QUESTION PAPER SOLUTION

MODULE I

INTRODUCTION TO C LANGUAGE

1. Explain the input & output statements with examples. (JULY 2014, JUN/JULY 2015)

Soln :Formatted Input / Output includes following:

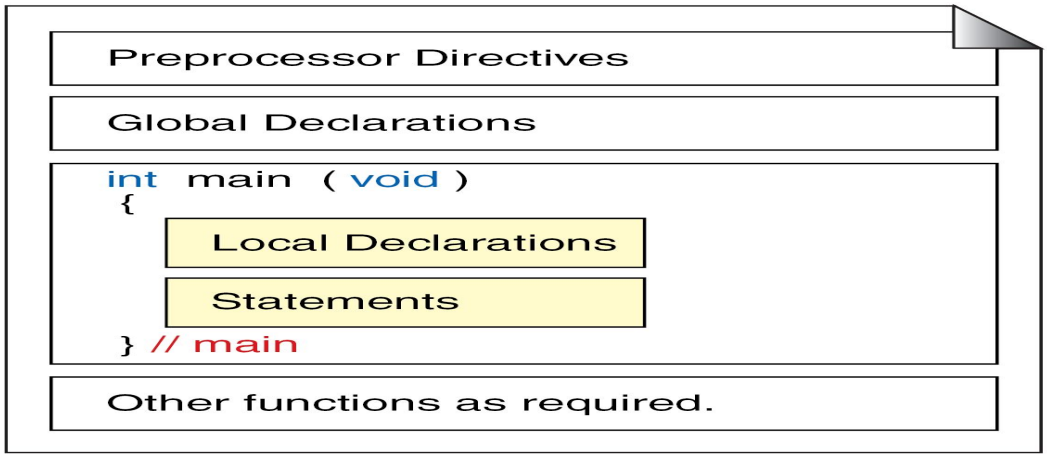
Printf

It is an formatted output statement whose syntax contains its arguments are, in order; a control string, which controls what get printed, followed by a list of values to be substituted for entries in the control string. The prototype for the printf() is:

Printf(format-string, var1,var2…..varn);

2. Draw the structure of a C-program & explain in brief (DEC/JAN 2014, JULY 2014, JAN 2015)

Soln :The basic structure of a C program is shown below



The documentation section consists of a set of comment lines giving the name of the program, the name author and other details which the programmer would like to use later.
The link section provides instructions to the compiler to link functions from the system library.
The definition section contains all symbolic constants.
There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section.
Every C program must have one main() function section. This section contains two parts declaration part and executable part. The declaration part declares all the variables used in the executable part. There should be at least one statement in the executable part. These two parts must appear between the opening and the closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function section is logical end of the program.
All statements in the declaration and executable parts end with a semicolon.
The subprogram section contains all the user-defined functions that are called in the main function. The main function is very important compared to other sections.

3. Explain the different phases of solving a given problem using computer. (JUN/JULY 2013)

Soln: The following steps need to be followed:

1. read the problem carefully
2. understand what the problem entails
3. and only then, write down the steps to solve the problem.

- **Problem Statement**
- **Input/Output Description**
- **Algorithm Development**

- These steps are called an **algorithm** that can be defined as a set of sequential instructions to solve a problem.
- The most important aspect of solving a problem by using a computer is to write an algorithm to solve it. An algorithm is a set of steps that must be written in such a way that is it unambiguous and precise. The computer cannot think for itself – you as the programmer must tell the computer exactly what to do. You may never

assume that the computer will do something if you have not explicitly included the specific step.

4. What are tokens? Explain the various types of tokens with example. (JUN/JULY 2013,JAN 2014,JUN/JULY 2015)

Soln: A **Token** is the basic and the smallest unit of a program
There are **6 types of tokens in 'C'**. They are:
1) Keywords
2) Identifiers
3) Constants
4) Strings
5) Special symbols
6) Operators

Key words & Character set

Instructions in C language are formed using syntax and keywords. It is necessary to strictly follow C language Syntax rules. Any instructions that mismatches with C language Syntax generates an error while compiling the program. Keywords should not be used either as Variable or Constant names. The character set in C Language can be grouped into the following categories.

1. **Letters**
2. **Digits**
3. **Special Characters**
4. **White Spaces**

Constants

Constants in C refer to fixed values that do not change during the executing of a program. C Support several types of constants are listed below.

1. **Numeric Constants**
   i. Integer Constants
   ii. Real Constants
2. **Character Constants**
   i. Single Character Constants
   ii. String Constants

Operators Introduction

An operator is a symbol which helps the user to command the computer to do a certain mathematical or logical manipulations. Operators are used in C language program to operate on data and variables. C has a rich set of operators as listed previously.

Classification of Operators

- Arithmetic operators
- Relational Operators
- Logical Operators
- Assignment Operators

- Increments and Decrement Operators
- Conditional Operators
- Bitwise Operators
- Special Operators.

5. Explain software development life cycle. (JUN/JULY 2013)

Soln:

- **Systems analysis, requirements definition**: Defines project goals into defined functions and operation of the intended application. Analyzes end-user information needs.
- **Systems design**: Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode and other documentation.
- **Development**: The real code is written here.
- **Integration and testing**: Brings all the pieces together into a special testing environment, then checks for errors, bugs and interoperability.
- **Acceptance, installation, deployment**: The final stage of initial development, where the software is put into production and runs actual business.

6. What are identifiers? Discuss the rules to be followed while naming identifiers. Give Examples. (JUN/JULY 2013)

Soln: In c language every word is classified into either keyword or identifier. All keywords have fixed meanings and these meanings cannot be changed. These serve as basic building blocks for program statements. Identifiers refer to the names of variables, functions and arrays.

Rules for defining identifiers:

Identifiers names may consist of letters, digits, and the underscore(_) character, subject to the rules given below:

1. The **identifiers** must always begin with a letter. Some systems permit underscore as the first character.
2. ANSI standard recognizes a length of 31 characters. However, the length should not be normally mare than eight characters. Since first eight characters are treated as significant by many compilers.
3. Uppercase and lowercase are significant. That is ,the variable Rate is not the same as rate or TOTAL.
4. The **identifiers** name should not be a keyword.
5. White space is not allowed.

7. Explain format specifiers used in scanf( ) function to read int, float, char, double and longint datatypes. (JUNE/JULY 2013)

Soln: Format specifiers used in scanf( ) are %d for integer, %f for floating point, %c for character, %f for double.

8. Explain different datatypes available in C (Dec /JAN 2014,JUN/JULY 2015)
Soln: C has the following basic built-in datatypes.

- int
- float
- double
- char

int - data type

int is used to define integer numbers

float - data type

float is used to define floating point numbers.

double - data type

double is used to define BIG floating point numbers. It reserves twice the storage for the number. On PCs this is likely to be 8 bytes.

char - data type

char defines characters.

9. Explain precedence and associativity of operators in C with example (JUN/JULY2013, JAN 2015)

Soln :Arithmetic operators precedence:-

In a program the value of any expression is calculated by executing one arithmetic operation at a time. The order in which the arithmetic operations are executed in an expression is based on the rules of precedence of operators.

The precedence of operators is :

Unary (-) FIRST
Multiplication(*) SECOND
Division(/) and (%)
Addition(+) and Subtraction(-) LAST

For example, in the integer expression –a *b/c+d the unary- is done first, the result –a is multiplied by b, the product is divided by c(integer division) and d is added to it. The answer is thus:

-ab/c+d

All the expressions are evaluated from left to right. All the unary negations are done first. After completing this the expression is scanned from left to right; now all *, / and % operations are executed in the order of their appearance. Finally all the additions and subtractions are done starting from the left of the expression.

