**Static array**

#include<conio.h>

#include<iostream>

using namespace std;

void display(int a[], int size);

int main()

{

int a[5];

for(int i=0;i<5;i++)

{

cout<<"Enter data value " <<i+1 <<": ";

cin>>a[i];

}

display(a,5);

getch();

return 0;

}

void display(int a[], int size)

{

cout<<"Values in array are: ";

for(int i=0;i<size; i++)

{

cout<<a[i] <<" ";

}

}

**Dynamic array**

void display(int a[], int size);

int main()

{

int \*a, n;

cout<<"Enter the size of array: ";

cin>>n;

a = new int[n];

for(int i=0;i<n;i++)

{

cout<<"Enter data value " <<i+1 <<": ";

cin>>a[i];

}

display(a,n);

getch();

return 0;

}

void display(int a[], int size)

{

cout<<"Values in array are: ";

for(int i=0;i<size; i++)

{

cout<<a[i] <<" ";

}

}

**Insertion in array**

void display(int a[], int n);

int main()

{

int \*a, pos, val, size, n;

cout<<"Enter size of array: ";

cin>>size;

a = new int[size];

cout<<"Enter no. of elements to enter: ";

cin>>n;

for(int i=0;i<n;i++)

{

cout<<"Enter data value " <<i+1 <<": ";

cin>>a[i];

}

display(a,n);

cout<<"\n Enter data-value to insert: ";

cin>>val;

cout<<"\n Enter index position where to insert: ";

cin>>pos;

for(int j=n; j>=pos; j--)

{

a[j+1] = a[j];

}

a[pos] = val;

n++;

display(a,n);

delete []a;

getch();

return 0;

}

void display(int a[], int n)

{

cout<<"\n Values in array are: ";

for(int i=0;i<n; i++)

{

cout<<a[i] <<" ";

}

}

**Deletion in array**

void display(int a[], int size);

int main()

{

int size, pos, \*a;

cout<<"Enter the size of array: ";

cin>>size;

a = new int[size];

for(int i=0;i<size;i++)

{

cout<<"Enter data value " <<i+1 <<": ";

cin>>a[i];

}

display(a,size);

cout<<"\n Enter index position you want to delete: ";

cin>>pos;

for(int i=pos; i<(size-1); i++)

{

a[i] = a[i+1];

}

size--;

display(a,size);

delete []a;

getch();

return 0;

}

void display(int a[], int s)

{

cout<<"\n Values in array are: ";

for(int i=0;i<s; i++)

{

cout<<a[i] <<" ";

}

}

**Linear Search**

void main()

{

clrscr();

int a[100],i,n,item,s=0;

cout<<"\n------------ LINEAR SEARCH ------------ \n\n";

cout<<"Enter No. of Elements=";

cin>>n;

cout<<"\nEnter Elements=\n";

for(i=1;i<=n;i++)

{

cin>>a[i];

}

cout<<"\nEnter Element you want to Search=";

cin>>item;

for(i=1;i<=n;i++) //Array Elements Comparsion with Item

{

if(a[i]==item)

{

cout<<"\nData is Found at Location : "<<i;

s=1;

break;

}

}

if(s==0)

{

cout<<"Data is Not Found";

}

getch();

}

**Bubble Sort**

int main()

{

int n, \*a;

cout<<"Enter size of array:";

cin>>n;

a = new int[n];

for(int i=0; i<n; i++)

{

cout<<"Enter value: ";

cin>>a[i];

}

cout<<"Elements entered are:" <<endl;

for(int i=0; i<n; i++)

{

cout<<a[i] <<" ";

}

int temp;

for(int i=0; i<(n-1); i++)

{

for(int j=0; j<(n-1); j++)

{

if(a[j]>a[j+1])

{

temp = a[j];

a[j]= a[j+1];

a[j+1] = temp;

}

}

}

cout<<"\n Sorted entered are:" <<endl;

for(int i=0; i<n; i++)

{

cout<<a[i] <<" ";

}

getch();

return 0;

}

**2-D Array**

int main()

