printf("\n");
        }
        break;
    }
}
```

Output:

```
Enter the order of the matrix array1 and array2
3 3
Enter the elements of matrix array1
2 3 4
7 8 9
5 6 8
Enter the elements of matrix array2
3 3 3
3 4 6
8 4 7
Enter your option: 1 for Addition and 2 for Subtraction
1
Sum matrix is
 5 6 7
10 12 15
13 10 15
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO COMPUTE THE PRODUCT OF TWO MATRICES.**

```
#include<stdio.h>
void main()
{
    int array1[10][10], array2[10][10],
    array3[10][10];
    int m, n;
    int i, j ,k;

    printf("Enter the value of m and n \n");
    scanf("%d %d", &m, &n);
    printf("Enter Matrix array1 \n");
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &arr[i][j]);

    printf("Enter Matrix array2 \n");
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &arr[i][j]);

    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
        {
            array3[i][j] = 0;
            for (k = 0; k < n; k++)
            {
                array3[i][j] = array3[i][j] + array1[i][k] * array2[k][j];
            }
        }
    printf("The product matrix is \n");
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("%3d", arr[i][j]);
        }
        printf("\n");
    }
}
```

Output:

```
Enter the value of m and n
3 3
Enter matrix array1
4 5 6
1 2 3
3 7 8
Enter matrix array2
5 6 9
8 5 3
2 9 1
The product matrix is
72103 57
27 43 18
87125 56
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO FIND THE TRANSPOSE OF A GIVEN MATRIX.**

```
#include <stdio.h>
void main()
{
    int array[10][10];
    int i, j, m, n;

    printf("Enter the order of the matrix \n");
    scanf("%d %d", &m, &n);
    printf("Enter the coefficients of the matrix\n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            scanf("%d", &array[i][j]);
        }
    }
    printf("Transpose of matrix is \n");
    for (j = 0; j < n; ++j)
    {
        for (i = 0; i < m; ++i)
        {
            printf(" %d", array[i][j]);
        }
        printf("\n");
    }
}
```

Output:

```
Enter the order of the matrix
3 3
Enter the coefficients of the matrix
3 7 9
2 7 5
6 3 4
Transpose of matrix is
3 2 6
7 7 3
9 5 4
```



C PROGRAMS BASED ON FUNCTIONS

PROGRAM TO FIND SQUARE OF A NUMBER USING FUNCTION.

```
#include<stdio.h>
int square()
{
    return n*n;
}
void main()
{
    int n, p;
    printf("enter a number");
    scanf("%d",&n);
    p= square(n);
    printf("The square of the number is %d",p);
}
```

Output:

enter a number
4
The square of the number is 16

PROGRAM TO FIND CUBE OF A NUMBER USING FUNCTION.

```
#include<stdio.h>
int cube()
{
    return n*n*n;
}
void main()
{
    int n, p;
    printf("enter a number");
    scanf("%d",&n);
    p=cube(n);
    printf("The cube of the number is %d",p);
}
```

Output:

enter a number
4
The cube of the number is 64

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO CALCULATE THE POWER OF A NUMBER USING RECURSION.**

```
#include <stdio.h>

int power(int base, int exp)
{
    if ( exp!=1 )
        return (base*power(base,exp-1));
}

void main()
{
    int base, exp;
    printf("Enter base number: ");
    scanf("%d", &base);
    printf("Enter power number(positive integer): ");
    scanf("%d", &exp);
    printf("%d^%d = %d", base, exp, power(base, exp));
    return;
}
```

Output:

```
Enter base number: 3
Enter power number(positive integer): 3
3^3 = 27
```

PROGRAM TO PRINT FIBONACCI SERIES USING RECURSION.