Parentheses are used if the order of operations governed by the precedence rules are to overridden. In the expression with a single pair of parentheses the expression inside the parentheses is evaluated FIRST. Within the parentheses the evaluation is governed by the precedence rules.

For example, in the expression:

a * b/(c+d * k/m+k)+a

the expression within the parentheses is evaluated first giving:

c+dk/m+k

After this the expression is evaluated from left to right using again the rules of precedence giving

ab/c+dk/m+k +a

The associativity of operators:

The operators of the same precedence are evaluated either from left to right or from right to left depending on the level. This is known as the associativity property of an operator.

The table below shows the associativity of the operators:

Operators	Associativity
() [ ] →	left to right
~ ! –(unary)	left to right
<<>>	left to right
<<= >>=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
=+ =- *= /= %= &= ^= |= <<= >>=	right to left
,(comma operator)	left to right

10. What is type conversion? What are the different ways of type conversion? Explain with an example. (JUN/JULY 2013, JAN 2015)

Soln : The process of converting operand or an expression of one type, into another data type is referred as type casting or type conversion. There are two types:

1) Implicit type casting (or Automatic)
2) Explicit type casting
3)

Implicit type casting: This is the process of converting an operand or an expression of one type, into other type; this process is done by the compiler itself at the time of execution. Hence it is also known as automatic type conversion.

Example for implicit type conversion:

```
int  a ;
float  b , c ;
      c = a / b    5 / 9 =  0.55
```

Explicit type casting: This is also the process which converts an operand of one type, into another type; But this process is done by the user explicitly by using type qualifiers.

Example for explicit type conversion:

```
int a , b ;
float   c ;
      c = (float) a / b   (5 / 9 = 0.55   where (float) is called as type qualifier.
```

If we don't use the type qualifier (float) it results with answer **0.00** instead of ***0.55.***

11. Write C program to swap values of two integers without using third variable and give flow chart for the same. (jun/jul 2013)

Soln:

```
#include <stdio.h>
main()
        {
                int a=5, b=10;
                a=a+b; b=a-b; a=a-b;
        }
```

12. Find the result of each of the following expressions with i=4, j=2, k=6, a=2.
i) k*=i+j ii) j=j/=k iii) i%=i/3 iv) m= i+(j=2+k) v) a=i*(j/=k/2) **(jun/jul 2013)**

i) K*=i+j, K=K*i+j= 6*4+2 =26
ii) J=j/=k, j=j/k, j=2/6= 0
iii) i%=i/3, i=i%i/3, i=i%4/3=0
iv) m=i+(j=2+k), m=4+(j=2+6), m=4+8=12
v) a=i*(j=j/k/2), a=4*(j=2/6/2)=0

13. Explain relational operators in C, with examples. (JAN 2014, JULY 2014, JAN 2015,JUN/JULY 2015)

Soln :i) Relational

A simple relational expression contains only one relational operator and takes the following form.

exp1 relational operator exp2

Where exp1 and exp2 are expressions, which may be simple constants, variables or combination of them. Given below is a list of examples of relational expressions and evaluated values.

6.5	<=	25		TRUE
-65	>	0		FALSE
10	<	7	+ 5	TRUE

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to != is not equal to

Often it is required to compare the relationship between operands and bring out a decision and program accordingly. This is when the relational operator come into picture. C supports the following relational operators.

ii) Increment

C allows two very useful operators not generally found in other languages. These are the increment and decrement operators: ++ and --

The Operator ++ adds 1 to the operand (variable), while - - subtracts 1. Both are unary operators and takes the following form: The syntax of the operators is given below

1. ++ variable name
2. variable name++
3. --variable name
4. variable name--

The increment operator ++ adds the value 1 to the current value of operand and the decrement operator -- subtracts the value 1 from the current value of operand. ++variable name and variable name++ mean the same thing when they form statements independently, they behave differently when they are used in expression on the right hand side of an assignment statement. Consider the following

m = 5;
y = ++m; (prefix)

In this case the value of y and m would be 6 Suppose if we rewrite the above statement as m = 5;
y = m++; (post fix) Then the value of y will be 5 and that of m will be 6. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on the left. On the other hand, a postfix operator first assigns the value to the variable on the left and then increments the operand.

iii) Conditional

The conditional operator consists of 2 symbols the question mark (?) and the colon (:) The syntax for a ternary operator is as follows

exp1 ? exp2 : exp3

Here exp1 is evaluated first. If the expression is true then exp2 is evaluated & its value becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expression is evaluated.

For example

```
a = 10;
b = 15;
x = (a > b) ? a : b
```

Here x will be assigned to the value of b.
The condition follows that the expression is false therefore b is assigned to x.

iv) Special operators.

C supports some special operators of interest such as comma operator, size of operator, pointer operators (& and *) and member selection operators (. and ->). The size of and the comma operators are discussed here. The remaining operators are discussed in forth coming chapters.

The Comma Operator

The comma operator can be used to link related expressions together. A comma-linked list of expressions are evaluated left to right and value of right most expression is the value of the combined expression.

For example the statement

```
value = (x = 10, y = 5, x + y);
```

First assigns 10 to x and 5 to y and finally assigns 15 to value. Since comma has the lowest precedence in operators the parenthesis is necessary

The size of Operator

The operator size of gives the size of the data type or variable in terms of bytes occupied in the memory. The operand may be a variable, a constant or a data type qualifier.

Example

```
m = sizeof (sum);
n = sizeof (long int);
```

The size of operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer.

14. Explain bitwise operators in C. (JAN2014,JULY 2014,JUN/JULY 2015)
Soln:

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

15. Explain unary operators in C. (JAN 2014)
Soln:

Unary expressions are formed by combining a unary operator with a single operand. All unary operators are of equal precedence and have right-to-left associativity. The unary operators are:

- Unary minus (-) and unary plus (+)
- Logical negation (!)
- Prefix increment (++) and decrement (- -)
- Address operator (&) and indirection (*)
- Bitwise negation (one's complement) (~)
- Cast operator
- sizeof operator.

16. **Write a c program which takes as input p,t,r compute simple interest and display the result(JUNE/JULY 2015).**

```
Soln:#include<stdio.h>
      #include<conio.h>
       Void main()
{
     int p,t,r,SI;
     printf("enter p , t and r value\n");
     scanf("%d %d %d", &p,&t,&r);
     SI=p*t*r/100
     printf("%d",SI);
 }
```

MODULE II

BRANCHING AND LOOPING

1. List the different decision making statements. Explain any 2 with their syntax & example. (JUN/JULY2013,JAN 2015,JUN/JULY 2015)

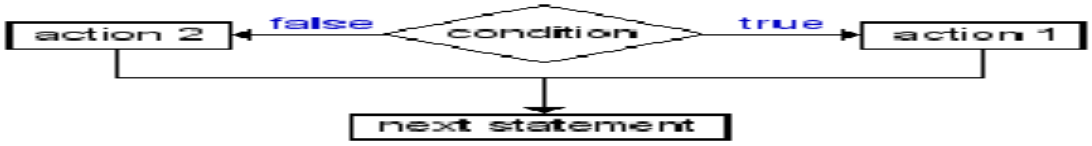
Soln : i)The if-else statement
ii)Nested if statement
iii)Else- if ladder statement
iv)Switch statement

i) **The if-else statement**

Syntax

```
if(condition)
{
statements(s)1
}
else
{
statements(s)2
}
next statement;
```

The if ... else statement allows the program to choose between two alternatives. If the condition is satisfied, the statement(s) after the if are executed, and processing then skips over the else block to the next statement. If the condition is false processing branches to the block of statement(s) after the else. As before the curly braces are not mandatory if the true or false action involves only one statement. However it is advisable to use them for clarity.