{

int row, col;

cout<<"Enter no. of rows: ";

cin>>row;

int \*a[row]; //array of pointers

cout<<"Enter no. of columns: ";

cin>>col;

for(int r=0; r<row; r++)

{

a[r] = new int[col]; //creating array for each row

}

for(int r=0; r<row; r++)

{

for(int c=0; c<col; c++)

{

cout<<"Enter element in row:"<<r <<" col:"<<c <<": ";

cin>>a[r][c];

}

}

cout<<"\n Final 2-D array is: " <<endl;

for(int r=0; r<row; r++)

{

for(int c=0; c<col; c++)

{

cout<<a[r][c] <<" ";

}

cout<<endl;

}

for(int r=0; r<row; r++)

{

delete []a[r]; //deleting array of each row

}

delete \*a;

getch();

return 0;

}

**Merging Sorted-array**

void display(int arr[], int n);

int main()

{

int \*a, \*b, \*c, aSize, bSize, cSize;

int ja, kb, lc;

ja = kb = lc = 0;

cout<<"Enter size of array a[]: ";

cin>>aSize;

a = new int[aSize];

cout<<"Enter size of array b[]: ";

cin>>bSize;

b = new int[bSize];

cout<<"Enter elements for array a[]: " <<endl;

for(int i=0; i<aSize; i++)

{

cout<<"Enter data value " <<i+1 <<": ";

cin>>a[i];

}

cout<<"Enter elements for array b[]: " <<endl;

for(int i=0; i<bSize; i++)

{

cout<<"Enter data value " <<i+1 <<": ";

cin>>b[i];

}

display(a, aSize);

cout<<endl;

display(b, bSize);

//Merging

cSize = aSize + bSize;

c = new int[cSize];

while( (ja < aSize) && (kb < bSize) )

{

if(a[ja] < b[kb])

{

c[lc] = a[ja];

ja++; lc++;

}

else

{

c[lc] = b[kb];

kb++; lc++;

}

}

if(ja < aSize)

{

while(ja < aSize)

{

c[lc] = a[ja];

ja++; lc++;

}

}

else

{

while(kb < bSize)

{

c[lc] = b[kb];

kb++; lc++;

}

}

display(c,cSize);

getch();

return 0;

}

void display(int arr[], int n)

{

cout<<"\n Values in array are: ";

for(int i=0;i<n; i++)

{

cout<<arr[i] <<" ";

}

}

**1-way Linked-list**

struct node

{

int info;

node \*next;

}\*start=NULL;

void creation();

void insertion(node \*);

int deletion(node \*);

void display(node \*);

int main()

{

int choice;

if(start==NULL)

{

creation();

}

while(1)

{

cout<<"\n\*\*\*\* Enter choice: \*\*\*\*" <<endl;

cout<<"1 : Insertion" <<endl;

cout<<"2 : Deletion" <<endl;

cout<<"3 : Display" <<endl;

cout<<"4 : Exit" <<endl;

cin>>choice;

switch(choice)

{

case 1:

insertion(start);

break;

case 2:

deletion(start);

break;

case 3:

display(start);

break;

case 4:

exit(1);

break;

default:

cout<<"Wrong choice..." <<endl;

}

}

getch();

return 0;

}

void creation()

{

start = new node;

cout<<"Enter data-value for 1st node: ";

cin>>start->info;

start->next = NULL;

}

void insertion(node \*p) //Here "p" is traversing pointer

{

while(p->next!=NULL)

{

p = p->next;

}

node \*temp = new node;

cout<<"Enter data-value: ";

cin>> temp->info;

temp->next = NULL;

p->next = temp; //joining last node with new node

}

int deletion(node \*ptr) //Here "ptr" is traversing pointer

{

node \*ptrP = NULL; //pointer previous to ptr

if(ptr==NULL)

{

cout<<"List is empty.." <<endl;

return 0;

}

int val;

cout<<"Enter data-value to delete: ";

cin>> val;

while( (ptr!=NULL)&&(ptr->info!=val) )

{

ptrP = ptr;

ptr = ptr->next;

}

if(ptr==NULL) //element not found

{

cout<<"Value doesn't exist in list" <<endl;

return 0;

