U218-144 (T1)

Bansilal Ramnath Agarwal Charitable Trust's
Vishwakarma Institute of Information Technology, Pune
Department of Information Technology
Marking scheme for T1 FDS (**ITUA211708**)

Q.1) a) without pointer: 3M (explanation:1.5M example:1.5M)
with pointer: 3M (explanation:1.5M example:1.5M)

b) 4M: 4 points for difference 2M: examples

c) Opening file: 2M reading content and accessing: 2M.

OR

Q.2) a) Correct C code: 4M Example: 2M

b) Any 3 function syntax with explanation(1M) and example(1M) from fopen,fclose, fseek, ftell, fgetc, fputc

c) 4M for correct code

Q.3) a) 1M for each defn & example

b) 1. Find the frequency count for the following codes:

1. 2+(n+1)+(m+1)+n*m
2. (n+1)+n(n+1)+ 2n^2

c) 2M: explanation 2M:example

OR

Q.4) a) linear DS: 3M (explanation:2M example:1M) non-linear DS: (explanation:2M example:1M)

b) 1)F O(n2) 2) F O(nlogn) 3)T

c) explanation:2M example:2M

U218-144(TL)

Bansilal Ramnath Agarwal Charitable Trust's
Vishwakarma Institute of Information Technology, Pune
Department of Information Technology

Solution for T1 FDS (ITUA211782) 4 3 8

Q.1) `n3=uni(set1,n1,set2,n2,set3);// Calling`

Without pointer definition:

```
     int uni(int set1[],int n1,int set2[],int n2,int set3[])
 {
   int n3,i,j;
   for(i=0;i<n1;i++)
       {
       set3[i]=set1[i];
       }
     n3=n1;
   for(i=0;i<n2;i++)
       {
       for(j=0;j<n3;j++)
           {
             if(set3[j]==set2[i])
              {
      break;
           }
           }
      if(j==n3)
       {
      set3[n3]=set2[i];
      n3++;
       }
      }
     return n3;
}
```

With pointer definition:

```
int uni(int *set1,int n1,int *set2,int n2,int *set3)
 {
   int n3,i,j;
   int* temp=set1;
   for(i=0;i<n1;i++)
       {
       *set3=*set1;
       }
     n3=n1;
  for(i=0;i<n2;i++)
       {
         set1=temp;
      for(j=0;j<n1;j++)
          { printf("comparing %d and %d",*set1,*set2);
            if(*set1==*set2)
             {
                      break;
          }
          set1++;
          }
     if(j==n1)
          {
          *set3=*set2;
```

```
            n3++;
            set3++;
             }
        set2++;
       }
      return n3;
}
```

b) Call by Value: When a function is called by an argument/parameter which is not a pointer the copy of ***the argument*** is passed to the function. Therefore a possible change on the copy does not change the original value of the argument.

```
#include<stdio.h> /*The function calls are Call by Value*/
#define pi 3.14
float area(float);
int main( )
{
float r, a, p;
printf("Enter the radius\n");
scanf("%f",&r);
a = area(r);
printf("The area = %.2f, \n", a);
return 0;
}
float area(float x)
{
return pi*x*x;
}
```

Call by Reference: When a function is called by an argument/parameter which is a pointer (address of the argument) the copy of ***the address of the argument*** is passed to the function. Therefore a possible change on the data at the referenced address change the original value of the argument.

- Call by reference
  - If instead of passing the values of the variables to the called function, we pass their addresses, so that the called function can change the values stored in the calling routine. This is known as "call by reference" since we are *referencing* the variables.
- Pointers can be used to pass addresses of variables to called functions
  - Permanent change
  - Multiple outputs from a function

```
#include<stdio.h> /*The function calls is Call by Reference*/
#define pi 3.14
void area_perimeter(float, float *, float *);
int main( )
```

```
{
float r, a, p;
printf("Enter the radius\n");
scanf("%f",&r);
area_perimeter(r,&a,&p);
printf("The area = %.2f, \n The Perimeter = %.2f", a, p);
return 0;
}
void area_perimeter(float x, float *aptr, float *pptr);
{
*aptr = pi*x*x;
*pptr = 2.0*pi*x;
}
```

c) #include<stdio.h>

```
#include<stdlib.h>
void main( )
{
FILE *fp;
char ch;
fp=fopen(argv[1],"r");
if(fp==NULL)
        {
        printf("cannot open file!!");
        return;
        }

while(1)
        {
        ch=fgetc(fp);
        printf("%c",ch);
        if(ch==EOF)
                break;
        }
        fclose(fp);
}
```

OR

Q.2) a)

```
char name1[30];
printf("Enter the string1 :");
scanf("%s",name1);//input

xpalin(name1);//call
```

```
void xpalin(char *name1)//definition
    {
      int i,len,j,flag;
      len=xstrlen(name1);
      len--;
      flag=0;
      for(i=0,j=len;*(name1+i)!='\0';i++,j--)
      {
            if(*(name1+i)==*(name1+j))
            flag++;
      }
      len++;
      if(flag==len)
      printf("Palindrome");
      else
      printf("Not Palindrome");
}
```

b) fopen():open an existing file or create new file for use.

```
fp=fopen("myfile","r");
```

fclose(): closes a file.

```
fclose(fp);
```

fgetc(): Read a character from file

```
ch=fgetc(fs);
```

fputc(): write a character to file.

```
fputc(ch,ft)
```

fprintf(): write a set of data values to a file.

```
fprintf(f1, "%d %f\n", i, f);
```

fscanf(): read a set of data values from a file.

```
fscanf(f2, "%d%f", &i, &f);
```

fread():reading a record from file

```
fread(&e,sizeof(e),1,fp);
```

fwrite(): writing a record to a file

```
fwrite(&e,sizeof(e),1,fp);
```

fseek(): set the pointer to the desired location in the file

```
fseek (fp,-recsize,SEEK_CUR)
```

ftell():Gives the current position in the file

Position=ftell(fp);

rewind(): set the position to the beginning of the file.