For example:-

```
#include <stdio.h>
main()
{
 int x;
printf("input an integer\n");
scanf("%d",&x);
if ((x % 2) == 0)
printf("it is even number\n");
else
printf("it is odd number\n");
}
```

iv)**Switch statement**

Syntax:

```
switch (expr){
case const1:
statement(s);
case const2:
statement(s);
….
case constn:
statement(s);
default:
statement(s);
}
```

This is another form of the multi way decision. It is well structured, but can only be used in certain cases where;

- Only one variable is tested, all branches must depend on the value of that variable. The variable must be an integral type. (int, long, short or char).
- Each possible value of the variable can control a single branch. A final, catch all, default branch may optionally be used to trap all unspecified cases.

In each case the value of $item_i$ must be a constant, variables are not allowed.

The break is needed if you want to terminate the switch after execution of one choice. Otherwise the next case would get evaluated.

The default case is optional and catches any other cases.

For example:-

```
switch (letter)
                    {
                    case `A':
                    case `E':
                    case `I':
                    case `O':
                    case `U':
                                numberofvowels++;
                                break;

                    case ` ':
                                numberofspaces++;
                                break;

                    default:
                                numberofconstants++;
                                break;
                    }
```

In the above example if the value of letter is `A', `E', `I', `O' or `U' then numberofvowels is incremented.

If the value of letter is ` ' then numberofspaces is incremented.

If none of these is true then the default condition is executed, that is numberofconstants is incremented.

2. Write the c-code to find the factorial of a number with all the looping statements.(JAN 2014)

Soln : #include <stdio.h>

```
Void main()
{
        Int I, fact,n;
        Printf("enter the value of n\n");
        Scanf("%d", &n);
        I=1;
        Fact=I;
        While(i<=n)
```

```
                {
                        Fact=fact*I;
                        I++;
                }
                Printf("factorial of a %d is %d\n", n, fact);
                Getch();
        }
#include <stdio.h>
        void main()
        {
                Int I, fact,n;
                Printf("enter the value of n\n");
                Scanf("%d", &n);
                I=1;
                Fact=I;
                Do
                {
                        Fact=fact*I;
                        I++;
                }while(i<=n);
                Printf("factorial of a %d is %d\n", n, fact);
                Getch();
        }


#include <stdio.h>
        Void main()
        {
                Int I, fact,n;
                Printf("enter the value of n\n");
                Scanf("%d", &n);
                Fact=I;
                For(i=1;i<=n;i++)
                {
                        Fact=fact*I;
                }

        Printf("factorial of a %d is %d\n", n, fact);
                getch();
        }
```

3. Explain the use of break & continue statements.(Jan 2014)

Soln : Two keywords that are very important to looping are break and continue. The break command will exit the most immediately surrounding loop regardless of what the conditions of the loop are. Break is useful if we want to exit a loop under special circumstances. For example, let's say the program we're working on is a two-person checkers game.

The basic structure of the program might look like this:

```
while (true)
{
take_turn(player1);
take_turn(player2);
}
```

This will make the game alternate between having player 1 and player 2 take turns. The only problem with this logic is that there's no way to exit the game; the loop will run forever! Let's try something like this instead:

```
while(true)
{
if (someone_has_won() || someone_wants_to_quit() == TRUE)
{break;}
take_turn(player1);
if (someone_has_won() || someone_wants_to_quit() == TRUE)
{break;}
take_turn(player2);
}
```

This code accomplishes what we want--the primary loop of the game will continue under normal circumstances, but under a special condition (winning or exiting) the flow will stop and our program will do something else.

Continue is another keyword that controls the flow of loops. If you are executing a loop and hit a continue statement, the loop will stop its current iteration, update itself (in the case of for loops) and begin to execute again from the top. Essentially, the continue statement is saying "this iteration of the loop is done, let's continue with the loop without executing whatever code comes after me." Let's say we're implementing a game of Monopoly. Like above, we want to use a loop to control whose turn it is, but controlling turns is a bit more complicated in Monopoly than in checkers. The basic structure of our code might then look something like this:

```
for (player = 1; someone_has_won == FALSE; player++)
{
if (player > total_number_of_players)
{player = 1;}
if (is_bankrupt(player))
{continue;}
take_turn(player);
}
```

4. Differentiate between while and do-while statements, with an example for each. (JUN/JULY 2013, JUN/JULY 2015)

Soln: While statement

We use a while statement to continually execute a block of statements while a condition remains true. The following is the general syntax of the while statement.

while (*expression*) {
statement
}

First, the while statement evaluates *expression*, which must return a boolean value. If the expression returns true, the while statement executes the statement(s) in the while block. The while statement continues testing the expression and executing its block until the expression returns false.

Let us consider a simple example, to calculate the factorial of a number

```
#include <stdio.h>
main()
{
  int fact, i,n;
printf("input a number\n");
scanf("%d",&n);
fact=1;
i=1;
while ( i <= n)
{
  fact=fact*i;
  ++i;
}
printf(" factorial = %d\n", fact);
}
```

do –while statement

Syntax:

do {
statement(s)
} while (*expression*);

Instead of evaluating the expression at the top of the loop, do-while evaluates the expression at the bottom. Thus, the statements within the block associated with a do-while are executed at least once.

Each line of a C program up to the semicolon is called a statement. The semicolon is the statement's terminator. The braces { and } which have appeared at the beginning and end of our program unit can also be used to group together related declarations and statements into a ***compound statement*** or a *block*.

In the case of the **while** loop before the ***compound statement*** is carried out the ***condition*** is checked, and if it is *true* the ***statement*** is obeyed one more time. If the ***condition*** turns out to be *false*, the looping isn't obeyed and the program moves on to the next statement. So we can see that the instruction really means *while something or other is true keep on doing the statement*.

In the case of the **do while** loop it will always execute the code within the loop at least once, since the ***condition*** controlling the loop is tested at the bottom of the loop. The **do while** loop repeats the instruction while the ***condition*** is *true*. If the ***condition*** turns out to be *false*, the looping isn't obeyed and the program moves on to the next statement.

