PAPER 5

1. (a) What are the general characteristics of C?

C is a middle level language. Its characteristics lie between those of problem oriented language such as Fortran, Basic, Pascal and low level Languages. It is a procedural language. It is reliable, efficient, simple and easy to understand.The approach of programming in C is bottom-up. The modularity feature of C language, i.e. breaking the problem into modules called the function makes it easier to organize and maintain.

1. (b) Give and explain the structure of a C program.

The basic structure of a C program is as follows:

```
#include<headerfile1>
#include<headerfile2>
.
.
.#include<headerfilen>

int main()
{

     statement  1;
     statement  2;
     .
     .
     statement  n;

     return value;
}
```

Any C program is a set of functions. One and the must among these is main(). Empty Parenthesis after main() is necessary.

Every function contains one or more C statements or instructions which are enclosed within a pair of braces. Each instruction should end with a;.

If the program uses library functions, i.e. functions which are defined elsewhere, the program must include that library file using a #include preprocessor directive at the beginning of the program. If the keyword 'void' is not preceded with the main(), the main() should have a return statement at the end of program.

1. (c) Write a C program to print the Pascal's triangle.

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

printf("Enter how many lines do you want");
 scanf("%d",&n);

 a[1][1]=1;
 printf("%5d",a[1][1]);
 a[2][1]=1;a[2][2]=2;a[2][3]=1;
 printf("%d %d %d",a[2][1],a[2][2],a[2][3]);

 for(i=3;i<=n;i++)
 {
   a[i][1]=1;
   printf("%d",a[i][1]);
  j=2;c=3;
  while(j<=i)
  {
     a[i][j]=a[i-1][c-1]+a[i-1][c-2];
    printf("%5d",a[i][j]);
   c=c+1;
   j=j+1;
  }
 a[i][j]=1;
 printf("%d",a[i][j]);
 }
}
```

2. (a) In what way is an array different from an ordinary variable?

An ordinary variable is a single variable while an array is a set of variables which are arranged in contiguous memory locations. If we want to store a number of items of similar types we use array instead of a variable. Each ordinary variable used in any C program has different names but in case of array each variable in the set has same name, i.e. the name of the array. Each variable in the set or group is referred by its position in the sequence of variables in the array.

2. (b) What conditions must be satisfied by the entire elements of a given array?

The only condition that must be satisfied by the entire elements of an array is that all the elements of the array must be of the same data types of the declared array.

2. (c) What are subscripts? How are they written? What restrictions apply to values that can be assigned to subscripts?

A subscript is the number which indicates the position of an individual variable within an array. It is used to refer to individual array element. It is also called an index. The following syntax is used to refer any variable in a array using subscript:

arrname[subscript];

where arrname is the name of the array and subscript is the index or position of that element in the array.

In case of C, the subscript starts with 0, i.e. the first variable in the array is given a position number 0, the second is given the position 1 and so on. In this way the position of the last variable will always be one less than the size of the array. Hence the values of the subscript can lie only within the range 0 to (size-1).

2. (d) What is the advantage of defining an array size in terms of a symbolic constant rather than a fixed integer quantity?

The main advantage of defining the size of an array in terms of a symbolic constant say a preprocessor directive like #define or using a constant variable is that if we are referring the size of the variable at number of places in the program and at a later time if we need to change the size of the array, we need not change the size by searching it and changing at each place. Even if we change the size given by the symbolic constant, it will get replaced at each place. Hence there are fewer chances of errors because of mismatching of size.

2. (e) Write a C program to find the largest element in an array.

/* Program to find the largest element in an array */

```
#include<stdio.h>
#include<conio.h>
#define maxsize 50
int main()
{
    int arr[maxsize], maxNo;
    int i;
    for(i=0;i<maxsize;i++)
    {
        //Accept the array elements
        printf("\n Enter the next element");
        scanf("%d", &arr[i]);

        maxNo=arr[0];
        //find the largest no
        for(i=0;i<maxsize;i++)
        {
            if (arr[i]>maxNo)
                maxNo=arr[i];
        }

        printf("\n The largest element is  %d ", maxNo);

        return 0;
}
```

3. (a) Explain the different ways of passing structure as arguments in functions.

Structure variable can be passed to the function using one of the following ways.

1. By passing individual member of the structure.
2. By passing the entire structure variable.

Following program demonstrates the way of passing individual member of the structure.

