INDEX

/* To check whether the given number is prime or not */

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

int main()
{
        int i,n;

        clrscr();
        printf ("Enter the Number ...\n");
        scanf ("%d", &n);

        for (i=2; i<=n/2; i++)
                if (n%i==0)
                        break;

        if (i > n/2)
                printf ("Given Number is Prime");
        else
                printf ("Given Number is Not Prime");

        getch();
        return 0;
}
```

/* To find the Reverse of a given Number */

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

int main()
{
        int n, d, s=0;

        clrscr();
        printf ("Enter the Number ...\n");
        scanf ("%d", &n);

        while (n > 0)
        {
                d = n % 10;
                s = s * 10 + d;
                n = n / 10;
        }

        printf ("Reverse is %d", s);
        getch();
        return 0;
}
```

```
                    /*      Matrix Addition, Generalized */

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

int main()
{
        int a[3][3], b[3][3], c[3][3];
        int i, j, m, n, p, q;

        clrscr();
        printf ("Enter the Size of the Matrix-1 ...\n");
        scanf ("%d%d", &m, &n);

        printf ("Enter the Size of the Matrix-2 ...\n");
        scanf ("%d%d", &p, &q);

        if (m==p && n==q)
        {
                printf ("Enter the Elements of the Matrix-1 ...\n");
                for (i=0; i<m; i++)
                        for (j=0; j<n; j++)
                                scanf ("%d", &a[i][j]);

                printf ("Enter the Elements of the 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<n; j++)
                                c[i][j] = a[i][j] + b[i][j];

                printf ("The Resultant Matrix is ...\n");
                for (i=0; i<m; i++)
                {
                        for (j=0; j<n; j++)
                                printf ("%6d", c[i][j]);

                        printf ("\n");
                }
        }
        else
                printf ("Matrix Addition is Not Possible ...");

        getch();
        return 0;
}
```

```
                    /*      Matrix Multiplication  */

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

int main()
{
        int a[3][3], b[3][3], c[3][3];
        int i, j, k, m, n, p, q;

        clrscr();
        printf ("Enter the Size of the Matrix-1 ...\n");
        scanf ("%d%d", &m, &n);

        printf ("Enter the Size of the Matrix-2    ...\n");
        scanf ("%d%d", &p, &q);

        if (n==p)
        {
                printf ("Enter the Elements of the Matrix-1 ...\n");
                for (i=0; i<m; i++)
                        for (j=0; j<n; j++)
                                scanf ("%d", &a[i][j]);

                printf ("Enter the Elements of the 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++)
                        {
                                c[i][j] = 0;
                                for (k=0; k<n; k++)
                                        c[i][j] = c[i][j] + a[i][k] * b[k][j];
                        }

                printf ("The Resultant Matrix is ...\n");
                for (i=0; i<m; i++)
                {
                        for (j=0; j<q; j++)
                                printf ("%6d", c[i][j]);

                        printf ("\n");
                }
        }
        else
                printf ("Matrix Multiplication is Not Possible ...");

        getch();
        return 0;
}
```

/* To find the Transpose, for a given Matrix */

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

int main()
{
        int a[4][4], b[4][4];
        int i,j,m,n;

        clrscr();
        printf ("Enter the size of Matrix ...\n");
        scanf ("%d%d", &m, &n);

        printf ("Enter the elements of Matrix, one by one ...\n");
        for (i=0; i<m; i++)
                for (j=0; j<n; j++)
                        scanf ("%d", &a[i][j]);

        for (i=0; i<n; i++)
                 for (j=0; j<m; j++)
                          b[i][j] = a[j][i];

        printf ("The Transpose Matrix is ...\n");
        for (i=0; i<n; i++)
        {
                for (j=0; j<m; j++)
                        printf ("%6d", b[i][j]);

                printf ("\n");
        }

        getch();
        return 0;
}
```

/* To print the fibonacci series using recursive function */

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

int fib(int);

int main()
{
        int i,n;

//      clrscr();
        printf ("Enter the Number of Terms ...\n");
        scanf ("%d", &n);

        printf ("Fibonacci Series ...\n");
        for (i=1; i<=n; i++)
                printf ("%d ", fib(i));

        getch();
        return 0;
}

int fib(int m)
{
        if (m==1)
                return (0);
        else
                if (m==2)
                        return (1);
                else
                        return (fib(m-1) + fib(m-2));
}
```

/* call-by-value*/

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

void swap(int,int);

int main()
{
        int a=3, b=5;

//      clrscr();
        printf ("Before Calling The Function ...\n");
        printf ("a = %d                 b = %d\n\n", a, b);

                swap(a,b);

        printf ("After Calling The Function ...\n");
        printf ("a = %d                 b = %d\n\n", a, b);
        getch();
        return 0;
}

void swap(int p, int q)
{
        int temp;

        temp = p;
        p = q;
        q = temp;
}
```

output

Before Calling The Function ...
a = 3 b = 5

After Calling The Function ...
a = 3 b = 5

```
                    /*      call-by-reference    */

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

void swap(int *, int *);

int main()
{
        int a=3, b=5;

//      clrscr();
        printf ("Before Calling The Function ...\n");
        printf ("a = %d                 b = %d\n\n",a,b);

                swap(&a, &b);

        printf ("After Calling The Function ...\n");
        printf ("a = %d                 b = %d\n\n",a,b);
        getch();
        return 0;
}

void swap(int *p, int *q)
{
        int temp;

        temp = *p;
        *p = *q;
        *q = temp;
}

output
        Before Calling The Function ...
        a = 3           b = 5

        After Calling The Function ...
        a = 5           b = 3
```

/* To find the length of a given string */

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

int main()
{
        char p[20];
        int i;

        clrscr();
        printf ("Enter a String ...\n");
        gets(p);

        for (i=0; p[i]!='\0'; i++)
        {       }

        printf ("String Length = %d\n", i);
        getch();
        return 0;
}
```