Let us consider an example to calculate the factorial using do-while loop.

```
#include < stdio.h>
main()
{
  int n, fact, i;

printf("input an integer\n");
scanf("%d",&n);
fact=1;
i=1;
do
 {
    fact=fact*i;
    ++i;
  }
    while ( i <= n);

printf("factorial = %d\n", fact);
}
```

5. Write a 'C' program to calculate area of circle, rectangle and triangle using switch statement. Area of circle = $\pi * r * r$ Area of rectangle = length * breadth, Area of triangle = 0.5*base*height. (JUN/JULY 2013)

soln :

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

void main()
{
  int  ch,length,breadth,base,height,radius;
   clrscr();
  printff(" 1.area of circle\n 2. Area of rectangle 3. Area of triangle\n");
printf("enter the choice\n");
scanf("%d", &ch);
```

```
 switch(ch)
 {
  case 1: printf("enter the radius\n")
                        scanf("%d", &radius);
                        area =π * r * r
                        printf("%d", area);
                  break;
  case 2: printf("enter the length & breadth\n")
                        scanf("%d%d", & length,&breadth);

                        area= length * breadth
                        printf("%d", area);
                  break;
  case 3: printf("enter the base & height\n")
                        scanf("%d%d", &radius,&height);

                        area=0.5*base*height
                        printf("%d", area);
                  break;
  default:  printf(" Illegal operation \n");
            exit(0);
  }
  getch();
}

  default:  printf(" Illegal operation \n");
            exit(0);
  }
  getch();
}
```

6. Write a C Program to find roots of Quadratic equation. Consider all possible cases of roots.(JUN /JULY 2014,JAN 2015)

```
  #include <stdio.h>
  #include <math.h>
   main()
       {
        int a,b,c,e,f,g;
        float d;
        printf("Finding roots of equation of the form \n a*x^2+b*x+c=0\n");
        printf("Enter the values of constants a, b and c\n");
        scanf("%d%d%d",&a,&b,&c);
        d = b*b-4*a*c;
        if(d==0)
        e=((-1)*b-sqrt(d))/2;
```

```
        printf("Given equation has 2 same roots %f\n",e);
        else if(d<0)
                printf("The given equation has no Real roots\n");
        else if(d>0)
        f=(b-sqrt(d))/2
        g=((-1)*b-sqrt(d))/2
        printf("Roots are %f and %f\n",f,g);
        else if(x<0&&y<0)
                printf("This point lies in the Third Quadrant\n");
}
```

7. Differentiate pre-test and post-test loops. Illustrate your answer with suitable example.(JUN/JULY 2013,JAN 2015)

Soln:

The Difference Between Pretest And Post test. A pretest loop is one in which the block is to be repeated until the specified condition is no longer true, and the condition is tested before the block is executed. A post test loop is one in which the block is to be repeated until the specified condition is no longer true, and the condition is tested after the block is executed.

1)Entry controlled or pre-test loop e.g. While, for.

2) Exit controlled of post-test loop e.g. Do, while.

8.Explain declration and syntax of while and do while loop(DEC/JAN 2014,JUN/JULY2015).

Soln: The following is the general syntax of the while and do while statement.

while (*expression*)
{
statement
}

do –while statement
Syntax:
do {
statement(s)
} while (*expression*);

9. Explain switch statement. (JULY 2014).

Soln: Switch statement

The C `switch` allows multiple choice of a selection of items at one level of a conditional where it is a far neater way of writing multiple `if` statements:

```
switch (expression)        {
                               case item1:
                                                  statement1;
                                                  break;
                               case item2:
                                                  statement2;
                                                  break;
                                                  ⋮
                                                  statementn;
                                                  break;
                               default:
                                                  statement;
                                                  break;
                              } .
```

MODULE III

ARRAYS, STRINGS AND FUNCTIONS

1. **Write a C-program to find GCD of two numbers. (JAN 2014)**

Soln:

```
#include<stdio.h>

int main(){

   int x,y,m,i;

   printf("Insert any two number: ");

   scanf("%d%d",&x,&y);
   if(x>y)
      m=y;
   else
      m=x;

   for(i=m;i>=1;i--){
      if(x%i==0&&y%i==0){
         printf("\nHCF of two number is : %d",i) ;
         break;
```

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

2. Write a 'C' program using function, to compute the sum of N numbers. (JUN/JULY 2013)

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

Void main()
 {
        intm,n,res;
int ADD(int a, int b);
clrscr();
        printf("Enter two numbers\n");
        scanf("%d,%d",&m,&n);
        Res=ADD(m,n);
        printf("sum of two numbers=%d",res);
        getch();
 }
        int ADD(int a, int b);
        {
            int sum;
            sum=a+b;
            return(sum);
        }
```

3. Describe the different ways of passing parameters to a function (JUN/JULY 2013, JAN 2014, JAN 2015)

Soln: Passing parameters to functions:-

Data transfer between functions can be achieved by the following three ways. This is also called by the name parameter passing.

1. pass by value
2. pass by reference
3. pass by address

Pass by value: in pass by value, the values of actual parameters are copied into formal parameters in the called function. Here formal parameters contain only the copy of actual parameters.

Pass by value makes the function more self contained and it protects them against accidental changes. That is changes in the formal parameters, does not make any change in actual parameters.

Pass by reference:- in pass by reference the formal parameters are treated as alternate names
for actual parameters . so, any change in formal parameters imply there is a change in actual
parameter. But this technique is not supported in C. it is supported in C++ language.

Pass by address:-in pass by address when a function is called, the addresses of actual parameters are sent. In this case formal parameters is called function should be declared as
pointer with the same data type as actual parameters. Using these addresses the values of the
actual parameters can be changed indirectly.

4 .What is formatted output? Explain output of integer & real no using an example for each (JUN/JULY 2013)

Formatted input and output using format specifiers:
The function scanf is the input analog of printf, providing many of the same conversion facilities in the opposite direction.

int scanf (char *format, …..)

scanf reads characters from the standard input, interprets them according to the specification in format, and stores the results through the remaining arguments. The format argument is described below; the other arguments, each of which must be a pointer, indicate where the corresponding converted input to be stored.

scanf stops when it exhausts its format string, or when some input fails to match the control specification. It returns as its value the number of successfully matched and assigned input items. This can be used to decide how many items were found . On end of file EOF is returned; note that this is different from 0, which means that the next input character does not match the first specification in the format string. The next call to scanf resumes searching immediately after the last character already converted.

A conversion specification directs the conversion of the next input field. Normally the result is placed in the variable pointed to by the corresponding argument. If assignment suppression is indicated by the * character, however, the input field is skipped; no assignment is made. An input field is defined as a string of non-white space characters; it extends either to the next white space character or until the field width, if specified, is exhausted. This implies that scanf will read across line boundaries to find its input, since new lines are white space. (White space characters are blank, tab, new line, carriage return, vertical tab and form feed).
The general syntax is

int printf (char *format, arg1,arg2……..)

printf converts, formats, and prints its arguments on the standard output under control of the format. It returns the number of characters printed.

The format string contains two types of objects: ordinary characters, which are copied to the output stream, and conversion specifications each of which causes conversion and printing of the next successive argument to printf. Each conversion specification begins with a % and ends with a conversion character. Between the % and the conversion character there may be in order:

- A minus sign, which specifies left adjustment of the converted argument.
- A number that specifies the minimum field width. The converted argument will be printed in a field at least this wide. If necessary it will be padded on the left or right, to make up the field width.
- A period, which separates the field width from the precision.
- A number, the precision, that specifies the maximum number of characters to printed from a string, or the number of digits after the decimal point of a floating point value, or the minimum number of digits for an integer.
- An h if the integer is to be printed as a short, or l if as a long.

Scanf functions:-

The function scanf() is used to read data into variables from the standard input, namely a keyboard. The general format is:

Scanf(format-string, var1,var2,……varn)

Where format-string gives information to the computer on the type of data to be stored in the list of variables var1,var2……varn and in how many columns they will be found

For example, in the statement:

Scanf("%d %d", &p, &q);

The two variables in which numbers are used to be stored are p and q. The data to be stored are integers. The integers will be separated by a blank in the data typed on the keyboard.

