PESIT Bangalore South Campus

Hosur road, 1km before Electronic City, Bengaluru -100

Department of Basic Science and Humanities

Continuous Internal Evaluation Test 2

Subject & Code: **Programming in C and Data Structures- 17PCD23**

Sec: F, G, H, I, J

PART 1

1. **A) Explain the differences between while loop and do while loop with an example.**

While loop: it is entry controlled. i.e. condition of the loop is checked before executing statements inside the loop.

The C syntax -

```
while(test-condition)
{
        body of the loop
}
```

If test-condition fails at the beginning, the statements inside the loop will not get executed.

Do-while loop: it is exit controlled. i.e. condition of the loop is checked after executing a statement inside the loop.

It is preferred when the body of the loop has to be executed at least once.

The C syntax -

```
do
{
        body of the loop
}while(test-condition);
```

If test-condition fails at the beginning, the statements inside the loop will get executed once.

Example of both -

```
int a = 11;

do
 {  printf("value of a: %d\n", a);
    a = a + 1;
  }while( a < =20 );
```

```
int a = 11;

while( a < =20 )
 {  printf("value of a: %d\n", a);
    a = a + 1;
  }
```

1. **B) Write C program to generate Fibonacci series of n terms, using while loop.**

```
#include<stdio.h>
void main()
{
        int fib1,fib2,fib3,num,i=3;
        prinft("\nEnter the limit\n");
        scanf("%d",&num);
        fib1=0,fib2=1;
        printf("The first %d numbers in Fibonacci series:\n" , num);
        if(num==1)

                printf("%d\t",fib1);

        else
        {
                printf("%d\t%d\t",fib1,fib2);
                while(i<=num)
                {
                        fib3=fib1+fib2;
                        printf("%d\t",fib3);
                         fib1=fib2;
                        fib2=fib3;
                        i++;
                }
           }
}
```

2. A. Write a program to find number of digits and sum of the digits of a positive number using do while loop.

```
#include<stdio.h>
int main()
{
        int num,temp,sum,count,digit;
        printf("\nEnter the number:");
        scanf("%d",&num);
        sum=0,count=0;
        temp=num;
        do

        {

                digit=num%10;
                sum+=digit;
                num/=10;
                count+=1;
        } while(num!=0);
printf("\n The sum of digits of %d is %d\n", temp,sum);
printf("\n The number of digits of % d is %d\n", temp,count);

return 0;
}
```

2. B) What will be the output of the following programs

```
#include<stdio.h>
void main()
{
int i;
for(i=4;i<=8;i++)
{
  printf("\n%d",i);
  if(i==6)
    continue;
}
}
```

Ans. 4 5 6 7 8

```
#include<stdio.h>
void main()
{
int i;
for(i=4;i<=8;i++)
{
 printf("\n%d",i);
 if(i==6)
   break;
}
}
```

Ans. 4 5 6

PART 2

3. a) What is an array? Explain declaration and initialization of 1 dimensional array.

Ans. An array is a structures of related data items under the same name. The size of an array is same throughout the program. An array is allocated contiguous memory under the same name.

The declaration of 1-d array is -

data-type array-name[array size];

Data-type can be basic data type like int, float, double or char or it can be derived data type. Array-name is the name given to the array variable. Array-size is the number of elements to be stored in the array.

Ex- float average[5];

Array average can store maximum 5 floating point numbers.

The elements of an array can be accessed with index number. The first element in the array is assigned index number 0 and the last element is assigned index number n-1, where n is the size of the array.

Ex. Average [5]. 0 1 2 3 4

v1 V2 V3 V4 V5

In array average, v1 v2 v3 v4 and v5 are the elements stored at index 0 1 2 3 & 4 respectively.

Initialization of array: In initialization, data is stored in an array. Initialization usually comes after declaration. We can combine both of them together also. It can be done in different ways -

- Float average [5]; average [5] = {10.5, 12.0, 25.5, 13.0, 20.0};
  Or,
- Float average [5] = {10.5, 12.0, 25.5, 13.0, 20.0};

Array average is of size 5 and 5 numbers are initialized to the array.

- Float average [ ] = {10.5, 12.0, 25.5, 13.0, 20.0};

Array average's size is not assigned and 5 numbers are initialized to the array.

- Float average [5]; average [5] = {10.5, 12.0, 25.5};
  Or,
- Float average [5] = {10.5, 12.0, 25.5};

Array average is of size 5 and 3 numbers are initialized to the array, remaining 2 places are not initialized with any data.