/* To convert a String in to Lower Case */

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

int main()
{
	char p[20], q[20];
	int i;

	clrscr();
	printf ("Enter a String ...\n");
	scanf("%s", p);

	for (i=0; p[i]!='\0'; i++)
	{
		if (p[i]>='A' && p[i]<='Z')
			q[i] = p[i] + 32;
		else
			q[i] = p[i];
	}

	q[i] = '\0';

	printf ("Lower Case is : %s", q);
	getch();
	return 0;
}
```

/* To convert a String in to Upper Case */

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

int main()
{
	char p[20], q[20];
	int i;

	clrscr();
	printf ("Enter a String ...\n");
	scanf("%s", p);

	for (i=0; p[i]!='\0'; i++)
	{
		if (p[i]>='a' && p[i]<='z')
			q[i] = p[i] - 32;
		else
			q[i] = p[i];
	}

	q[i] = '\0';

	printf ("Upper Case is : %s", q);
	getch();
	return 0;
}
```

```
                    /*      To sort a list of strings  */

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

int main()
{
        char p[30][20];
        char temp[20];
        int i, j, n;

        clrscr();
        printf ("Enter the No. of Strings ...\n");
        scanf ("%d", &n);

        printf ("Enter the Strings, One by one ...\n");
        for (i=0; i<n; i++)
                scanf ("%s", p[i]);

        for (i=0; i<n-1; i++)
                for (j=i+1; j<n; j++)
                        if ( strcmp(p[i],p[j]) > 0)
                        {
                                strcpy(temp,p[i]);
                                strcpy(p[i],p[j]);
                                strcpy(p[j],temp);
                        }

        printf ("The Sorted List of Strings ...\n");
        for (i=0; i<n; i++)
                printf ("%s\n", p[i]);

        getch();
        return 0;
}
```

/* Passing the address of a structure as parameter */

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

struct circle
{
        int r;
        float a,c;
};

void areacirc(struct circle *);

int main()
{
        struct circle s;

        clrscr();
        printf ("Enter the Radius ...\n");
        scanf ("%d", &s.r);

                areacirc(&s);

        printf ("  Area = %8.3f\n", s.a);
        printf ("Circum = %8.3f\n", s.c);

        getch();
        return 0;
}

void areacirc(struct circle *p)
{
        (*p).a = 22/7.0 * (*p).r * (*p).r;
        (*p).c = 2 * 22/7.0 * (*p).r;
}
```

```
                /*      Passing the address of a structure as parameter     */

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

struct circle
{
        int r;
        float a,c;
};

void areacirc(struct circle *);

int main()
{
        struct circle s;

        clrscr();
        printf ("Enter the Radius ...\n");
        scanf ("%d", &s.r);

                areacirc(&s);

        printf ("  Area = %8.3f\n", s.a);
        printf ("Circum = %8.3f\n", s.c);

        getch();
        return 0;
}

void areacirc(struct circle *p)
{
        p->a = 22/7.0 * p->r * p->r;
        p->c = 2 * 22/7.0 * p->r;
}
```

/* To sort a given List - using DMA */

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

int main()
{
	int *a;
	int i,j,n,temp;

	clrscr();
	printf ("Enter the No. of Elements ...\n");
	scanf ("%d", &n);

		a = malloc(n*sizeof(int));

	if (a==NULL)
		printf ("Suficient Memory Space is Not Available.");
	else
	{
		printf ("Enter the Elements, One by one ...\n");
		for (i=0; i<n; i++)
			scanf ("%d", &a[i]);

		for (i=0; i<n-1; i++)
			for (j=i+1; j<n; j++)
				if (a[i] > a[j])
				{
					temp = a[i];
					a[i] = a[j];
					a[j] = temp;
				}

		printf ("The Sorted List is ...\n");
		for (i=0; i<n; i++)
			printf ("%d ", a[i]);

		free(a);
	}

	getch();
	return 0;
}
```

/* Linear Search */

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

int LinSrch(int a[], int n, int ele)
{
	int i;

	for (i=0; i<n; i++)
		if (a[i]==ele)
			return (i);

	return -1;
}

int main()
{
	int a[10];
	int data, i, n, pos;
	char ch;

	system("cls");
	printf ("Enter the No. of Elements ...\n");
	scanf ("%d", &n);

	printf ("Enter the Elements, One by one ...\n");
	for (i=0; i<n; i++)
	scanf ("%d", &a[i]);

	do
	{
		system("cls");
		printf ("\n\nEnter the Element, to be Searched : ");
		scanf ("%d", &data);

		pos = LinSrch(a, n, data);

		if (pos==-1)
			printf ("Element Not Found in the List.");
		else
			printf ("Element Found at - %d", pos);

		printf ("\nOne More Search [y/n] : ");
		ch = getche();
	}
```

```
        while (ch!='n');
        return 0;
}
```

/* Binary Search */

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

int BinSrch (int a[], int n, int ele)
{
	int l,h,mid;
	l=0;
	h=n-1;

	while(l<=h)
	{
		mid = (l+h)/2;

		if (ele < a[mid])
			h = mid-1;
		else
			if (ele > a[mid])
				l = mid+1;
			else
				return mid;
	}

	return -1;
}

int main()
{
	int a[10];
	int data, i, n, pos;
	char ch;

	system ("cls");
	printf ("Enter the No. of Elements ...\n");
	scanf ("%d", &n);

	printf ("Enter the Elements, One by one ...\n");
	for (i=0; i<n; i++)
		scanf ("%d", &a[i]);

	do
	{
		system ("cls");
		printf ("\n\nEnter the Element, to be Searched : ");
		scanf ("%d", &data);
```

```
                pos = BinSrch(a, n, data);

            if (pos==-1)
                printf ("Element Not Found in the List.");
            else
                printf ("Element Found at - %d", pos);

            printf ("\nOne More Search [y/n] : ");
            ch = getche();
        }
        while (ch!='n');
        getch();
        return 0;
}
```

/* Bubble Sort */

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

void BubSort(int a[], int n)
{
	int temp;
	int i,j;

	for (i=n-1; i>0; i--)
		for (j=0; j<i; j++)
			if (a[j] > a[j+1])
			{
				temp = a[j];
				a[j] = a[j+1];
				a[j+1] = temp;
			}
}

int main()
{
	int a[20];
	int i, n;

	sstem("cls");
	printf ("Enter the No. of Elements, of the List : ");
	scanf ("%d", &n);

	printf ("Enter the Elements, One by one ...\n");
	for (i=0; i<n; i++)
		scanf ("%d", &a[i]);