}

else if(ptrP==NULL) //element found at 1st location

{

start = start->next;

delete ptr;

}

else //element found at other locations

{

ptrP->next = ptr->next; //joining previous node with next node

delete ptr;

}

}

void display(node \*p) //Here "p" is traversing pointer

{

cout<<"\n Elements in linked-list are: " <<endl;

while(p!=NULL)

{

cout<<p->info <<" --> ";

p = p->next;

}

cout<<"NULL" <<endl;

}

**2-way Linked-list**

struct node

{

node \*prev;

int info;

node \*next;

}\*start=NULL;

void creation();

void insertion(node \*);

int deletion(node \*);

void display(node \*);

int main()

{

int choice;

if(start==NULL)

{

creation();

}

while(1)

{

cout<<"\n\*\*\*\* Enter choice: \*\*\*\*" <<endl;

cout<<"1 : Insertion" <<endl;

cout<<"2 : Deletion" <<endl;

cout<<"3 : Display" <<endl;

cout<<"4 : Exit" <<endl;

cin>>choice;

switch(choice)

{

case 1:

insertion(start);

break;

case 2:

deletion(start);

break;

case 3:

display(start);

break;

case 4:

exit(1);

break;

default:

cout<<"Wrong choice..." <<endl;

}

}

getch();

return 0;

}

void creation()

{

start = new node;

cout<<"Enter data-value for 1st node: ";

cin>>start->info;

start->prev = NULL;

start->next = NULL;

}

void insertion(node \*p) //Here "p" is traversing pointer

{

while(p->next!=NULL)

{

p = p->next;

}

node \*temp = new node;

cout<<"Enter data-value: ";

cin>> temp->info;

temp->next = NULL;

p->next = temp; //joining last node with new node

temp->prev = p; //joining new node with last node

}

int deletion(node \*ptr) //Here "ptr" is traversing pointer

{

if(ptr==NULL)

{

cout<<"List is empty.." <<endl;

return 0;

}

int val;

cout<<"Enter data-value to delete: ";

cin>> val;

while( (ptr!=NULL)&&(ptr->info!=val) )

{

ptr = ptr->next;

}

if(ptr==NULL) //element not found

{

cout<<"Value doesn't exist in list" <<endl;

return 0;

}

else if(ptr->prev==NULL) //element found at 1st location

{

start = start->next;

delete ptr;

}

else //element found at other locations

{

(ptr->prev)->next = ptr->next; //joining previous node with next node

delete ptr;

}

}

void display(node \*p) //Here "p" is traversing pointer

{

cout<<"\n Elements in linked-list are: " <<endl;

while(p!=NULL)

{

cout<<p->info <<" <==> ";

p = p->next;

}

cout<<"NULL" <<endl;

}

**Circular Linked-list (1-way)**

struct node

{

int info;

node \*next;

}\*start=NULL;

void creation();

void insertAtLast(node \*);

int deletion(node \*);

void display(node \*);

int main()

{

int choice;

if(start==NULL)

{

creation();

}

while(1)

{

cout<<"\n\*\*\*\* Enter choice: \*\*\*\*" <<endl;

cout<<"1 : Insertion At Last" <<endl;

cout<<"2 : Deletion" <<endl;

cout<<"3 : Display" <<endl;

cout<<"4 : Exit" <<endl;

cin>>choice;

switch(choice)

{

case 1:

insertAtLast(start);

break;

case 2:

deletion(start);

break;

case 3:

display(start);

break;

case 4:

exit(1);

break;

default:

cout<<"Wrong choice..." <<endl;

}

}

getch();

return 0;

}

void creation()

{

start = new node;

cout<<"Enter data-value for 1st node: ";

cin>>start->info;

start->next = start;

}

void insertAtLast(node \*p) //Here "p" is traversing pointer

{

while(p->next!=start)

{

p = p->next;

}

node \*temp = new node;

cout<<"Enter data-value: ";

cin>> temp->info;

p->next = temp; //joining last node with new node

temp->next = start; //joining new node with 1st node

}

int deletion(node \*ptr) //Here "ptr" is traversing pointer

{

node \*loc = NULL; //location of element to delete

if(ptr==NULL)