```
#include<stdio.h>

int Fib(int n)
{
    if ( n == 0 )
        return 0;
    else if ( n == 1 )
        return 1;
    else
        return Fib(n-1)+Fib(n-2);
}

void main()
{
    int n, i = 0, c;
    printf("enter n value: ");
    scanf("%d",&n);
    printf("Fibonacci series . . . \n");
    for ( c = 1 ; c <= n ; c++ )
    {
        printf("%d \t", Fib(i));
        i++;
    }
}
```

Output:

```
enter n value: 7
Fibonacci series . . .
0 1 1 2 3 5 8
```


**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO CALCULATE FACTORIAL OF A NUMBER USING RECURSION.**

```
#include<stdio.h>

int fact(int n)
{
    if(n!=1)
        return n*fact(n-1);
}

void main()
{
    int n;
    printf("Enter an positive integer: ");
    scanf("%d", &n);
    printf("Factorial of %d = %ld", n, fact(n));
    return;
}
```

Output:

Enter an positive integer: 6
Factorial of 6 = 720

PROGRAM TO SOLVE TOWER-OF-HANOI PROBLEM USING RECURSION

```
#include <stdio.h>

void towers(int num, char frompeg, char topeg, char auxpeg)
{
    if (num == 1)
    {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}

void main()
{
    int num;
    printf("Enter the number of disks : ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are :\n");
    towers(num, 'A', 'C', 'B');
}
```

Output:

Enter the number of disks : 3
The sequence of moves involved in the Tower of Hanoi are :
Move disk 1 from peg A to peg C
Move disk 2 from peg A to peg B
Move disk 1 from peg C to peg B
Move disk 3 from peg A to peg C
Move disk 1 from peg B to peg A
Move disk 2 from peg B to peg C
Move disk 1 from peg A to peg C

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO PERFORM BINARY SEARCH USING RECURSION**

```
#include <stdio.h>
void bubble_sort(int list[], int size)
{
    int temp, i, j;
    for (i = 0; i < size; i++)
        for (j = i; j < size; j++)
            if (list[i] > list[j])
            {
                temp = list[i];
                list[i] = list[j];
                list[j] = temp;
            }
}
void binary_search(int list[], int lo, int hi, int key)
{
    int mid;
    if (lo > hi)
    {
        printf("Key not found\n");
        return;
    }
    mid = (lo + hi) / 2;
    if (list[mid] == key)
    {
        printf("Key found\n");
    }
    else if (list[mid] > key)
    {
        binary_search(list, lo, mid - 1, key);
    }
    else if (list[mid] < key)
    {
        binary_search(list, mid + 1, hi, key);
    }
}
void main()
{
    int key, size, i;
    int list[25];
    printf("Enter size of a list: ");
    scanf("%d", &size);
    printf("Enter %d numbers ", size);
    for(i = 0; i < size; i++)
        scanf("%d ", &list[i]);
    bubble_sort(list, size);
    printf("\n\n");
    printf("Enter key to search: ");
    scanf("%d", &key);
    binary_search(list, 0, size, key);
}
```

Output:

```
Enter size of a list: 10
Enter 10 numbers : 83 86 77 15 93 35 86 92 49 21
Enter key to search: 21
Key found
```



C PROGRAMS BASED ON STRUCTURES AND FILE MANAGEMENT

PROGRAM TO STORE INFORMATION (NAME, ROLL AND MARKS) OF A STUDENT USING STRUCTURE.

```
#include <stdio.h>
struct student
{
    char name[50];
    int roll;
    float marks;
};
void main()
{
    struct student s;
    printf("Enter information of students:\n\n");
    printf("Enter name: ");
    scanf("%s",s.name);
    printf("Enter roll number: ");
    scanf("%d",&s.roll);
    printf("Enter marks: ");
    scanf("%f",&s.marks);
    printf("\nDisplaying Information\n");
    printf("Name: %s\n",s.name);
    printf("Roll: %d\n",s.roll);
    printf("Marks: %.2f\n",s.marks);
    return;
}
```

Output:

Enter information of students:

Enter name: Adele

Enter roll number: 21

Enter marks: 334.5

Displaying Information

name: Adele

Roll: 21

Marks: 334.50

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO ADD TWO COMPLEX NUMBERS BY PASSING STRUCTURE TO A FUNCTION.**