		BubSort(a,n);

	printf ("After Sorting The List is...\n");
	for (i=0; i<n; i++)
		printf ("%d ", a[i]);

	return 0;
}
```

/* Selection Sort - Maximum Method */

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

void SelSort(int a[], int n)
{
	int i,j,pos;
	int max, temp;

	for (i=n-1; i>=0; i--)
	{
		max = a[0];
		pos=0;

		for (j=1; j<=i; j++)
			if (a[j] > max)
			{
				max = a[j];
				pos = j;
			}

			temp = a[pos];
			a[pos] = a[i];
			a[i] = temp;
	}
}

int main()
{
	int a[20];
	int i, n;

	system ("cls");
	printf ("Enter the No. of Elements, of the List : ");
	scanf ("%d", &n);

	printf ("Enter the Elements, One by one ...\n");
	for (i=0; i<n; i++)
		scanf ("%d", &a[i]);

		SelSort(a,n);

	printf ("After Sorting The List is...\n");
	for (i=0; i<n; i++)
		printf ("%d ", a[i]);
```

```
        return 0;
}
```

/* Insertion Sort */

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

void InsSort(int a[], int n)
{
	int i, j;
	int ele;

	for (i=1; i<n; i++)
	{
		ele = a[i];

		for (j=i-1; j>=0 && a[j]>ele; j--)
			a[j+1] = a[j];

		a[j+1] = ele;
	}
}

int main()
{
	int a[20];
	int i, n;

	clrscr();
	printf ("Enter the No. of Elements, of the List : ");
	scanf ("%d", &n);

	printf ("Enter the Elements, One by one ...\n");
	for (i=0; i<n; i++)
		scanf ("%d", &a[i]);

		InsSort(a,n);

	printf ("After Sorting The List is...\n");
	for (i=0; i<n; i++)
		printf ("%d ", a[i]);

	getch();
	return 0;
}
/*
```

/* Merge Sort */

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

void Merge(int a[], int p, int q, int r)
{
	int *c;
	int i=p, j=q+1, k=0;

		c = (int *) malloc ((r-p+1) * sizeof(int));

	while (i<=q && j<=r)
	{
		if (a[i] < a[j])
			c[k++] = a[i++];
		else
			c[k++] = a[j++];
	}

	while (i<=q)
		c[k++] = a[i++];

	while (j<=r)
		c[k++] = a[j++];

	for (i=p; i<=r; i++)
		a[i] = c[i-p];

	free(c);
}

void MergeSort(int a[], int p, int r)
{
	int q;

	if (p < r)
	{
		q = (p+r)/2;
		MergeSort(a,p,q);
		MergeSort(a,q+1,r);
		Merge(a,p,q,r);
	}
}

int main()
{
	int a[20];
	int i, n;

	clrscr();
	printf ("Enter the No. of Elements, of the List : ");
	scanf ("%d", &n);
```

```
	printf ("Enter the Elements, One by one ...\n");
	for (i=0; i<n; i++)
		scanf ("%d", &a[i]);

		MergeSort(a,0,n-1);

	printf ("After Merging The List is...\n");
	for (i=0; i<n; i++)
		printf ("%d ", a[i]);

	getch();
	return 0;
}
```

/* Quick Sort */

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

void QuickSort(int a[], int l, int r)
{
	int i,j;
	int pivot, temp;

	if (l>=r)
		return;

	i = l;
	j = r+1;
	pivot = a[l];

	while(1)
	{
		do
		{
			i++;
		}
		while(a[i] < pivot);

		do
		{
			j--;
		}
		while(a[j] > pivot);

		if (i>=j)
			break;

		temp = a[i];
		a[i] = a[j];
		a[j] = temp;
	}

	a[l] = a[j];
	a[j] = pivot;
	QuickSort(a, l,j-1);
	QuickSort(a, j+1,r);
}

int main()
{
	int a[20];
	int i, n;

	clrscr();
	printf ("Enter the No. of Elements, of the List : ");
	scanf ("%d", &n);

	printf ("Enter the Elements, One by one ...\n");
```

```
	for (i=0; i<n; i++)
		scanf ("%d", &a[i]);

		QuickSort(a,0,n-1);

	printf ("After The List is...\n");
	for (i=0; i<n; i++)
		printf ("%d ", a[i]);

	getch();
	return 0;
}
```

```
                        /*     Heap Sort     */

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

int a[20];
int n;

void HeapSort(int m)
{
       int i,j;
       int temp;

       for (i=2; i<=m; i++)
              for (j=i; j>1; j=j/2)
              {
                     if (a[j-1] > a[j/2-1])
                     {
                            temp = a[j-1];
                            a[j-1] = a[j/2-1];
                            a[j/2-1] = temp;
                     }
                     else
                            break;
              }

       printf ("%d ", a[0]);

       for (i=0; i<m-1; i++)
              a[i] = a[i+1];

       m--;
       if (m>0)
              HeapSort(m);
}

int main()
{
       int i;

       clrscr();
       printf ("Enter the No. of Elements ...\n");
       scanf ("%d", &n);

       printf ("Enter the Elements, One by one ...\n");
       for (i=0; i<n; i++)
              scanf ("%d", &a[i]);

       printf ("\nThe Sorted elements are ...\n");
       HeapSort(n);
       getch();
       return 0;
}
```

/* Single Linked List */

```
#define bool int
#define True 1
#define False 0

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

typedef struct node Node;

struct node
{
	int data;
	Node *next;
};

Node *First=NULL;
int n=0;

void Create()
{
	int ele;
	Node *p, *Last=NULL;

	printf ("Enter the Data items, at last 0 ...\n");
	while (1)
	{
		scanf ("%d", &ele);
		if (ele==0)
			break;

		p = (Node *) malloc(sizeof(Node));
		p->data = ele;
		p->next = NULL;

		if (First==NULL)
			First = p;
		else
			Last->next=p;