{

cout<<"List is empty.." <<endl;

return 0;

}

node \*ptrP = start; //pointer previous to ptr

ptr = ptr->next; //initialize "ptr" from 2nd node

int val;

cout<<"Enter data-value to delete: ";

cin>> val;

while(ptr!=start)

{

if(ptr->info==val)

{

loc = ptr; //location found

break;

}

else

{

ptrP = ptr;

ptr = ptr->next;

}

}

if(ptr->info==val) //testing element at 1st location which is not tested in loop

{

loc = ptr; //location found

}

if(loc==NULL) //element not found

{

cout<<"Value doesn't exist in list" <<endl;

return 0;

}

else if(ptrP==ptr) //element found at 1st location

{

start = NULL;

delete ptr;

}

else //element found at other locations

{

ptrP->next = ptr->next; //joining previous node with next node

start = ptr->next; //resetting the start pointer

delete ptr;

}

}

void display(node \*p) //Here "p" is traversing pointer

{

cout<<"\n Elements in linked-list are: " <<endl;

if(p==NULL) //testing is list is empty

{

cout<<"NULL";

}

else

{

do

{

cout<<p->info <<" --> ";

p = p->next;

}while(p!=start);

cout<<p->info; //printing 1st element again to depict circular list

}

}

**Stack (using liked-list)**

struct node

{

int info;

node \*next;

}\*top=NULL;

void push(int);

void pop();

void peek();

int main()

{

int val, choice;

while(1)

{

cout<<"\n\*\*\*\* Enter choice: \*\*\*\*" <<endl;

cout<<"1 : Push" <<endl;

cout<<"2 : Pop" <<endl;

cout<<"3 : Peek" <<endl;

cout<<"4 : Exit" <<endl;

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter value to push: ";

cin>>val;

push(val);

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

exit(1);

break;

default:

cout<<"Wrong choice..." <<endl;

}

}

getch();

return 0;

}

void push(int val)

{

node \*temp = new node; //creating new node

temp->info = val;

if(top==NULL) //checking if stack is empty

{

temp->next = NULL; //assigning NULL to link of last node

top = temp;

}

else

{

temp->next = top; //adding new node in stack

top = temp;

}

}

void pop()

{

node \*p;

if(top==NULL) //checking if stack is empty

{

cout<<"Stack is empty...";

}

else

{

p = top;

top = top->next; //moving to next node

cout<<"Element deleted is: " <<p->info <<endl;

delete p;

}

}

void peek()

{

node \*p = top; //taking temporary pointer for traversing

cout<<"\n Elements in stack are: " <<endl;

while(p!=NULL)

{

cout<<p->info <<endl;

p = p->next;

}

cout<<"NULL" <<endl;

}

**Stack (using array)**

void push(int \*, int);

void pop(int \*);

void peek(int \*);

int top = -1; //initializing top variable

int size = 3; //setting size or capacity of stack

int main()

{

int val, choice, arr[size];

while(1)

{

cout<<"\n\*\*\*\* Enter choice: \*\*\*\*" <<endl;

cout<<"1 : Push" <<endl;

cout<<"2 : Pop" <<endl;

cout<<"3 : Peek" <<endl;

cout<<"4 : Exit" <<endl;

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter value to push: ";

cin>>val;

push(arr, val);

break;

case 2:

pop(arr);

break;

case 3:

peek(arr);

break;

case 4:

exit(1);

break;

default:

cout<<"Wrong choice..." <<endl;

}

}

getch();

return 0;

}

void push(int arr[], int val)

{

if(top==size-1) //checking if stack is full and comparing with (size-1) as array starts from 0 index;

{

cout<<"Overflow..." <<endl;

}

else

{

top++; //incrementing top variable

arr[top] = val;

}

}

void pop(int arr[])

{

if(top==-1) //checking if stack is empty

{

cout<<"Underflow..." <<endl;

}

else

{

cout<<"Value poped is: " <<arr[top] <<endl;

top--; //decrementing top variable

}

}

void peek(int arr[])

{

int p = top;

cout<<"\n Elements in stack are: " <<endl;

while(p!=-1)

