Appendix C

Model Question Papers

PAPER I

1. (a) **What is an algorithm? Write the various criteria used for judging an algorithm.**

Algorithm can be defined as a well-defined computational procedure that takes single value or a set of values as inputs and produces a single or multiple values as output. Thus it is a sequence of steps which transforms an input into an output. A single problem may have multiple algorithms. Also, an algorithm is always a language-free computational steps.

In order to compare algorithms, we must have some criteria to measure the efficiency of our algorithms. Suppose an algorithm has m input data. The time and space used by the algorithm are the two main measures for the efficiency of the algorithm. The time is measured by counting the number of key operations—in sorting and searching algorithms, for example the number of comparisons, searches, etc. This is because key operations are so defined that the time for the other operation is much less than or at the most equal to the time for key operations. The space is measured by counting the maximum of memory needed by the algorithm.

The complexity of an algorithm is a function which gives the running time and / or storage space requirement of the algorithm in terms of the size of the input data. Normally, the complexity will refer to the running time of the algorithm.

1. (b) **Write an algorithm to find the roots of quadratic equation for all the cases.**

```
Step 1: Input A, B, C
Step 2: Set D = B * B – 4 * A * C
Step 3: If D > 0 then :
        (a) Set X1 = (-B + SQRT(D))/ 2 * A
        (b) Set X2 = (-B - SQRT(D))/ 2 * A
        (c) Print X1, X2
        Else if D = 0 then:
        (a) Set X = -B / 2 * A
        (b) Print X
        Else:
        Print "Roots are imaginary"
    End If
Step 4: Exit
```

2. (a) Distinguish between the following:

(i) Actual and formal arguments

Actual argument—these are the arguments, which are specified in the function call. For example, consider, a = gcd (x, y). Here x and y are called the actual arguments

Formal arguments—these are the arguments used in the unction declaration. For example, consider int gcd (int m, int n). Here m and n are called the formal arguments

(ii) Global and local variables

Global variable—the variables, which are declared outside all the functions (including the main) is, called as global variable. They are available to all the functions.

Local variable—the variables, which are declared within a function, are called local variables. They can be used only within the function in which it is declared.

(iii) Automatic and static variables

Automatic variable—they are created when the function block is entered and removed when the function is terminated. By default, all the variables are automatic variables.

Static variables—they are variables, which are declared local to a function. However they are available even outside the function in which they are declared.

2. (b) Explain in detail about pass by value and pass by reference. Explain with a sample program.

Pass by value – this is default way of passing parameter to a function. When the called function changes the value of such a variable, it does not get reflected back in the calling function.

Pass by reference – here the parameter is passed as a pointer. This will allow the called function to alter the value of the variable.

In the example below, the argument a is passed by value and b is passed by reference:

```
main()
{
  int a = 10;
  int b = 20;
  printf(“\nBefore the call: a = %d, b = %d”, a, b);
  alterthem(a, &b);
  printf(“\nAfter the call: a = %d, b = %d”, a, b);
}

alterthem(int x, int *y)
{
  x = 50;
  *y = 60;
}
```

3. (a) What is a structure? How it is declared? How it is initialized?

Structures help to organize complex data in a more meaningful manner. It creates a format, which may be used to declare many other variables in that format. For example, we want a single collection to hold an integer and a character variable. This is possible only through a structure. It is declared as below. We will define a structure, which will define details of a book—title of the book, author of the book, price of the book and the number of pages it has. Look at the declaration given as follows:

```
struct book
{
          char title[20];
          char author[15];
          int pages;
          float price;
};
```

It can be initialized as below:

static struct book mybook = {“Data Structure”, “tenenbaum”, 500, 200.0};

3. (b) Define the structure to represent a date. Use your structure that accept two different dates in the format mmdd of the year and do the following: Write a C program to display the month names of both dates.

```
struct mydate
{
  int yyyy;
  int mmdd;
};
main()
{
  struct mydate date1, date2;
  int month1, month2;
  char *months[] = {“”,”Jan”,”Feb”,”Mar”,”Apr”,”May”,”Jun”,”Jul”,
                   “Aug”,”Sep”,”Oct”,”Nov”,”Dec”};
  printf(“\nEnter first date (mmdd) “);
  scanf(“%4d”,&date1.mmdd);
  printf(“\nEnter second date (mmdd) “);
  scanf(“%4d”,&date2.mmdd);
  month1 = date1.mmdd / 100; // take month alone
  month2 = date2.mmdd / 100; // take month alone
  printf(“\nFirst month is %s”,months[month1]);
  printf(“\nSecond month is %s”,months[month2]);
}
```