A sample data line may thus be:

456 18578

Observe that the symbol &(called ampersand) should precede each variable name. Ampersand is used to indicate that the address of the variable name should be found to store a value in it. The manner in which data is read by a scanf statement may be explained by assuming an arrow to be positioned above the first data value. The arrow moves to the next data value after storing the first data value in the storage location corresponding to the first variable name in the list. A blank character should separate the data values.

The scanf statement causes data to be read from one or more lines till numbers are stored in all the specified variable names.

No that no blanks should be left between characters in the format-string. The symbol & is very essential in front of the variable name.

If some of the variables in the list of variables in the list of variables in scanf are of type integer and some are float, appropriate descriptions should be used in the format-string.

For example:

Scanf("%d %f %e", &a , &b, &c);

Specifies that an integer is to be stored in a, float is to be stored in b and a float written using the exponent format in c. The appropriate sample data line is:

485 498.762 6.845e-12

Printf function:
The general format of an output function is

Printf(format-string, var1,var2…..varn);

Where format-string gives information on how many variables to expect, what type of arguments they are , how many columns are to be reserved for displaying them and any character string to be printed. The printf() function may sometimes display only a message and not any variable value. In the following example:

printf("Answers are given below");

The format-string is:

Answers are given below

And there are no variables. This statement displays the format-string on the video display and there are no variables. After displaying, the cursor on the screen will remain at the end of the string. If we want it to move to the next line to display information on the next line, we should have the format-string:

printf("Answers are given below\n");

In this string the symbol \n commands that the cursor should advance to the beginning of the next line.

In the following example:

printf("Answer x= %d \n", x);

%d specifies how the value of x is to be displayed. It indicates the x is to be displayed as a decimal integer. The variable x is of type int. %d is called the conversion specification and d the conversion character . In the example:

printf("a= %d, b=%f\n", a, b);

the variable a is of type int and b of type float or double. % d specifies that a is to be displayed as an integer and %f specifies that, b is to be displayed as a decimal fraction. In this example %d and %f are conversion specifications and d, f are conversion characters.

5.Write C program to print n fibonacci numbers using function.(JAN 2013, JAN 2014)

Soln:#include<stdio.h>
#include<conio.h>

```
int fib(int n)
{
int a=-1,b=1,c=0,i;
for(i=0;i<n;i++)
{
c=a+b;
printf("%d ",c);
a=b;
b=c;
}
return(0);
}
```

```
void main()
{
int n;
clrscr();
printf("enter any number\n");
scanf("%d",&n);
fib(n);
getch();
}
```

6. Differentiate call by value and call by address. (JUN/JULY 2013, JAN 2015)

Soln: **Call By Value**

1.Creates a new memory loaction for use within the subroutine.The memory is freed once it leaves the subroutine.Changes made to the variable are not affected outside the subroutine.

2.In call by value, both the actual and formal parameters will be created in different memory locations.

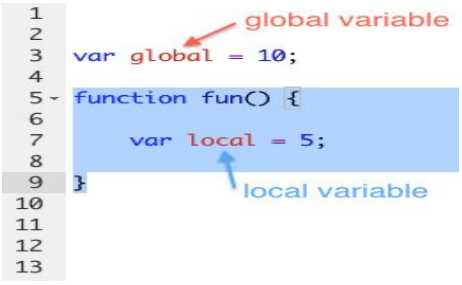
Call By Reference:

1. Passes a pointer to the memory location.Changes made to the variable within the subroutine affects the variable outside the subroutine.

2. They are called by reference both will be created at the same location.

7. Explain scope of local and global variables with sample example(jun/jul 2013)

Soln:Variables defined **outside** a function are […] called **global variables**. variables defined **within** a function are **local variables**. "Scope" is just a technical term for the parts of your code that have access to a variable. In the picture below, the **scope** of the local variable is highlighted blue – it's the function where that var was defined. Any code **inside** that function can access (read and change) this variable. Any code **outside** it can't. It's *local*, so it's invisible from outside.



8. What is a function? Describe with declaration syntax (DEC/JAN 2014,JAN 2015)

Soln: A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

Function Declarations:

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts:

return_type function_name(parameter list);

For the above defined function max(), following is the function declaration:

int max(int num1, int num2);

Parameter names are not important in function declaration only their type is required, so following is also valid declaration:

int max(int, int);

Function declaration is required when you define a function in one source file and you call that function in another file. In such case you should declare the function at the top of the file calling the function.

9. **Explain different function designs. (JULY 2014).**

Soln:

Category of functions:

A function may depend on whether arguments are present or not and whether a value is returned or not. It may belong to one of the following categories.

Category 1: Functions with no arguments and no return values.
Category 2: Functions with arguments and no return values.
Category 3: Functions with arguments and return values.

10. Explain the declaration & initialization of 1-dimensional array, with an example (JUN/JULY 2013, JULY 2014, JAN2015)

Soln : Declaration and initialization of arrays:

The arrays are declared before they are used in the program. The general form of array declaration is

Type variable_name[size];

The type specifies the type of element that will be contained in the array, such as int,float,or char and the size indicates the maximum number of elements that can be stored inside the array.

Example:

Float weight[40]

Declares the weight to be an array containing 40 real elements. Any subscripts 0 to 39 are valid.

Similarly,

Int group1[11];

Decalres the group1 as an array to contain a maximum of 10 integer constants.

The C language treats character strings simply as arrays of characters. The size in a character string represents the maximum number of characters that the string can hold.

For example:

Char text[10];

Suppose we read the following string constant into the string variable text.

"HOW ARE YOU"

Each character of the string is treated as an element of the array text and is stored in the memory as follows.

'H'
'O'
'W'
'A'
'R'
'E'
'Y'
'O'
'U'
'\o'

When the compiler sees a character string, it terminates it with an additional null character. Thus, the element text[11] holds the null character '\o' at the end. When declaring character arrays, we must always allow one extra element space for the null terminator.

Initialization of arrays:

The general form of initialization of arrays is:

Data type array-name[size]={ list of values};

The values in the list are separated by commas.

For example, the statement below shows

int num[3]={2,2,2};

Will declare the variable num as an array of size 3 and will assign two to each element. If the number of values is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically.

For example:

float num1[5]={0.1,2.3,4.5};

Will initialize the first three elements to 0.1,2.3 and 4.5 and the remaining two elements to zero. The word static used before type declaration declares the variable as a static variable.

In some cases the size may be omitted. In such cases, the compiler allocates enough space for all initialized elements. For example, the statement

int count[ ]= {2,2,2,2};

Will declare the counter array to contain four elements with initial values 2.

Character arrays may be initialized in a similar manner. Thus, the statement

char name[ ]={ 'S 'W,'A,'N}

Declares the name to be an array of four characters, initialized with the string "SWAN"

There certain draw backs in initialization of arrays.

1. There is no convenient way to initialize only selected elements.
2. There is no shortcut method for initializing a large number of array elements.

11. Explain the initialization & declaration of C –strings.(JULY 2014,JAN 2015)

Soln: Declaring and initializing string variables:

The general form of string variable is

char string_name[size];

The size determines the number of characters in the string-name.

Some examples are:

char state[10];

char name[30];

When the compiler assigns a character string to a character array, it automatically supplies a null character('\0') at the end of the string.

Character arrays may be initialized when they are declared. C permits a character array to be initialized in either of the following two forms:

Static char state[10]=" KARNATAKA";

Static char state[10]={'K','A','R','N','A','T','A','K','A'\O'};

The reason that state had to be 10 elements long is that the string KARNATAKA contains 10 characters and one element space is provided for the null terminator.