		Last=p;
		n++;
	}
}
```

```
void Display()
{
	Node *p;

	if (First==NULL)
		printf ("SLL is Empty.\n");
	else
	{
		printf ("SLL is ...\n");
		for (p=First; p!=NULL; p=p->next)
			printf ("--->%d", p->data);
	}
}

void Insert(int k, int x)
{
	Node *p, *t;
	int cnt=1;

	if (k<1 || k>n+1)
		printf ("Position is out of Range ...\n");
	else
		if (k==1)
		{
			p = (Node *) malloc(sizeof(Node));
			p->data = x;
			p->next = First;
			First = p;
			n++;
		}
		else
		{
			for (p=First; cnt<k-1; p=p->next, cnt++)
			{	}

			t = (Node *) malloc(sizeof(Node));
			t->data = x;
			t->next = p->next;
			p->next = t;
			n++;
		}
}

int Delete(int k)
{
	Node *p, *t;
	int cnt=1;
	int x;
```

```
	if (First==NULL)
	{
		printf ("SLL is Empty.");
		return -1;
	}
	else
		if (k<1 || k>n)
		{
			printf ("Position is out of Range ...\n");
			return -1;
		}
		else
			if (k==1)
			{
				x = First->data;
				p = First;
				First = First->next;
				free(p);
				n--;
				return x;
			}
			else
			{
				for (p=First; cnt<k-1; p=p->next, cnt++)
				{	}
				t = p->next;
				x = t->data;
				p->next = t->next;
				free(t);
				n--;
				return x;
			}
}

int main()
{
	int ch, pos;
	int ele;

	do
	{
		system("cls");
		printf ("Single Linked List Operations Menu\n");
		printf ("----------------------------------\n");
		printf ("1. To Create the SLL.\n");
		printf ("2. Insert an element at a position\n");
		printf ("3. Delete an element at a position\n");
```

```
		printf ("4. Display\n");
		printf ("5. Exit\n\n");

		printf ("Your Choice [1..8] : ");
		scanf ("%d", &ch);

		switch(ch)
		{
			case 1 :	Create();
					break;

			case 2 :	printf ("Enter the Position & Element ...\n");
					scanf ("%d%d", &pos, &ele);
					Insert(pos,ele);
					break;

			case 3 :	printf ("Enter the Position : ");
					scanf ("%d", &pos);
					ele = Delete(pos);

					if (ele!=-1)
						printf ("%d is Deleted.\n", ele);
					break;
			case 4 :	Display();
					break;

			case 5 :	break;

			default:	printf ("Invalid Choice.");
					getch();
		}
		getch();
	}
	while (ch!=5);
	return 0;
}
```

/* Circular Linked List */

```
#define bool int
#define true 1
#define false 0

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

typedef struct node Node;

struct node
{
	int data;
	Node *next;
};

Node *First=NULL;
int n=0;

void Create()
{
	int ele;
	Node *p, *Last=NULL;

	printf ("Enter the Data items, at last 0 ...\n");
	while (1)
	{
		scanf ("%d", &ele);
		if (ele==0)
			break;

		p = (Node *) malloc(sizeof(Node));
		p->data = ele;
		if (First==NULL)
		{
			p->next = p;
			First = p;
		}
		else
		{
			p->next = First;
			Last->next=p;
		}
		Last=p;
		n++;
```

```
        }
}

void Display()
{
        Node *p;

        if (First==NULL)
                printf ("CLL is Empty.\n");
        else
        {
                printf ("CLL is ...\n");
                p=First;
                do
                {
                        printf ("--->%d", p->data);
                        p = p->next;
                }
                while(p!=First);
        }
}

void Insert(int k, int x)
{
        Node *p, *Last, *t;
        int cnt=1;

        if (k<1 || k>n+1)
                printf ("Position is out of Range ...\n");
        else
                if (k==1)
                {
                        for (Last=First; Last->next!=First; Last=Last->next)
                        {       }
                        p = (Node *) malloc(sizeof(Node));
                        p->data = x;
                        p->next = First;
                        First = p;
                        Last->next = First;
                        n++;
                }
                else
                {
                        for (p=First; cnt<k-1; p=p->next, cnt++)
                        {       }
                        t = (Node *) malloc(sizeof(Node));
                        t->data = x;
                        t->next = p->next;
```

```
                        p->next = t;
                        n++;
                }
}

int Delete(int k)
{
        Node *p, *Last, *t;
        int cnt=1;
        int x;

        if (First==NULL)
        {
                printf ("SLL is Empty.");
                return -1;
        }
        else
                if (k<1 || k>n)
                {
                        printf ("Position is out of Range ...\n");
                        return -1;
                }
                else
                        if (k==1)
                        {
                                for (Last=First; Last->next!=First; Last=Last->next)
                                {       }
                                x = First->data;
                                p = First;
                                First = First->next;
                                free(p);
                                Last->next = First;
                                n--;
                                return x;
                        }
                        else
                        {
                                for (p=First; cnt<k-1; p=p->next, cnt++)
                                {       }
                                t = p->next;
                                x = t->data;
                                p->next = t->next;
                                free(t);
                                n--;
                                return x;
                        }
}
```

```
int main()
{
	int ch, pos;
	int ele;

	do
	{
		system("cls");
		printf ("Circular Linked List Operations Menu\n");
		printf ("-----------------------------------\n");
		printf ("1. To Create the SLL.\n");
		printf ("2. Insert an element at a position\n");
		printf ("3. Delete an element at a position\n");
		printf ("4. Display\n");
		printf ("5. Exit\n\n");

		printf ("Your Choice [1..8] : ");
		scanf ("%d", &ch);

		switch(ch)
		{
			case 1 :	Create();
					break;

			case 2 :	printf ("Enter the Position & Element ...\n");
					scanf ("%d%d", &pos, &ele);

					Insert(pos,ele);
					break;

			case 3 :	printf ("Enter the Position : ");
					scanf ("%d", &pos);

					ele = Delete(pos);
					if (ele!=-1)
						printf ("%d is Deleted.\n", ele);

					break;

			case 4 :	Display();
					break;

			case 5 :	break;

			default:	printf ("Invalid Choice.");
					getch();
		}
		getch();
```

```
        }
        while (ch!=5);
        return 0;
}
```

/* Double Linked List */