4. (a) Write a C program to illustrate the use of structure pointer.

```
struct currency
{
  int rupees;
  int paise;
};

main()
{
  struct currency mycurr = {123,25};
  showit(&mycurr);
}
showit(struct currency *myptr)
{
  printf(“\nRupees %d.%d”,myptr->rupees,myptr->paise);
}
```

4. (b) **Explain the effect of the following statements:**
   - (i) **int a, *b = &a** - variable is declared as an integer variable. Variable b is declared as a pointer variable which is assigned to the address of the variable a
   - (ii) **int p, *p** – this will give an error while compiling since the same variable name cannot be used as a normal and pointer variable
   - (iii) **char *s** – this will create a pointer to string variable and name it as s. The variable s points to a string
   - (iv) **a=(float*)&X** – this will take the address of the variable X and assign it to the variable a. The contents of the variable is casted with float. Hence the contents of variable a is a float

5. **Write a program to detect error while opening a file that does not exist.**

```
/**************************************************************************************/
/*                  ERROR HANDLING IN FILE OPERATIONS                  */
/**************************************************************************************/

#include <stdio.h>

main( )
{
  char  *filename;
  FILE  *fp1, *fp2;
  int   i, number;
  fp1 = fopen(“TEST”, “w”);
  for(i = 10; i <=100; i + = 10)
    putw(i, fp1);

  fclose(fp1);

  printf(“\nInput filename\n”);

open_file:
  scanf(“%s”, filename);

  if((fp2 = fopen(filename, “r”)) == NULL)
  {
    printf(“Cannot open the file.\n”);
    printf(“Type filename again,\n\n”);
    goto open_file;
  }
  else

  for(i = 1; i <=20; i++)
  { number = getw(fp2);
    if(feof(fp2))
    {
    printf(“\nRan out of data.\n”);
    break;
  }
    else
      printf(“%d\n”, number);
  }
    fclose(fp2);
}
```

Output

```
Input filename
TETS
Cannot open the file.
Type filename again.

TEST
10
20
30
40
50
60
70
80
90
100

Ran out of data.
```

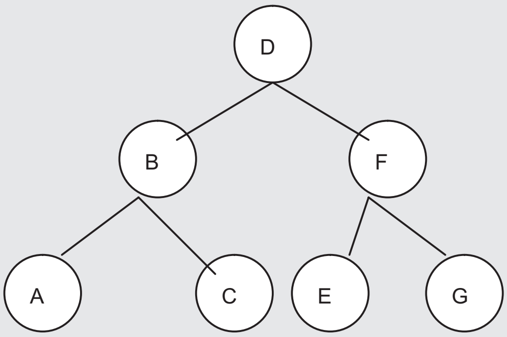
6. Write a program to convert a given infix to postfix expression using stack.

```
#include<stdio.h>
#define MAX 25

int stack[MAX],TOS=-1;

int precedence(char symbol)
{
  int result;
  switch(symbol)
  {
  case ‘(‘:result=0;break;
  case ‘+’:
  case ‘-’:result=1;break;
  case ‘*’:
  case ‘/’:result=2;break;
  case ‘^’:result=3;break;
  }
  return(result);
}
int isoperand(char symbol)
{
  return((symbol >= ‘a’ && symbol <= ‘z’) ||
        (symbol >= ‘A’ && symbol <= ‘Z’) ||
        (symbol >= ‘0’ && symbol <=’9'));
}

int isoperator(char symbol)
{
  int result=0;
  switch(symbol)
  {
```

```
  case ‘+’:
  case ‘-’:
  case ‘*’:
  case ‘/’:
  case ‘^’:result=1;break;
  }
  return(result);
}
main()
{
  char infix[MAX],postfix[MAX];
  int inputpointer,outputpointer;
  inputpointer = outputpointer = -1;
  stack[++TOS]=’(‘;
  printf(“\nEnter the infix expression : “);
  scanf(“%s”,infix);
  while(infix[++inputpointer]!=’\0')
  {
    if(isoperand(infix[inputpointer]))
      postfix[++outputpointer]=infix[inputpointer];
    else if(isoperator(infix[inputpointer]))
      {
            while(precedence(infix[inputpointer]) <=
          precedence(stack[TOS]))
              postfix[++outputpointer]=stack[TOS—];
      stack[++TOS]=infix[inputpointer];
      }
    else if(infix[inputpointer]==’(‘)
        stack[++TOS]=’(‘;
    else if(infix[inputpointer]==’)’)
      {
        while(stack[TOS]!=’(‘)
          postfix[++outputpointer]=stack[TOS—];
        TOS—;
        }
        else
        {
        printf(“invalid symbol “);
        exit(1);
        }
  }
  while(stack[TOS]!=’(‘)
    postfix[++outputpointer]=stack[TOS—];
  postfix[++outputpointer]=’\0';
  printf(“\nThe postfix expression is %s”,postfix);
}
```