```
/*** Passing individual structure elements ****/
main()
{
        struct book
        {
            char  name[25];
            char  name[25];
            int callno;
        };
        stuct book b1={"Lets us C","YPK",101};

        display(b1.name,b1.author,b1.callno);

    }

    void display (char *s, char *t ,int n)
     {
        printf("\n%s %s %d",s,t,n);
    }
```

3. (b) Write a C program to illustrate the method of sending an entire structure as a parameter to a function.

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

struct book
{
    char bname[20];
    float price;
};

//prototype
void display(struct book);

int main()
{

struct book b1={" C Primer", 250.00};
display (b1);
return 0;
}

Void display (struct book b2)
{
    Puts(b2.bname);
```

```
        puts(b2.price);
    }
```

4. (a) How to use pointer variables in expressions? Explain through examples.

Pointer variables can be used in expressions in place of ordinary variables. For this purpose, we need to prefix the pointer variable with an asterisk. For example, consider the following.

```
int i = 100;
int *p = &i;
```

Now we can use *p wherever we would use i. For example, suppose that we want to multiply the value of i by 2 to compute the value of another variable j. Then, the following two expressions are the same.

```
int j = i * 2;
int j = *p * 2;
```

Note that we have simply substituted i with *p. Similarly, we can have the following equivalent if conditions.

```
if (i > 100)
  printf ("Hello");
if (*p > 100)
  printf ("Hello");
```

4. (b) Write a 'C' program to illustrate the use of pointers in arithmetic operations.

Pointer variables can be used in arithmetic operations in place of ordinary variables. For this purpose, we need to prefix the pointer variable with an asterisk. For example, consider the following:

```
int i = 100;
int *p = &i;
```

Now we can use *p wherever we would use i. For example, suppose that we want to find out if i is an even or odd number. Then the following two arithmetic operations are equivalent.

```
int j = i / 2;
if (j == 0)
  printf ("Even");
else
  printf ("Odd");

int j = *p / 2;
if (j == 0)
  printf ("Even");
else
  printf ("Odd");
```

Another example follows.

/ Program of addition of two variables using pointer.*/

```
#include<stdio.h>
#include<conio.h>
main()
{

    int var1,var2,result;
    var1=20;
```

```
    var2=30;

    add(&var1 ,&var2,&result);

    printf("Result of addition is %d ",result);
}

void add( int *a , int *b, int *c)
{

    *c= *a+*b;

}
```

5. Explain in detail about file handling functions in C.

If we want to store data in a file in the secondary memory, we must specify certain things about the file, to the operating system. They include:

1. Filename.
2. Data structure.
3. Purpose.

Filename is a string of characters that make up a valid filename for the operating system. It may contain two parts, a primary name and an optional period with the extension. Examples:

```
Input.data
store
PROG.C
Student c
Text.out
```

Data structure of a file is defined as **FILE** in the library of standard I/O function definitions. Therefore, all files should be declared as type **FILE** before they are used. **FILE** is a defined data type.

When we open a file, we must specify what we want to do with the file. For example, we may write data to the file or read the already existing data.

Following is the general format for declaring and opening a file:

```
FILE        *fp;
fp = fopen("filename", "mode");
```

The first statement declares the variable **fp** as a "pointer to the data type **FILE**". As stated earlier. **FILE** is a structure that is defined in the I/O library. The second statement opens the file named *filename* and assigns an identifier to the **FILE** type pointer **fp**. This pointer which contains all the information about the file is subsequently used as a communication link between the system and the program.