```
#define bool int
#define true 1
#define false 0

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

typedef struct node Node;

struct node
{
	Node *prev;
	int data;
	Node *next;
};

Node *First=NULL, *Last=NULL;
int n=0;

void Create()
{
	int ele;
	Node *p, *Last=NULL;

	printf ("Enter the Data items, at last 0 ...\n");
	while (1)
	{
		scanf ("%d", &ele);
		if (ele==0)
			break;

		p = (Node *) malloc(sizeof(Node));
		p->prev = Last;
		p->data = ele;
		p->next = NULL;

		if (First==NULL)
			First = p;
		else
			Last->next=p;

		Last=p;
		n++;
	}
```

```
}

void Display()
{
	Node *p;

	if (First==NULL && Last==NULL)
		printf ("DLL is Empty.\n");
	else
	{
		printf ("DLL is ...\n");
		for (p=First; p!=NULL; p=p->next)
			printf ("--->%d", p->data);
	}
}

void Insert(int k, int x)
{
	Node *p, *t;
	int cnt=1;

	if (k<1 || k>n+1)
		printf ("Position is out of Range ...\n");
	else
		if (k==1)
		{
			p = (Node *) malloc(sizeof(Node));
			p->prev = NULL;
			p->data = x;
			p->next = First;
			First->prev = p;
			First = p;
			n++;
		}
		else
			if (k==n+1)
			{
				p = (Node *) malloc(sizeof(Node));
				p->prev = Last;
				p->data = x;
				p->next = NULL;
				Last->next = p;
				Last = p;
				n++;
			}
			else
			{
				for (p=First; cnt<k-1; p=p->next, cnt++)
```

```
                                {       }
                                t = (Node *) malloc(sizeof(Node));
                                t->prev = p;
                                t->data = x;
                                t->next = p->next;
                                p->next = t;
                                t->next->prev = t;
                                n++;
                        }
}

int Delete(int k)
{
        Node *p, *t;
        int cnt=1;
        int x;

        if (First==NULL && Last==NULL)
        {
                printf ("DLL is Empty.");
                return -1;
        }
        else
                if (k<1 || k>n)
                {
                        printf ("Position is out of Range ...\n");
                        return -1;
                }
                else
                        if (k==1)
                        {
                                x = First->data;
                                p = First;
                                First = First->next;
                                free(p);
                                First->prev = NULL;
                                n--;
                                return x;
                        }
                        else
                                if (k==n)
                                {
                                        x = Last->data;
                                        p = Last;
                                        Last = Last->prev;
                                        free(p);
                                        Last->next = NULL;
                                        n--;
```

```
                return x;
        }
        else
        {
                for (p=First; cnt<k; p=p->next, cnt++)
                {       }
                x = p->data;
                p->next->prev = p->prev;
                p->prev->next = p->next;
                free(p);
                n--;
                return x;
        }
}

int main()
{
        int ch, pos;
        int ele;

        do
        {
                system("cls");
                printf ("Single Linked List Operations Menu\n");
                printf ("----------------------------------\n");
                printf ("1. To Create the SLL.\n");
                printf ("2. Insert an element at a position\n");
                printf ("3. Delete an element at a position\n");
                printf ("4. Display\n");
                printf ("5. Exit\n\n");

                printf ("Your Choice [1..8] : ");
                scanf ("%d", &ch);

                switch(ch)
                {
                        case 1 :        Create();
                                        break;

                        case 2 :        printf ("Enter the Position & Element ...\n");
                                        scanf ("%d%d", &pos, &ele);

                                        Insert(pos,ele);
                                        break;

                        case 3 :        printf ("Enter the Position : ");
                                        scanf ("%d", &pos);
```

```
                                        ele = Delete(pos);

                                        if (ele!=-1)
                                                printf ("%d is Deleted.\n", ele);

                                        break;

                        case 4 :        Display();
                                        break;

                        case 5 :        break;

                        default:        printf ("Invalid Choice.");
                                        getch();
                }
                getch();
        }
        while (ch!=5);
        return 0;
}
```

/* Linear Stack - Stack using Arrays */

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

#define Bool int
#define True 1
#define False 0
#define Capacity 5

int S[Capacity];
int Top = -1;

Bool IsEmpty()
{
        return (Top==-1);
}

Bool IsFull()
{
        return (Top==Capacity-1);
}

void Push(int ele)
{
        if (IsFull())
                printf ("Stack is Full.");
        else
                S[++Top] = ele;
}

int Pop()
{
        if (IsEmpty())
        {
                printf ("Stack is Empty.");
                return (-1);
        }

        return S[Top--];
}

void Display()
{
        int i;

        if (IsEmpty())
                printf("Stack is Empty");
        else
        {
                printf("Stack is....\n");
                for (i=Top; i>=0; i--)
                        printf("%d\n", S[i]);
        }
}
```

```
int main()
{
	int ch;
	int data;

	clrscr();
	do
	{
		clrscr();
		printf ("Stack Operations Menu\n");
		printf ("----------------------\n");
		printf ("1.Push\n");
		printf ("2.Pop\n");
		printf ("3.Display\n");
		printf ("4.Exit\n\n");

		printf("Enter the choice [1..4] : ");
		scanf("%d", &ch);

		switch(ch)
		{
			case 1  :	printf("Enter the element : ");
					scanf("%d", &data);
					Push(data);
					break;

			case 2  :	data = Pop();
					if (data!=-1)
						printf("%d deleted from the Stack", data);

					break;

			case 3 :	Display();
					break;

			case 4 :	break;
			default:	printf("Invalid choice.");
		}

		getch();
	}
	while(ch!=4);
	return 0;
}
```

/* Linear Queue - Queue using Arrays */

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

#define Bool int
#define True 1
#define False 0

#define Capacity 5

int Q[Capacity];
int Front=0, Rear=-1;