- Float average [5]; average [5] = {0}.0;
  Or,
- Float average [5] = {0.0};

Array average is of size 5 and 0 is initialized to all 5 places.

3.b) Write a program to evaluate polynomial f(x) = $a_n x^n$+.....+a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0, for a given value of *x* and its coefficients using Horner's method.

```
#include<stdio.h>

#include<math.h>

#define MAX 10

void main()

{

int n,i,a[MAX],x;

float product = 0;

printf("\n Enter the value of n");
```

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

printf("\n Enter the value of coefficient");

for(i =n; i>=0; i--)

        scanf("%d",&a[i]);

printf("\n Enter the value of x");

        scanf("%d",&x);

product=a[n]*x;

//[evaluate the polynomial using horner's method]

for (i=n-1; i>=1; i--)

        product = (product+a[i]) * x;

product = product + a[0];

//[output the value of the given polynomial expression]

printf("\nThe final result is %f", product);

}
```

4. **A Explain row-major order and column-major order in a two dimensional array with example.**

Ans. Row major order and column major order are two ways of storing and accessing data in a 2-d array.

Row-major order - here elements in a 2-d array is stored or accessed one row at a time. First row 1 is stored or accessed, second row 2 is stored or accessed and so on.

```
for(i=0; i<p; i++)  //represents row
        for(j=0; j<q; j++)  //represents column
                scanf("%d", &B[i][j]);
```

Column-major order - here elements in a 2-d array is stored or accessed one column at a time. First column 1 is stored or accessed, second column 2 is stored or accessed and so on.

```
for(j=0; j<n; j++)  //represents column
        for(i=0; i<m; i++)  //represents row
                scanf("%d", &B[i][j]);
```

4.b) Write a C program to search an element in a list of integers using *Binary Search.*

```
#include<stdio.h>
void main()
{
int i, n, a[100], key, flag=0;
int low = 0, high,mid;
printf("Enter the number of elements in array\n");
scanf("%d",&n);
printf("Enter the elements of the array in ascending order\n");
for(i=0;i<n;i++)
        scanf("%d",&a[i]);
printf("\nEnter  key element:");
        scanf("%d",&key);
while (low <= high)
{
    mid = (low + high)/2;
    if (key < a[mid])
        high = mid - 1;
    else if (key > a[mid])
        low = mid + 1;
    else {
        flag=1;
        break;
        }
}
if (flag==1)
        printf("\nSuccess! element found at %d!\n",mid);
else
        printf("\nFailure! element not  found!\n");
```

```
return(0);
}
```

PART 3

5. **Write a C program to sort a list of elements using Bubble sort algorithm. Trace the program for the given elements: 8, 5, 12, 19, 4.**

```
#include<stdio.h>
void main( )
 {
  int a[20], i, j, n, t;
  printf("input the size of the array\n");
  scanf("%d", &n);
  printf("\n Input the elements of the array");
  for (i= 0; i<n; i++)
          scanf("%d", &a[i]);

  printf("sort the elements using bubble sort technique");
  for (i=1; i<n; i++)
          for (j = 0; j< = n-1-i; j++)
              if(a[j+1]<a[j]){
                    t=a[j];
                    a[j]=a[j+1];
                    a[j+1]=t;
              }

          printf("Display the sorted list\n");
          for(i=0; i<n; i++)
                    printf("%d\t", a[i]);
          printf("\n");
}
```

Tracing of Bubble Sort Algorithm –

Input: 8, 5, 12, 19, 4

Here, n=5; number of elements in array

a[] = 8, 5, 12, 19, 4

step1: i=1, j= 0, 1, 2, 3

if(a[j] >a[j+1]) then swap a[j] & a[j+1]

at the end of step 1: a[ ] = 5, 8, 9, 4, 12

step2: i=2, j= 0, 1, 2

if(a[j] >a[j+1]) then swap a[j] & a[j+1]

at the end of step 2: a[ ] = 5, 8, 4, 9, 12

step3: i=3, j= 0, 1
if(a[j] >a[j+1]) then swap a[j] & a[j+1]
at the end of step 3: a[ ] = 5, 4, 8, 9, 12

step4: i=4, j= 0
if(a[j] >a[j+1]) then swap a[j] & a[j+1]
at the end of step 4: a[ ] = 4, 5, 8, 9, 12

The final sorted array is: a[ ] = 4, 5, 8, 9, 12

6. **Write a C program to multiply two matrices. Check the conditions for matrix multiplication. Trace the program with suitable example.**