rewind(fp)

c) Write a C program to set diagonal element to 1 and all other elements to 0 and display the Matrix using pointers

```
void change_mat(int **a,int m, int n)
{
      int i,j;
      for(i=0;i<m;i++)
      {
       for(j=0;j<n;j++)
       {
             if(i==j)
              *(*(a+i)+j))=1
             else
             *(*(a+i)+j))=0
       }
      }
}
```

Q.3) a)

1.Data type : Basic construct of programming language

A data type is a term which refers to the kind of data, a variable may hold in PL.

Ex. Data types in C are int ,float, char, double etc.

Eg:- Data type: int has size 2 bytes, range -32768 to 32767, represented as 2's complement

2.Data Object : Data object is a set of elements/ variables used. Instances may or may not be related to each other. The set may be finite or infinite. Variable or constant

Ex. Object A={'p', 2, 34, "IT"}//instances not related

Bank employee-finite and instances are related

Set of natural no-infinite.

3.Abstract data type: An ADT is effective when it models something in the real world. The features of an object that are represented in the ADT are the object's attributes. The actions of an object that are operations of the ADT are the object's behaviors.

Eg- car ADT

4.Data Structure: Implementation of ADT

A data structure is the portion of memory allotted for a mathematical model, in which the required data can be arranged in a proper fashion. How the data will be stored and logical relationship among data elements, and what operations will be

performed on it. The data structure can be defined as the collection of elements and all possible operations which are required for those set of elements.
It is concrete i.e. actual implementation of ADT. It is aggregation of atomic and composite data into a defined relationship. Structure: set of rules (relationship/association) that holds the data together Eg-array, linked list, hash table, tree, graph

5. Data: Raw form. When processed, information. Numerical, char etc.
Atomic data: Cannot be divided into other meaningful pieces of data.
eg- 249, "abc" etc
Composite data: Broken into subfields having meaning
eg- address: 249,"abc"

6. Classification of data structure: Primitive and non-primitive data structures
Linear and non-linear data structures
Static and dynamic data structures
Persistent and ephemeral data structures

b) 1.

```
1. j=n;      .....1
   while(j>=1).....n+1
   {
        product=1;.....n
        for(i=1;i<=m;i++).....n* (m+1)
             {
             product=product*i;.....n*m
             }
        j--;.....n
   }
   Total= 2+(n+1)+(m+1)+n*m

2. for(i=0;i<n;i++){.....n+1
     j=n;.....n
     while(j>=1){.....n* (n+1)
          x=x+j;.....n*n
          j--;.....n*n
          }
   }
   Total=   (n+1)+n(n+1)+ 2n^2
```

c) An algorithm can require different times to solve different problems of the same size.

Eg. Searching an item in a list of n elements using sequential search. → Cost: 1,2,...,n

- *Worst-Case Analysis* –The maximum amount of time that an algorithm require to solve a problem of size n.
  This gives an upper bound for the time complexity of an algorithm.
  Normally, we try to find worst-case behavior of an algorithm.
- *Best-Case Analysis* –The minimum amount of time that an algorithm require to solve a problem of size n.
  The best case behavior of an algorithm is NOT so useful.
- *Average-Case Analysis* –The average amount of time that an algorithm require to solve a problem of size n.
  Sometimes, it is difficult to find the average-case behavior of an algorithm.
  We have to look at all possible data organizations of a given size n, and their distribution probabilities of these organizations.
  Worst-case analysis is more common than average-case analysis.

OR

Q.4) a) Linear DS:

- A list is an ordered list, which consists of different data items connected by means of position or a link/pointer.
- Adjacency relationship
- list is frequently used for similar kinds of data.
- This type of list is also called a list
  - Array & multidimensional array
  - Single linked list: - A single linked list is used to traverse among the nodes in one direction.
  - Doubly linked list: - A double linked list is used to traverse among the nodes in both the directions.

Non-linear DS:

- A list, which doesn't show the relationship of adjacency between elements, is said to be non-linear data structure.
- The frequently used non-linear data structures are
  - Trees : - It maintains hierarchical relationship between various elements
  - Graphs : - It maintains random relationship or point-to-point relationship between various elements.

b) 1)F O(n2) 2) F O(nlogn) 3)T

c) Persistent: Modifications are nondestructive. Each modification creates a new version. All versions coexist.

- Eg-stack
  Stack S1: 1 2 3 4 ..... top=4----------(1)

Push (5) then
S1: 1 2 3 4 5..... top=5----------(2)
Pop()
S1: 1 2 3 4 ..... top=4----------same as (1) Thus, persistent.