Bool IsEmpty()
{
	return (Front==Rear+1);
}

Bool IsFull()
{
	return (Rear==Capacity-1);
}

void Add(int ele)
{
	if(IsFull())
		printf ("Queue is Full.");
	else
		Q[++Rear] = ele;
}

int Delete()
{
	if(IsEmpty())
	{
		printf ("Queue is Empty.");
		return (-1);
	}

	return Q[Front++];
}

void Display()
{
	int i;

	if (IsEmpty())
		printf ("Queue is Empty");
	else
	{
		printf ("Queue is....\n");
		for (i=Front; i<=Rear; i++)
			printf("%d ", Q[i]);
 }
```

```
}

int main()
{
	int ch;
	int data;

	clrscr();

	do
	{
		clrscr();

		printf ("Queue Operations Menu\n");
		printf ("----------------------\n");
		printf ("1.Add\n");
		printf ("2.Delete\n");
		printf ("3.Display\n");
		printf ("4.Exit\n\n");

		printf("Enter the choice [1..4]   :          ");
		scanf("%d", &ch);

		switch(ch)
		{
			case 1 : printf ("Enter the element : ");
				scanf ("%d", &data);
				Add(data);
				break;

			case 2 : data = Delete();
				if (data!=-1)
					printf ("%d deleted from the Queue", data);
				break;

			case 3 : Display();
				break;

			case 4 : break;

			default: printf ("Invalid choice");
	  }
	  getch();
	}
	while(ch!=4);

	return 0;
}
```

/* Linked Queue - Queue using SLL */

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

#define Bool int
#define True 1
#define False 0

typedef struct node  Node;

struct node
{
        int data;
        Node *next;
};

Node *Front=NULL, *Rear=NULL;

Bool IsEmpty()
{
        return(Front==NULL);
}

Bool IsFull()
{
        return (False);
}

void Add(int ele)
{
        Node *p;

        if (IsFull())
                printf("Queue is Full.");
        else
        {
                p = (Node*) malloc(sizeof(Node));
                p->data = ele;
                p->next = NULL;

                if (Front==NULL)
                        Front = p;
                else
                        Rear->next = p;

                Rear = p;
        }
}

int Delete()
{
        Node *p;
```

```
	int x;

	if (IsEmpty())
	{
		printf ("Queue is Empty");
		return(-1);
	}
	else
	{
		x = Front->data;
		p = Front;
		Front = Front->next;
		free(p);
		return(x);
	}
}

void Display()
{
	Node *p;

	if (IsEmpty())
		printf ("Queue is Empty");
	else
	{
		printf("Queue is ...\n");
		for (p=Front; p!=NULL; p=p->next)
			printf ("--->%d",p->data);
	}
}

int main()
{
	int ch;
	int data;

	clrscr();

	do
	{
		clrscr();

		printf ("Queue Operations Menu\n");
		printf ("----------------------\n");
		printf ("1.Add\n");
		printf ("2.Delete\n");
		printf ("3.Display\n");
		printf ("4.Exit\n\n");

		printf ("Enter the choice [1..4]  :          ");
		scanf ("%d", &ch);

		switch(ch)
		{
			case 1 : printf ("Enter the element : ");
```

```
                    scanf ("%d", &data);
                    Add(data);
                    break;

            case 2 : data = Delete();
                    if (data!=-1)
                        printf ("%d deleted from the Queue", data);
                    break;

            case 3 : Display();
                    break;

            case 4 : break;

            default: printf ("Invalid choice");
        }
        getch();
    }
    while(ch!=4);
    return 0;
}
```

/* Linked Stack - Stack using SLL */

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

#define Bool int
#define True 1
#define False 0

typedef struct node Node;
struct node
{
	int data;
	Node *next;
};

Node *Top = NULL;

Bool IsEmpty()
{
	return (Top==NULL);
}

Bool IsFull()
{
	return (False);
}

void Push(int ele)
{
	Node *p;

	if (IsFull())
		printf ("Stack is Full.");
	else
	{
		p = (Node*) malloc(sizeof(Node));
		p->data = ele;
		p->next = Top;
		Top = p;
	}
}

int Pop()
{
	Node *p;
	int x;
```

```
	if (IsEmpty())
	{
		printf("Stack is Empty");
		return (-1);
	}
	else
	{
		x = Top->data;
		p = Top;
		Top = Top->next;
		free(p);
		return(x);
	}
}

void Display()
{
	Node *p;

	if (IsEmpty())
		printf ("Stack is Empty");
	else
	{
		printf("Stack is ...\n");

		for (p=Top; p!=NULL; p=p->next)
			printf("--->%d\n",p->data);
	}
}

int main()
{
	int ch;
	int data;

	clrscr();
	do
	{
		clrscr();
		printf ("Stack Operations Menu\n");
		printf ("----------------------\n");
		printf ("1.Push\n");
		printf ("2.Pop\n");
		printf ("3.Display\n");
		printf ("4.Exit\n\n");

		printf("Enter the choice [1..4] : ");
```

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

		switch(ch)
		{
			case 1 :	printf("Enter the element : ");
					scanf("%d", &data);
					Push(data);
					break;

			case 2 :	data = Pop();
					if (data!=-1)
						printf("%d deleted from the Stack", data);
					break;

			case 3 :	Display();
					break;

			case 4 :	break;

			default:	printf("Invalid choice.");
		}
		getch();
		reurn 0;
	}
	while(ch!=4);
	return 0;
}
```

/* To covert an Infix Expression into Postfix */

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

/*------------------------- Stack ------------------------------------------------------------*/