```
#include<stdio.h>

#define MAX 10

Void main( )
{
 int A[MAX][ MAX], B[MAX][MAX], product[MAX][ MAX];
 int i, j, k, m, n, p, q;
 printf("Enter the size of Matrix 1\n");
 scanf("%d%d", &m,&n);
 printf("Enter the size of Matrix 2\n");
 scanf("%d%d", &p,&q);
 if( n!=p )
 {
        Printf("Matrix Multiplication is not possible\n");
        Return;
 }
 Else
 {
        Printf("Enter Matrix 1\n");
        For(i=0; i<m; i++)
          For(j=0; j<n; j++)
                Scanf("%d", &A[i][j]);
```

```
        Printf("Enter Matrix 2\n");
        For(i=0; i<p; i++)
          For(j=0; j<q; j++)
                Scanf("%d", &B[i][j]);

        For(i=0; i<m; i++)
          For(j=0; j<q; j++)
          {
                product[i][j]=0;
                For(k=0; k<n; k++)
                        product[i][j] = product[i][j] + (A[i][k] * B[k][j]);
          }

        Printf("Display Matrix 1\n");
        For(i=0; i<m; i++)
          For(j=0; j<n; j++)
                printf("%d\t", A[i][j]);
          printf("\n");

        Printf("Display Matrix 2\n");
        For(i=0; i<p; i++)
          For(j=0; j<q; j++)
                printf("%d\t", B[i][j]);
          printf("\n");


        Printf("Display Resultant Matrix\n");
        For(i=0; i<m; i++)
          For(j=0; j<q; j++)
                printf("%d\t", product[i][j]);
          printf("\n");
        }
}
```

PART 4

7. **Define preprocessor directives. Explain macro expansion and file inclusion with suitable examples.**

Ans. Preprocessor directives:

Before a **C** program is compiled by a **compiler**, source code is processed by a program called **preprocessor**. This process is called preprocessing. Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation starts. Commands used in **preprocessor** are called **preprocessor directives** and they begin with "#" symbol.

File Inclusion: #include Preprocessor Directives is used to include an another file inside C Program. It checks for the file in current directory, If path is not mentioned. To include user defined file we use double quote instead of using triangular bracket. If the file is enclosed in triangular bracket, it is searched in standard directory.

```
#include<stdio.h>

#include "myfile.c"

void main()

{

        printf("\nDisplay the external file here");

        display();

}
```

The content of file myfile.c is as follows -

```
int display()

{

  printf("Hello, External function is called\n");

}
```

This program will display the content of the function display( ), i.e. Hello, External function is called.

Macro expansion: #define preprocessor directive is simple substitution macro. It substitute all occurrences of the constant and replace them with the expression. The syntax -

#define identifier value

#define : It is preprocessor directive used for text substitution.

identifier : It is an identifier used in program which will be replaced by value.

value : This is the value to be substituted for identifier.

Example: #define PI 3.14
In C program, wherever PI appears it is replaced with 3.14 before compilation.

#define macro substitution with arguments: Whenever a macro identifier is encountered, the arguments are substituted by the actual arguments from the c program. No data type defined for macro arguments.

Example: #define circumference(r) (2*3.141*(r))

In C program, wherever circumference (r) appears it is replaced by 2*3.141*r.

We can also represent operators with word. For example,

#define larger >

#define equalto = =

Consider the following program,

```
#include<stdio.h>
#define PI 3.14
#define cir(a)  2*3.14*a;

void main()
{
        int r;
        float area, cirf;
        printf("\nEnter the radius\n");
        scanf("%d",&r);
        cirf = cir(r);
        area = PI*r*r;
        printf(Area = %f\n, Circumference = %f\n ", area, cirf);
}
```

In the above program, before compilation PI in area = PI*r*r; will be replaced with 3.14. And cirf = cir(r); will be replaced with cirf = 2*3.14*a; This is known as macro expansion.

8. **Write a *recursive* C function to find the factorial of a number, *n*!, defined by *fact*(*n*)=1, if *n*=0. Otherwise *fact*(*n*)=*n***fact*(*n*-1). Using this function, write a C program to compute Sin(x) using Taylor series approximation given by Sin(x) = x - (x^3/3!) + (x^5/5!) - (x^7/7!) + …….**