C also permits us to initialize a character array without specifying the number of elements.

For example, the statement

static char string[ ] ={'H', 'E', 'L', 'L', 'O' \O};

12. Write a C-program to read an array of size 'N' & print the array elements(Jan 2014)

Soln:
Program to read and write two dimensional arrays.

```
#include<stdio.h>
main()
{
        int a[10][10];
         int i, j row,col;
        printf("\n Input row and column  of a matrix:");
        scanf("%d %d",  &row,&col);
        for(i=0; i<row;i++)
            for(j=0;j<col;j++)
          scanf("%d", &a[i][j]);
for(i=0;i<row;i++)
{
        for(j=0;j<col;j++)
        printf("%5d", a[i][j]);
printf("\n");
}
```

13. What is an array ?Write a program to print the sum of the two dimensional array and store the result into another array.(JAN 2014,JAN 2015)
A group of related data items that share a common name is called an array. For example, we can define an array name marks to represent a set of marks obtained by a group of students.

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

void main()
{
  int i,j,m,n,c,r,a[5][5];
  void rsum(int a[5][5],int m,int n);
  void csum(int a[5][5],int m,int n);
  void tsum(int a[5][5],int m,int n);

  clrscr();
  printf("Enter matrix size m and n: \n");
  scanf("%d%d",&m,&n);

  printf("Enter the Matrix A:\n");
  for(i=0;i<m;i++)
    for(j=0;j<n;j++)
```

```
        scanf("%d", &a[i][j]);
    printf("\n Enter specified row: ");
  scanf("%d",&r);
  rsum(a,r-1,n);/* since row subscript starts with zero */
  printf("\n Enter specified column:");
  scanf("%d",&c);
  csum(a,m,c-1);/*since column subscript start with zero*/
  tsum(a,m,n);
  getch();
}

/* Function to find sum of the elements of the specified row */
void rsum(int a[5][5],int r,int n)
{
  int j,sum=0;
  for( j=0; j<n; j++ )
        sum = sum + a[r][j];
  printf(" Sum of the elements of the row specified row is %d\n", sum);
}

/* Function to find sum of the elements of the specified column */
void csum(int a[5][5],int m,int c)
{
  int i,sum=0;
  for( i=0; i<m; i++ )
        sum = sum + a[i][c];
  printf(" Sum of the elements of the column specified column is %d\n", sum);
}
```

14. Write a program that accepts a string and checks string is palindrome or not. (JAN 2014, JUN/JULY 2015)

Soln : #include <stdio.h>

```
#include <string.h>

main()
{
  char a[100], b[100];

  printf("Enter the string to check if it is a palindrome\n");
  gets(a);

  strcpy(b,a);
  strrev(b);
```

```
  if( strcmp(a,b) == 0 )
    printf("Entered string is a palindrome.\n");
  else
    printf("Entered string is not a palindrome.\n");

  return 0;
}
```

15. Write a C program to search an element from unsorted list using binary search. (jun/jul 2013)

Soln:

```
#include<stdio.h>
#include<conio.h>
main()
{
  int c, first, last, middle, n, search, array[100];
  printf("Binary Search Program in C\n\n");
  printf("Enter the number of values\n");
  scanf("%d",&n);

  printf("Enter the values in ascending order\n", n);

  for ( c = 0 ; c < n ; c++ )
    scanf("%d",&array[c]);

  printf("Enter the search value\n");
  scanf("%d",&search);

  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while( first <= last )
  {
    if ( array[middle] < search )
      first = middle + 1;
    else if ( array[middle] == search )
    {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if ( first > last )
```

```
    printf("Not found! %d is not present in the list.\n", search);

  getch();
  return 0;
}
```

MODULE 1V

STRUCTURES AND FILE MANAGEMENT

1. What is structure data type? Explain (JAN 2015,JUN/JULY 2015)

Soln: C arrays allow you to define type of variables that can hold several data items of the same kind
but **structure** is another user defined data type available in C programming, which allows you to combine data items of different kinds.

Structures are used to represent a record, suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:

- Title
- Author
- Subject
- Book ID

Defining a Structure

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member for your program. The format of the struct statement is this:

```
struct [structure tag]
{
  member definition;
  member definition;
  ...
  member definition;
} [one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure:

```
struct Books
{
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
```

```
} book;
```

2. **Show how a structure variable is passed as a parameter to a function with an example (JAN 2015)**

Soln: In C, structure can be passed to functions by two 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.

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);
/* function prototype should be below to the structure declaration otherwise compiler shows error */
int 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
    return 0;
}
void Display(struct student stu){
  printf("Output\nName: %s",stu.name);
  printf("\nRoll: %d",stu.roll);
}
```

Output

```
Enter student's name: Kevin Amla

Enter roll number: 149

Output

Name: Kevin Amla

Roll: 149
```

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.

3.Explain the concept array of structures with a suitable C program (JAN 2015)

Soln: C Structure is collection of different datatypes (variables) which are grouped together. Whereas, array of structures is nothing but collection of structures. This is also called as structure array in C.

Example program for array of structures in C:

This program is used to store and access "id, name and percentage" for 3 students. Structure array is used in this program to store and display records for many students. You can store "n" number of students record by declaring structure variable as 'struct student record[n]", where n can be 1000 or 5000 etc.

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

struct student
{
     int id;
     char name[30];
     float percentage;
};
 main()
{
     int i;
     struct student record[2];

     // 1st student's record
     record[0].id=1;
     strcpy(record[0].name, "Raju");
     record[0].percentage = 86.5;

     // 2nd student's record
     record[1].id=2;
     strcpy(record[1].name, "Surendren");
     record[1].percentage = 90.5;

     // 3rd student's record
     record[2].id=3;
     strcpy(record[2].name, "Thiyagu");
     record[2].percentage = 81.5;

     for(i=0; i<3; i++)
     {
         printf("     Records of STUDENT : %d \n", i+1);
         printf(" Id is: %d \n", record[i].id);
         printf(" Name is: %s \n", record[i].name);
```

```
        printf(" Percentage is: %f\n\n",record[i].percentage);
    }
```

4.What is file? Explain fopen() , fclose() functions (JAN 2015,JUN/JULY 2015)

Soln: For C File I/O you need to use a FILE pointer, which will let the program keep track of the file being accessed

fopen

To open a file you need to use the fopen function, which returns a FILE pointer. Once you've opened a file, you can use the FILE pointer to let the compiler perform input and output functions on the file.

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

In the filename, if you use a string literal as the argument, you need to remember to use double backslashes rather than a single backslash as you otherwise risk an escape character such as \t. Using double backslashes \\ escapes the \ key, so the string works as it is expected. Your users, of course, do not need to do this! It's just the way quoted strings are handled in C and C++.

fopen modes: The allowed modes for fopen are as follows:

```
r  - open for reading
w  - open for writing (file need not exist)
a  - open for appending (file need not exist)
r+ - open for reading and writing, start at beginning
w+ - open for reading and writing (overwrite file)
a+ - open for reading and writing (append if file exists)
```

:

fclose

When you're done working with a file, you should close it using the function

```
int fclose(FILE *a_file);
```

fclose returns zero if the file is closed successfully.

An example of fclose is

```
fclose(fp);
```

5.Explain how the input is accepted from a file and displayed(JAN 2015)

Soln: String Input and Output

You need not consider constructing for loops to get and print more than one character at a time. You can make use of similar functions for getting whole strings from standard input and printing them to standard output. They are fgets and fputs respectively.