#define MaxSize 20

int Top=-1;
char opstk[MaxSize];

void Push (char op)
{
        opstk[++Top] = op;
}

char Pop ()
{
        return(opstk[Top--]);
}

int IsEmpty ()
{
        return (Top==-1);
}

/*------------------------- End Of Stack -----------------------------------------------------*/

int IsOperand(char ch)
{
        return ( ch>='A' && ch<='Z' || ch>='a' && ch<='z' );
}

int PrcdVal (char op)
{
        switch(op)
        {
                case '^' : return (1);
                case '*' :
                case '/' : return (2);
                case '+' :
                case '-' : return (3);
                default : printf ("Invalid Operator.\n");
                return (0);
        }
}

int Prcd (char op1, char op2)
{
```

```
	int r1, r2;

	if (op1=='(')
		return(0);

	if ( op2=='(' && op1!=')')
		return(0);

	if ( op2==')' && op1!='(')
		return(1);

	r1 = PrcdVal(op1);
	r2 = PrcdVal(op2);

	return (r1<=r2);
}

int main()
{
	char s[40], ops[40];
	int j=0, i;

	system("cls");
	printf ("Enter the InFix Expression : ");
	gets(s);

	for (i=0; s[i]!='\0'; i++)
	{
		if ( IsOperand(s[i]) )
			ops[j++] = s[i];
		else
		{
			while (!IsEmpty() && Prcd(opstk[Top],s[i]) )
				ops[j++] = Pop(opstk);

			if (IsEmpty() || s[i]!=')')
				Push(s[i]);
			else
				Pop();
		}
	}

	while (!IsEmpty())
		ops[j++] = Pop(opstk);

	ops[j] = '\0';
	printf ("The Post Fix is : %s", ops);
	return 0;
}
```

/* Postfix Evaluation */

```
#define Bool
#define True 1
#define False 0

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

/*-------------------------------- Stack ------------------------------------------*/

#define MaxSize 10

int Top=-1;
float opstk[MaxSize];

void Push (float op)
{
        opstk[++Top] = op;
}

float Pop()
{
        return(opstk[Top--]);
}

Bool IsEmpty ()
{
        return (Top==-1);
}

/*---------------------------------- End Of Stack ----------------------------------------*/

Bool IsOperand(char ch)
{
        return (ch>='0' && ch<='9');
}

int main()
{
        char s[40];
        int i;
        float op1, op2, res;

        system("cls");
        printf ("Enter the PostFix Expression : ");
```

```
	scanf ("%s", s);

	for (i=0; s[i]!='\0'; i++)
	{
		if (IsOperand(s[i]))
			Push(s[i]-48.0);
		else
		{
			op2 = Pop();
			op1 = Pop();
			switch (s[i])
			{
				case '+' : res = op1 + op2; break;
				case '-' : res = op1 - op2; break;
				case '*' : res = op1 * op2; break;
				case '/' : res = op1 / op2; break;
				case '^' : res = pow(op1,op2); break;
				default : printf ("Invalid Operator Encountered.\n");
				getch();
			}
			Push(res);
		}
	}

	printf ("The Final Result is %8.3f\n", Pop());
	return 0;
}
```

```
                              /*     Binary Tree  */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>

typedef struct node Node;

struct node
{
        Node *left;
        char data;
        Node *right;
};

Node* Create()
{
        Node *p;
        char opt;

        p = (Node *) malloc(sizeof(Node));

        printf ("\nEnter the element : ");
        p->data = getche();

        printf ("\nIs there left child for %c [y/n] : ", p->data);
        opt = getche();

        if (opt=='y')
                p->left = Create();
        else
                p->left = NULL;

        printf ("\nIs there right child for %c [y/n] : ", p->data);
        opt = getche();

        if (opt=='y')
                p->right = Create();
        else
                p->right = NULL;

        return p;
}

void InOrder(Node *p)
{
        if (p!=NULL)
        {
                InOrder(p->left);
                printf ("%c ", p->data);
                InOrder(p->right);
        }
}

void PreOrder(Node *p)
{
        if (p!=NULL)
        {
                printf ("%c ", p->data);
                PreOrder(p->left);
                PreOrder(p->right);
        }
}

void PostOrder(Node *p)
{
        if (p!=NULL)
        {
```

```
		PostOrder(p->left);
		PostOrder(p->right);
		printf ("%c ", p->data);
	}
}

int main()
{
	Node *Root=NULL;
	int ch;

	do
	{
		system("cls");
		printf ("Binary Tree Operation Menu	\n");
		printf ("-------------------------	\n");
		printf ("1.Creation\n");
		printf ("2.InOrder Traversal\n");
		printf ("3.PreOrder Traversal\n");
		printf ("4.PostOrder Traversal\n");
		printf ("5.Exit\n\n");

		printf ("Enter Your Choice [1..5] : ");
		scanf ("%d", &ch);

		system("cls");
		switch(ch)
		{
			case 1 :	Root = Create();
					break;

			case 2 :	if (Root==NULL)
						printf ("Binary Tree is Empty.\n");
					else
					{
						printf ("Inorder Traversal is ...\n");
						InOrder(Root);
					}
					break;

			case 3 :	if (Root==NULL)
						printf ("Binary Tree is Empty.\n");
					else
					{
						printf ("Preorder Traversal is ...\n");
						PreOrder(Root);
					}
					break;

			case 4 :	if (Root==NULL)
						printf ("Binary Tree is Empty.\n");
					else
					{
						printf ("Postorder Traversal is ...\n");
						PostOrder(Root);
					}
					break;

			case 5 :	break;

		       default :	printf ("Invalid Choice.");
		}
		getch();
	}
	while (ch!=5);

	return 0;
}
```

```
                    //      Binary Search Tree Operations

#define Bool int
#define True 1
#define False 0

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

typedef struct node Node;

struct node
{
        Node *left;
        int data;
        Node *right;
};

void Insert(Node *rt, int ele)
{
        Node *p=rt, *pp=NULL;

        while (p!=NULL)
        {
                pp = p;
                if (ele < p->data)
                        p = p->left;
                else
                        if (ele > p->data)
                                p = p->right;
                        else
                                break;
        }

        if (p!=NULL)
                printf ("Duplicated Key Encountered.");
        else
        {
                p = (Node *) malloc(sizeof(Node));
                p->left = NULL;
                p->data = ele;
                p->right = NULL;

                if (ele < pp->data)
                        pp->left = p;
                else
                        pp->right = p;
        }
}

void Inorder(Node *p)
{
        if (p!=NULL)
        {
                Inorder(p->left);
                printf ("%d ", p->data);
                Inorder(p->right);
        }
}