```
#include<stdio.h>

#include<math.h>

int factorial(int);

void main( )

{

int n, x, i;

float term, sum=0;

printf("\n Input n (upper limit)");

scanf("%d", &n);

printf("\nInput the value of x");

scanf("%d", &x);

 term=pow(x,1)/factorial(1);

 //[Loop to find term and sum it up]

for(i=3 ; i<=n; i=i+2)

 if(term>0) term = -(pow(x,i)/factorial(i));

 if(term<0) term = pow(x,i)/factorial(i);

  sum+=term;

  printf("\n Display sin(%d) = %f", x, sum);

}
```

```
int factorial(int num)
{

        if(num==0 ) return 1;
        else
                return (num*factorial(num-1));
}
```

PART 5

9. Explain two categories of argument passing techniques in functions with an example.

Ans. There are two type of parameters : Actual parameter and formal parameter.

- *Actual parameters* are parameters as they appear in function calls. We pass the value of the parameter while a function is called.
- *Formal parameters* are parameters as they appear in function declarations and definition.

There are two methods to pass parameters from calling function to called function. They are - call by value and call by reference.

Call by value: The copy of actual parameter values are copied to formal parameters and these formal parameters are used in called function. The changes made on the formal parameters does not affect the values of actual parameters. After the execution control comes back to the calling function, the actual parameter values remains same. Only one value can be returned through return statement.

Call by reference: The address of the actual parameters is copied to formal parameters. This address is used to access the memory locations of the actual parameters in called function. In this method of parameter passing, the formal parameters must be pointer variables. So, the changes made on the formal parameters effects the values of actual parameters. More than one value can be returned.

Example: Call by value

```
#include <stdio.h>

void main(){
int num1, num2, sum;
   int add(int,int) ; // function declaration

   num1 = 10 ;
   num2 = 20 ;

   printf("\nBefore addition: num1 = %d, num2 = %d", num1, num2) ;

   sum = add(num1, num2) ; // calling function

   printf("\n Sum of : num1 = %d\t num2 = %d is %d", num1, num2,sum);

}

int add(int a, int b)  // called function
{
   int temp ;
   temp = a + b;
   return temp;
}
```

Example: Call by Reference

```
#include <stdio.h>

void main(){
int num1, num2 ;
void swap(int *,int *) ; // function declaration

   num1 = 10 ;
   num2 = 20 ;

   printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
   swap(&num1, &num2) ; // calling function

   printf("\nAfter swap: num1 = %d, num2 = %d", num1, num2);

}
void swap(int *a, int *b)  // function definition
{
   int temp ;
   temp = *a ;
   *a = *b ;
```

```
   *b = temp ;
}
```

One more Example on call by reference-

```
#include <stdio.h>
void main(){
int num1, num2, sum;
int add(int *,int *) ; // function declaration

   num1 = 10 ;
   num2 = 20 ;

   printf("\nBefore addition: num1 = %d, num2 = %d", num1, num2) ;

   sum = add(&num1, &num2) ; // calling function

   printf("\n Sum of : num1 = %d\t num2 = %d is %d", num1, num2,sum);

}
int add(int *a, int *b)  // called function
{
   int temp ;
   temp = *a + *b;
   return temp;
}
```

10. Write C user defined functions

(i) to input N integer numbers into a single dimension array.

(ii) to conduct a linear search.

Using these functions, write a C program to accept the N integer numbers & a key integer number and conduct a linear search. Report success or failure in the form of a suitable message.

```
#include <stdio.h>
void readarr(int n, int a[*]); //function to input N numbers to an array
int linear(int n, int a[*], int key); // function to conduct linear search

int main()
{
        int a[20], n, key, lin;
        printf("\nEnter size of the array:");
        scanf("%d",&n);
        printf("\nEnter %d elements:",n);
        readarr(n,a);
```

```
        printf("\nEnter  key element:");
        scanf("%d",&key);
        lin=linear(n,a,key);

        if (lin==1)
                printf("\nSuccess! element found!\n");
        else
                printf("\nFailure! element not  found!\n");
        return 0;
}

void readarr(int n, int a[n])
{
        int i;
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
}

int linear(int n, int a[n], int key)
{
        int i;
        for(i=0;i<n;i++)
        {
                if (a[i]==key)
               return 1;
        }
        return 0;
}
```