Prototypes:

```
1.  int fgets(char *restrict s, int n, FILE *restrict stream);

2.  int fputs(char *restrict s, FILE *restrict stream);
```

The first argument you pass to fgets is a character array for putting the string value taken from the file stream. The n value is used as a limiter for the size (in bytes) of input accepted into the argument s. fgets will read from the file stream until it detects an EOF, a press of "Enter", or n – 1 bytes have been read. It returns the value of s or a NULL pointer.

fputs is almost the same, besides that it does not require you to pass a number of bytes that will be accepted. It takes the character array you want to print out, and the stream you want to print it to.

Let us modify our example to read and print a string instead of just a character.

getastring.c:

```
1.  #include <stdio.h>
2.  void main() {
3.  int n = 40;
4.  char ip[n];
5.  //prompt for a sentence and get input.
6.  printf("Type a sentence and press enter.n");
7.  fgets(ip, n, stdin);
8.  //write input value back out to standard out.
9.  fputs(ip, stdout);
10. }
```

6.Explain typedefined structure. (JUN/JULY 2015).

Soln: structure defined using typedef keyword is called typedefined structure.

Typedef struct

```
{ data type ……member 1;
  data type……..member 2;
   datatype……..member 3;
};
```

7.Write a c program to input the following details of N students using structure:
ROLL no: integer, name: string, marks: float, grade: char
Print the names of the students with marks>=70.0% (JUN/JULY 2015).

Solution:

```
#include<stdio.h>
#include<conio.h>
struct student
{
        int   rollno, marks;
        char name[20], grade;
};
void main()
{
        int  i, n,pos,found=0;
        struct  student  s[10];
        char  keyname[20];
        clrscr();
        printf("Enter the number of students\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nenter the student details \n");
                printf("enter the roll number:");
                scanf("%d",&s[i].rollno);
                printf("enter the student name without white spaces:");
                scanf("%s", s[i].name);
                printf("enter the marks : ");
                scanf("%d", &s[i].marks);
                printf("enter the grade : ");
                fflush(stdin);
                scanf("%c",&s[i].grade);
        }
        printf("\nStudent details are \n");
        printf("\nRollno\tName\t\t\tMarks\tGrade\n");
        for(i=0;i<n;i++)

       {
                printf("%d\t%s\t\t%d\t%c\n", s[i].rollno, s[i].name, s[i].marks, s[i].grade);
       }
        printf("\nEnter the keyname to be searched:");
        scanf("%s", keyname);
        for(i=0;i<n;i++)
        {
                if(strcmp(s[i].name, keyname)= =0)
                {
```

```
                printf("\nMarks of the student is : %d", s[i].marks);
                found = 1;
            }
        }
        if(found = =0)
            printf(" Given student name not found\n");
        getch();
}
```

MODULE V

POINTERS AND PREPROCESSORS

1. What is a pointer? Write a program in C to find the sum and mean of all elements in an array. Use pointer technology (JAN 2015,JUN/JULY 2015)

Soln: Although arrays are good things, we cannot adjust the size of them in the middle of the program. If our array is too *small* - our program will fail for large data. If our array is too *big* - we waste a lot of space, again restricting what we can do. The right solution is to build the data structure from small pieces, and add a new piece whenever we need to make it larger. *Pointers* are the connections which hold these pieces together!

2. Pointers in Real Life

In many ways, telephone numbers serve as pointers in today's society. To contact someone, you do not have to carry them with you at all times. *All you need is their number.* Many different people can all have your number simultaneously. *All you need do is copy the pointer.* More complicated structures can be built by combining pointers. *For example, phone trees or directory information.* Addresses are a more physically correct analogy for pointers, since they really are memory addresses.

Linked Data Structures

All the dynamic data structures we will build have certain shared properties. We need a pointer to the entire object so we can find it. Note that this is a pointer, not a cell. Each cell contains one or more data fields, which is what we want to store. Each cell contains a pointer field to at least one ``next'' cell. Thus much of the space used in linked data structures is not data! We must be able to detect the end of the data structure. This is why we need the NIL pointers.

There are four functions defined in c standard for dynamic memmory allocation - calloc, free, malloc and realloc.The prototype of malloc ( ) function is -

void *malloc (size_t number_of_bytes)

The prototype of free ( ) function is -

void free (void *p)

C Source code shown below shows simple method of using dynamic memory allocation elegantly –

```
#include <stdio.h>

#include <stdlib.h>

int main ()

    {


int *p;

p = (int *) malloc ( sizeof (int) ); //Dynamic Memmory Allocation

if (p == NULL) //Incase of memmory allocation failure execute the error handling
code block

{
3.

printf ("\nOut of Memmory");

exit (1);


}

*p = 100;

printf ("\n p = %d", *p); //Display 100 ofcourse.

return 0
}
```

2.What is preprocessor directive ? Explain #define and # include preprocessor directive(JAN 2015)

- **Soln:** The C Preprocessor is not part of the compiler, but is a separate step in the compilation process. In simplistic terms, a C Preprocessor is just a text substitution tool. We'll refer to the C Preprocessor as the CPP.
- All preprocessor lines begin with #. This listing is from Weiss pg. 104. The unconditional directives are:
    - #include - Inserts a particular header from another file
    - #define - Defines a preprocessor macro
    - #undef - Undefines a preprocessor macro

  The conditional directives are:

    - #ifdef - If this macro is defined
    - #ifndef - If this macro is not defined

- #if - Test if a compile time condition is true
- #else - The alternative for #if
- #elif - #else an #if in one statement
- #endif - End preprocessor conditional

Other directives include:

- # - Stringization, replaces a macro parameter with a string constant
- ## - Token merge, creates a single token from two adjacent ones

- Some examples of the above:
  - #define MAX_ARRAY_LENGTH 20

Tells the CPP to replace instances of MAX_ARRAY_LENGTH with 20. Use #define for constants to increase readability. Notice the absence of the ;.

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

Tells the CPP to get stdio.h from **System Libraries** and add the text to this file. The next line tells CPP to get mystring.h from the local directory and add the text to the file. This is a difference you must take note of.

```
#undef MEANING_OF_LIFE
#define MEANING_OF_LIFE 42
```

3. Explain a) Dynamic memory allocation
b) Malloc() function (JAN 2015,JUN/JULY 2015)

Soln: There are four functions defined in c standard for dynamic memmory allocation - calloc, free, malloc and realloc.The prototype of malloc ( ) function is -

void *malloc (size_t number_of_bytes)

The prototype of free ( ) function is -

void free (void *p)

C Source code shown below shows simple method of using dynamic memory allocation elegantly –

```
#include <stdio.h>

#include <stdlib.h>

int main ()

{
```

```
int *p;
p = (int *) malloc ( sizeof (int) ); //Dynamic Memmory Allocation
if (p == NULL) //Incase of memmory allocation failure execute the error handling code block
{

 printf ("\nOut of Memmory");
  exit (1);

}
*p = 100;
printf ("\n p = %d", *p); //Display 100 ofcourse.
return 0;
}
```

4. **What are premetive and non primitive data types (JAN 2015,JUN/JULY 2015)**

Soln: **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: Primitive data type and Non-Primitive data type

PRIMITIVE DATATYPE

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 and Double:** 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 represent logical values either true or false.