Bool Search(Node *rt, int ele)
{
        Node *p=rt;
        Bool found=False;

        while (p!=NULL && !found)
```

```
	{
		if (p->data < ele)
			p = p->right;
		else
			if (p->data > ele)
				p = p->left;
			else
				found=True;
	}

	return found;
}

void Delete(Node **rt, int x)
{
	Node *p=*rt, *pp=NULL, *s, *ps, *c;

	while (p!=NULL && p->data!=x)
	{
		pp = p;

		if (x < p->data)
			p = p->left;
		else
			p = p->right;
	}

	if (p==NULL)
	{
		printf ("Node not found");
		return;
	}

	if (p->left!=NULL && p->right!=NULL)
	{
		ps = p;
		s = p->left;
		while (s->right)
		{
			ps = s;
			s = s->right;
		}
		p->data = s->data;
		pp = ps;
		p = s;
	}

	if (p->left!=NULL)
		c = p->left;
	else
		c = p->right;

	if (p==*rt)
		*rt = c;
	else
	{
		if (p==pp->left)
			pp->left = c;
		else
			pp->right = c;
	}

	free(p);
}

int main()
{
	Node *Root=NULL;
```

```
	int ele;
	int ch;

	do
	{
		system("cls");
		printf ("BSTree Operations Menu\n");
		printf ("----------------------\n\n");
		printf ("1.Insert\n2.Tree Sort\n3.Search\n4.Delete\n5.Exit\n\n");

		printf ("Your Choice [1..5] : ");
		scanf ("%d", &ch);

		switch(ch)
		{
			case 1 :printf ("Enter the Element : ");
				scanf ("%d", &ele);

				if (Root==NULL)
				{
					Root = (Node *) malloc (sizeof(int));
					Root->left = NULL;
					Root->data = ele;
					Root->right = NULL;
				}
				else
					Insert(Root, ele);

				break;

			case 2 : printf ("\nThe Sorted List is...\n");
				Inorder(Root);
				break;

			case 3 : printf ("Enter the Element : ");
				scanf ("%d", &ele);

				if (Search(Root, ele))
					printf ("Element found.\n");
				else
					printf ("Element not found.\n");
				break;

			 case 4 : printf ("Enter the Element : ");
				  scanf ("%d", &ele);

				  Delete(&Root,ele);
				  break;

			 case 5 : break;

			default : printf ("Invalid Choice.");
					getch();
		}
		getch();
	}
	while (ch!=5);

	return 0;
}
```

```
            /*    Hashing using SLL */

#define Bool int
#define True 1
#define False 0

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

typedef struct node Node;

struct node
{
      int data;
      Node *next;
};

int Size;
Node **ht;

void Destroy(Node *);
void Insert();
void Display();
void Search();
void Delete();
void Ins(Node**, int);
void Disp(Node*);
void Srch(Node*, int);
void Del(Node**, int);


Node* getNode(int ele, Node *ptr)
{
      Node *p;

      p = (Node*) malloc(sizeof(Node));
      p->data = ele;
      p->next = ptr;
}

void create(int ms)
{
      int i;

      Size = ms;
      ht = (Node**) malloc(Size * sizeof(Node*));
```

```
      for (i=0; i<Size; i++)
            ht[i]=NULL;
}

void Destroy(Node *First)
{
      Node *p;

      while(First!=NULL)
      {
            p = First;
            First = First->next;
            free(p);
      }
}

void Dest()
{
      int i;

      for (i=0; i<Size; i++)
            Destroy(ht[i]);

      free(ht);
}

void Insert()
{
      int ak, rk;

      printf ("Enter the Element to Insert : ");
      scanf ("%d", &ak);

      rk = ak % Size;
      Ins(&ht[rk],ak);
}

void Delete()
{
      int ak,rk;

      printf ("Enter the Element to Delete : ");
      scanf ("%d", &ak);

      rk = ak % Size;
      Del(&ht[rk],ak);
}
```

```
void Search()
{
	int ak,rk;

	printf ("Enter the Element to Search : ");
	scanf ("%d", &ak);

	rk = ak % Size;
	Srch(ht[rk],ak);
}

void Display()
{
	int i;

	for (i=0; i<Size; i++)
	{
		printf ("\nHT[%d] = ", i);
		Disp(ht[i]);
	}
}

void Ins(Node **ptr, int key)
{
	Node *First = *ptr;
	Node *p=NULL,*prev=NULL;

	if (First==0)
		*ptr = getNode(key,NULL);

	for (p=First; p!=NULL && p->data<key; p=p->next)
		prev = p;

	if (p==NULL || p->data>key)
	{
		if (p==First)
			*ptr = getNode(key,First);
		else
			prev->next = getNode(key,p);
	}
	else
		printf ("Duplicate Key Found.\n");
}

void Disp(Node *First)
{
	Node *p;
```

```
	for (p=First; p; p=p->next)
		printf ("--->%d", p->data);

	printf ("\n");
}

void Srch(Node *First, int key)
{
	Node *p;

	for (p=First; p!=NULL && p->data<key; p=p->next)
	{		}

	if (p==NULL || p->data>key)
		printf ("Key Not Found.\n");
	else
		printf ("Key Found.\n");
}

void Del(Node **ptr, int key)
{
	Node *First, *p,*prev=NULL;

	First = *ptr;

	for (p=First; p!=NULL && p->data<key; p=p->next)
		prev = p;

	if (p==NULL || p->data>key)
		printf ("Key Not Found.\n");
	else
	{
		if (p==First)
			*ptr = First->next;
		else
			prev->next = p->next;

		free(p);
	}
}

int main()
{
	int ch;

	system("cls");
	create(10);
```

```
	do
	{
		system("cls");
		printf ("HashTable Operations Menu\n");
		printf ("-------------------------\n");
		printf ("1.Insert\n");
		printf ("2.Display\n");
		printf ("3.Search\n");
		printf ("4.Delete\n");
		printf ("5.Exit\n\n");

		printf ("Enter your Choice [1..5] ? : ");
		scanf ("%d", &ch);

		switch (ch)
		{
			case 1 :	Insert();
					break;

			case 2 :	Display();
					break;

			case 3 :	Search();
					break;

			case 4 :	Delete();
					break;

			case 5 :	break;

			default :	printf  ("Invalid Choice");
		}
		getch();
	}
	while (ch!=5);
}
```