The second statement also specifies the purpose of opening this file. The *mode* does this job. *Mode* can be one of the following:

r open the file for reading only.

w open the file for writing only.

a open the file for appending (or adding) data to it.

Note that both the filename and mode are specified as strings. They should be enclosed in double quotation marks.

When trying to open a file, one of the following things may happen:

1. When the mode is 'writing' a file with the specified name is created if the file does not exist. The contents are deleted, if the file already exists.
2. When the purpose is 'appending', the file is opened with the current contents safe. A file with the specified name is created if the file does not exist.
3. If the purpose is 'reading', and if it exists, then the file is opened with the current contents safe; otherwise an error occurs.

Consider the following statements:

```
FILE *p1, *p2;
p1      = fopen("data", "r");
p2      = fopen("results", "w");
```

The file **data** is opened for reading and **results** is opened for writing. In case, the **results** file already exists, its contents are deleted and the file is opened as a new file. If **data** file does not exist, an error will occur.

Many recent compilers include additional modes of operation. They include:

r+ The existing file is opened to the beginning for both reading and writing.

w+ Same as **w** except both for reading and writing.

a+ Same as **a** except both for reading and writing.

We can open and use a number of files at a time. This number however depends on the system we use.

6. Declare two stacks of varying length in a single array. Write C program for push1, push2, pop1, pop2 to manipulate the two stacks.

```
#define MAXSIZE 50
struct  stack
{
      int arr[50];
      int top1;
      int top2;
};

int pop1()
{
            if (top1= = -1)
            {
            printf("Stack1 underflow");
            return -1;
            }
            return (arr[top1--]);
      }

      int pop2()
      {
            if (top2= =top1)
            {
                  printf("\n Stack2 underflow");
```

```
            return -1;
        }
        return (arr[top2--]);
    }

    void push1(int num)
    {
        if(top1= =top2-1)
        {
            printf("\n Stack1 overflow");
            return;
        }
        arr[++top1] = num;
    }

    void push2(int num)
    {
        if (top2= =MAXSIZE -1)
        {
            printf("Stack2 underflow");
            return;
        }
        arr[++top2] = num;
    }
```

7. Write a function in C for a singly linked list, which reverses the direction of links

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

struct node
{
    int data;
    struct node *next;
};
struct node *root;
void addnode()
{
    int item;
    struct node *newnode,*p;

    printf("\n Enter the item");
    scanf("%d", &item);
    newnode=(struct node*) malloc(sizeof(struct node));
    newnode->data=item;
    newnode->next=NULL;
    if(root==NULL)
    {
        root=newnode;
        return;
    }
```

```
        p=root;
        while(p->next!=NULL)
            p=p->next;
        p->next=newnode;
}
void display()
{
        struct node*p=root;
        while(p!=NULL)
        {
            printf("\n %d ",p->data);
            p=p->next;
        }
}
void reverse()
{
        struct node *p,*oldnode=NULL,*newnode;
        p=root;
        while(p!=NULL)
        {
            newnode=p;
            p=p->next;
            if(oldnode==NULL)
            {
                oldnode=newnode;
                oldnode->next=NULL;
            }
            else
            {
                newnode->next=oldnode;
                oldnode=newnode;
            }
        }
        root=oldnode;
}
void main()
{
        int i;
        clrscr();
        for(i=1;i<=10;i++)
        {
            addnode();
        }
        printf("\nbefore reverse");
        display();
        printf("\n After reverse");
        reverse();
        display();
        return 0;
}
```

8. Explain the algorithm of selection sort and give a suitable example.

```
void selectionSort(int numbers[], int array_size)
{
    int i, j;
    int min, temp;

    for (i = 0; i < array_size-1; i++)
    {
        min = i;
        for (j = i+1; j < array_size; j++)
        {
        if (numbers[j] < numbers[min])
        min = j;
        }
        temp = numbers[i];
        numbers[i] = numbers[min];
        numbers[min] = temp;
    }
}
```