NON-PRIMITIVE DATATYPES

The data types that are derived from primary data types are known as non-Primitive data types. These data types are used to store group of values.

The non-primitive data types are

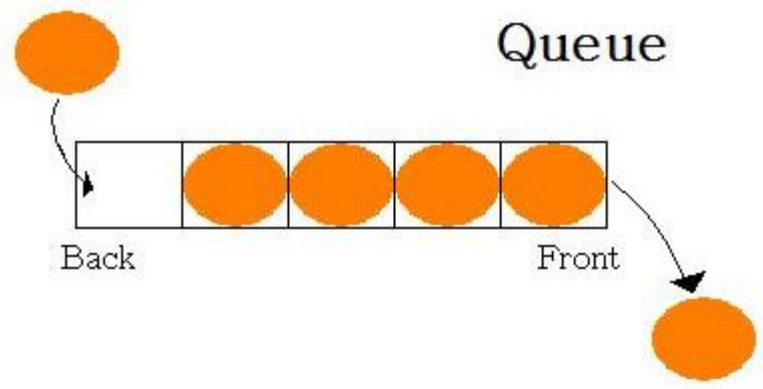
- Arrays
- Structure
- Union
- linked list
- Stacks
- Queue etc

5. Define Queue .Explain along with its application (JAN 2015)

Soln: Queues

The Queue Data Structure:

Queues are data structures that, like the stack, have restrictions on where you can add and remove elements. To understand a queue, think of a cafeteria line: the person at the front is served first, and people are added to the line at the back. Thus, the first person in line is served first, and the last person is served last. This can be abbreviated to **F**irst **I**n, **F**irst **O**ut (**FIFO**).



The cafeteria line is one type of queue. Queues are often used in programming networks, operating systems, and other situations in which many different processes must share resources such as CPU time.

A queue is like a line of people waiting for a bank teller. The queue has a **front** and a **rear**.

When we talk of queues we talk about two distinct ends: the front and the rear. Additions to the queue take place at the rear. Deletions are made from the front. So, if a job is submitted for execution, it joins at the rear of the job queue. The job at the front of the queue is the next one to be executed

Queue Operations

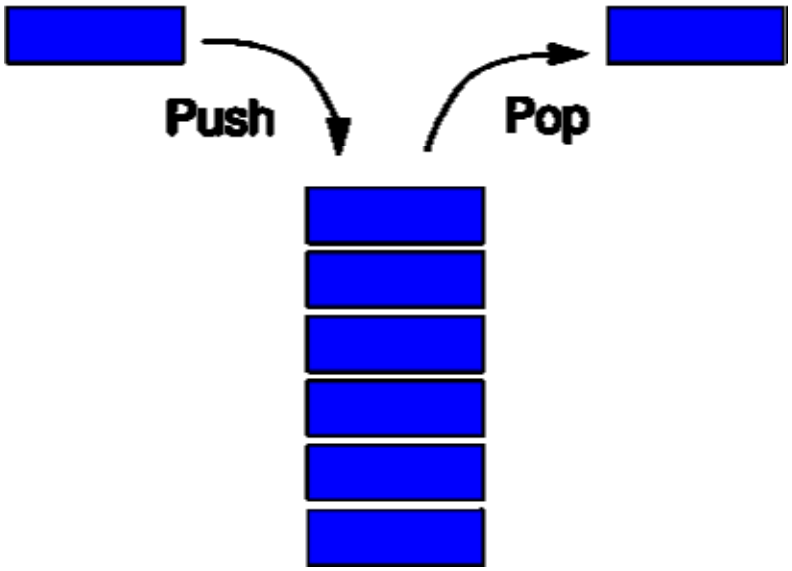
- Queue Overflow
- Insertion of the element into the queue
- Queue underflow
- Deletion of the element from the queue
- Display of the queue

6. Explain

1) Abstract data type
2) Stack
3) Linked list (JAN 2015,JUN/JULY 2015)

The Stack Data Structure:

The stack is a common data structure for representing things that need to maintained in a particular order. For instance, when a function calls another function, which in turn calls a third function, it's important that the third function return back to the second function rather than the first.



One way to think about this implementation is to think of functions as being stacked on top of each other; the last one added to the stack is the first one taken off. In this way, the data structure itself enforces the proper order of calls.

So what's the big deal? Where do stacks come into play? As you've already seen, stacks are a useful way to organize our thoughts about how functions are called. In fact, the "call stack" is the term used for the list of functions either executing or waiting for other functions to return.

In a sense, stacks are part of the fundamental language of computer science. When you want to express an idea of the "first in last out" variety, it just makes sense to talk about it using the common terminology. Moreover, such operations show up an awful lot, from theoretical computer science tools such as a push-down automaton to AI, including implementations of depth-first search.

Stacks have some useful terminology associated with them:

- **Push** To add an element to the stack
- **Pop** To remove an element from the stock
- **Peek** To look at elements in the stack without removing them
- **LIFO** Refers to the last in, first out behavior of the stack
- **FILO** Equivalent to LIFO
- ***Operation on stacks***

Like the data structure by the same name, there are two operations on the stack: push and pop.

PUSH OPERATION:

PUSH is an operation used to add a new element in to a stack. The push operation of a stack is implemented using arrays. When implementing the push operation, overflow condition of a stack is to be checked (i.e., you have to check whether the stack is full or not). If the size of the stack is defined as 5, then it is possible to inset (ie.,add) only 5 elements into the stack. It is not possible

to add any more elements to the stack, since there is no space to accommodate the elements in the array. The following procedure helps you to understand things better.

POP OPERATION:

Pop is an operation used to remove an element from the TOP of the stack. Pop operation of a stack is also implemented using arrays. When implementing he pop operation, underflow condition of a stack is to be checked (i.e., you have to check whether the stack is empty or not). The user should not pop an element from an empty stack. This type of an attempt is illegal and should be avoided. If such an attempt is made, the user should be informed of the underflow condition. The following procedure can help you to understand things better.

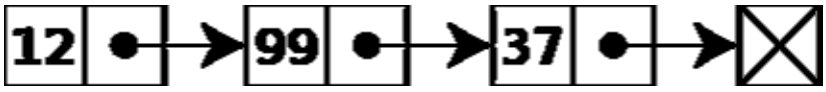
5.7. Applications of Stack

Some important applications using stacks are

- Towers of Hanoi
- Reversing a string
- Evaluation of arithmetic expressions

LINKED LISTS

- In computer science, a linked list is a data structure that consists of a sequence of data records such that in each record there is a field that contains a reference (i.e., a link) to the next record in the sequence.



A linked list whose nodes contain two fields: an integer value and a link to the next node

- Linked lists are among the simplest and most common data structures, and are used to implement many important abstract data structures, such as stacks, queues, hash tables, symbolic expressions, skip lists, and many more.

- The principal benefit of a linked list over a conventional array is that the order of the linked items may be different from the order that the data items are stored in memory or on disk. For that reason, linked lists allow insertion and removal of nodes at any point in the list, with a constant number of operations.

- On the other hand, linked lists by themselves do not allow random access to the data, or any form of efficient indexing. Thus, many basic operations — such as obtaining the last

node of the list, or finding a node that contains a given datum, or locating the place where a new node should be inserted — may require scanning most of the list elements.

- Linked lists can be implemented in most languages. Languages such as Lisp and Scheme have the data structure built in, along with operations to access the linked list. Procedural languages such as C, or object-oriented languages C++, and Java typically rely on mutable references to create linked lists.