{

cout<<arr[p] <<endl;

p--;

}

cout<<"NULL" <<endl;

}

**Queue (using array)**

int insertion(int \*, int);

int deletion(int \*);

int peek(int \*);

int front=-1, rear=-1; //initializing variables

int size = 5; //setting size or capacity of circular queue

int main()

{

int val, choice, arr[size];

while(1)

{

cout<<"\n\*\*\*\* Enter choice: \*\*\*\*" <<endl;

cout<<"1 : Insert" <<endl;

cout<<"2 : Delete" <<endl;

cout<<"3 : Peek" <<endl;

cout<<"4 : Exit" <<endl;

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter value to insert: ";

cin>>val;

insertion(arr, val);

break;

case 2:

deletion(arr);

break;

case 3:

peek(arr);

break;

case 4:

exit(1);

break;

default:

cout<<"Wrong choice..." <<endl;

}

}

getch();

return 0;

}

int insertion(int arr[], int val)

{

if( (front==0 && rear==size-1) || (front==rear+1) ) //comparison with (size-1) since index starts from 0

{

cout<<"Overflow..." <<endl;

return 0;

}

else if(front==-1) //checking if queue is empty

front = rear = 0; //bringing front and rear to beginning position

else if(rear==size-1) //checking if rear is at end position of array

rear = 0; //bringing rear to beginning position

else

rear++; //incrementing front variable

arr[rear] = val; //inserting in queue

}

int deletion(int arr[])

{

if(front==-1) //checking if queue is empty

{

cout<<"Underflow..." <<endl;

return 0;

}

int val = arr[front]; //temporarily storing value to be deleted

if(front==rear) //checking if only 1 element in queue

front = rear = -1;

else if(front==size-1) //comparison with (size-1) since index starts from 0

front = 0;

else

front++;

cout<<"Value deleted is: " <<val <<endl;

}

int peek(int arr[])

{

int p = front;

if(p==-1)

{

cout<<"Queue is empty..." <<endl;

return 0;

}

cout<<"\n Elements in queue are: " <<endl;

while(p!=rear)

{

cout<<arr[p] << " ";

if(p==size-1) //checking if end of array is reached while traversing

p=0; //bringing varible to beginning position

else

p++;

}

cout<<arr[p]; //printing last element of queue which was skipped in loop

}

**Queue (using linked-list)**

struct node

{

int info;

node \*next;

}\*front=NULL, \*rear=NULL;

void insertion(int);

void deletion();

int peek();

int main()

{

int val, choice;

while(1)

{

cout<<"\n\*\*\*\* Enter choice: \*\*\*\*" <<endl;

cout<<"1 : Insert" <<endl;

cout<<"2 : Delete" <<endl;

cout<<"3 : Peek" <<endl;

cout<<"4 : Exit" <<endl;

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter value to insert: ";

cin>>val;

insertion(val);

break;

case 2:

deletion();

break;

case 3:

peek();

break;

case 4:

exit(1);

break;

default:

cout<<"Wrong choice..." <<endl;

}

}

getch();

return 0;

}

void insertion(int val)

{

node \*temp = new node;

temp->info = val;

temp->next = NULL;

if(rear==NULL) //checking if queue is empty

front = rear = temp;

else

{

rear->next = temp; //connecting last node with new node

rear = temp; //moving rear pointer to last node

}

}

void deletion()

{

node \*p;

if(front==NULL) //checking if queue is empty

cout<<"UnderFlow..." <<endl;

else if(front==rear) //checking if only 1 element remaining in queue

{

p = front;

front = rear = NULL;

cout<<"Element deleted is: " <<p->info <<endl;

delete p;

}

else

{

p = front;

front = front->next;

cout<<"Element deleted is: " <<p->info <<endl;

delete p;

}

}

int peek()

{

node \*p = front;

if(p==NULL)

{

cout<<"Queue is empty..." <<endl;

return 0;

}

cout<<"\n Element in queue are: " <<endl;

while(p!=NULL)

{

cout<<p->info <<" ";

p = p->next;

}

cout<<endl;

}