```
#include <stdio.h>
typedef struct complex
{
    float real;
    float imag;
}complex;

complex add(complex n1,complex n2);

void main()
{
    complex n1,n2,temp;
    printf("For 1st complex number \n");
    printf("Enter real and imaginary respectively:\n");
    scanf("%f%f",&n1.real,&n1.imag);
    printf("\nFor 2nd complex number \n");
    printf("Enter real and imaginary respectively:\n");
    scanf("%f%f",&n2.real,&n2.imag);
    temp=add(n1,n2);
    printf("Sum=%.1f+%.1fi",temp.real,temp.imag);
    return;
}

complex add(complex n1,complex n2)
{
    complex temp;
    temp.real=n1.real+n2.real;
    temp.imag=n1.imag+n2.imag;
    return(temp);
}
```

Output:

```
For 1st complex number
Enter real and imaginary respectively: 2.3
4.5
For 1st complex number
Enter real and imaginary respectively: 3.4
5
Sum=5.7+9.5i
```

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO STORE INFORMATION OF STUDENTS USING STRUCTURE.**

```
#include <stdio.h>
struct student
{
    char name[50];
    int roll;
    float marks;
};

void main()
{
    struct student s[10];
    int i;
    printf("Enter information of students:\n");
    for(i=0;i<10;++i)
    {
        s[i].roll=i+1;
        printf("\nFor roll number %d\n",s[i].roll);
        printf("Enter name: ");
        scanf("%s",s[i].name);
        printf("Enter marks: ");
        scanf("%f",&s[i].marks);
        printf("\n");
    }
    printf("Displaying information of students:\n\n");
    for(i=0;i<10;++i)
    {
        printf("\nInformation for roll number %d:\n",i+1);
        printf("Name: ");
        puts(s[i].name);
        printf("Marks: %.1f",s[i].marks);
    }
    return;
}
```

Output:

Enter information of students:

For roll number 1

Enter name: Tom

Enter marks: 98

For roll number 2

Enter name: Jerry

Enter marks: 89

.
.

Displaying information of students:

Information for roll number 1:

Name: Tom

Marks: 98

.
.

**PROGRAMMING IN C AND DATA STRUCTURES****PROGRAM TO READ NAME AND MARKS OF N NUMBER OF STUDENTS FROM USER AND STORE THEM IN A FILE.**

```
void main()
{
    char name[50];
    int marks,i,n;
    FILE *fptr;
    printf("Enter number of students: ");
    scanf("%d",&n);
    fptr=(fopen("C:\\student.txt","w"));
    for(i=0;i<n;++i)
    {
        printf("For student %d\n Enter name: ",i+1);
        scanf("%s",name);
        printf("Enter marks: ");
        scanf("%d",&marks);
        fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);
    }
    fclose(fptr);
}
```

Output:

```
Enter number of students: 2
For student 1
Enter name: sri
Enter marks: 75
For student 2
.....
```

PROGRAM TO OPEN A FILE AND WRITE DATA USING THE FUNCTION fprintf().

```
void main()
{
    int n;
    FILE *fptr;
    fptr=fopen("C:\\program1.txt","w");
    printf("Enter n: ");
    scanf("%d",&n);
    fprintf(fptr,"%d",n);
    fclose(fptr);
}
```

Output:

```
Enter n: 1234
```

PROGRAM TO OPEN A FILE AND READ DATA USING THE FUNCTION fscanf()

```
void main()
{
    int n;
    FILE *fptr;
    fptr=fopen("C:\\program1.txt","r")
    fscanf(fptr,"%d",&n);
    printf("Value of n= %d",n);
    fclose(fptr);
}
```

Output:

```
Value of n= 1234
```