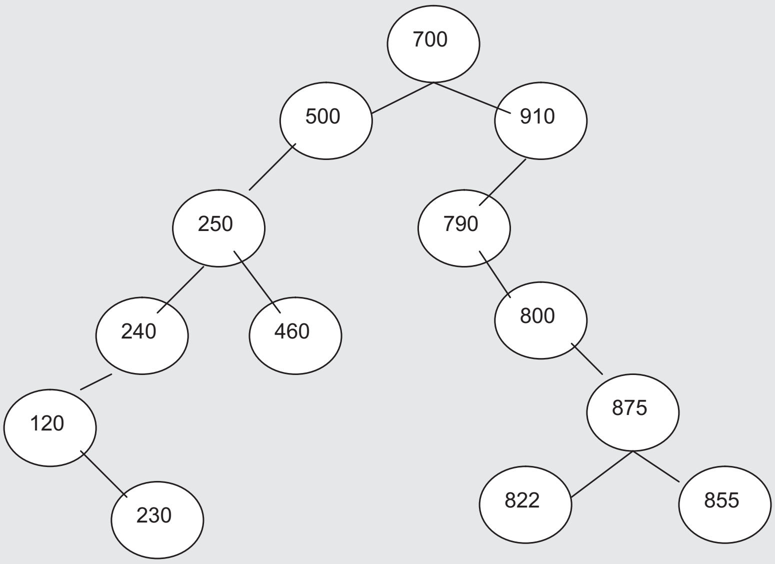
7. (a) Explain the preorder traversal of a binary tree with example.

In a preorder traversal, the root of the binary tree is visited first, followed by the left subtree, followed by the right subtree. Thus, the preorder traversal will finish off the left subtrees before proceeding to the right subtrees.



The preorder traversal for the above tree is the sequence : D-B-A-C-F-E-G

7. (b) Trace through procedure postorder traversal and converse postorder traversal for the following binary tree.



Preorder : 700 – 500 – 250 – 240 – 120 – 230 – 460 – 910 – 790 – 800 – 875 – 822 – 855
Converse preorder : 700 – 910 – 790 – 800 – 875 – 855 – 822 – 500 – 250 – 460 – 240 – 120 – 230

8. (a) Write and explain linear search procedure with a suitable example.

Sequential search is the traditional mechanism of searching for information. It is very simple to understand, but can be very poor in performance when the data gets larger. The process of searching for information in this method is done element by element. Each element is taken from the pool of numbers and compared for the desired number. Thus the search is done sequentially from the first to the last. The general algorithm is as below:

```
Step 1: Start with the number = 1
Step 2: If there are still more numbers in the list
                Compare the current number with the number to be searched
        Else
                Item is not found in the list, stop
```

```
Step 3: If a match is found
            The required number is found, stop
    Else
            Get the next number in the list and proceed to step 2
```

Consider the list – 12, 34, 45, 3, 19 and 20. We need to look for the number 3. The steps are:

Pass 1 : Compare 12 and 3. They are not same. Proceed to next step
Pass 2 : Compare 34 and 3. They are not same. Proceed to next step
Pass 3 : Compare 45 and 3. They are not same. Proceed to next step
Pass 4 : Compare 3 and 3. They are same. Stop

8. (b) Formulate recursive algorithm for binary search with its timing analysis.

```
Step 1: If (bottom > top)
            Element not in the list, stop
Step 2: middle = integer((bottom + top) / 2)
Step 3: If number = list[middle]
            Element is at middle position, Stop
Step 4: If number < list[middle]
            Repeat algorithm between bottom and (middle – 1)
      Else
            Repeat algorithm between (middle + 1) and top
```

In the above algorithm, Step 4 calls the same algorithm recursively—each time, the value of top and bottom changes.

Complexity (timing analysis)—Note that each time the algorithm executes, the number of comparisons to be made is reduced by half. Thus, to locate a number, we may require at most $f(N)$ comparisons where $f(N) = (\log_2 N + 1)$. Here N is the number of elements in the list to be searched.