////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//TEST RUN Q1//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(){

string Example="###### #. .####@$#$ ##$. .$ ## $.#########";

char\* Ex=&Example[0];

Sokoban B=Sokoban(Ex);

cout<<"1"<<endl;

B.print\_puzzle();

Sokoban A=Sokoban(1);

cout<<"2"<<endl;A.print\_puzzle();

A.move\_up();

cout<<"3"<<endl;A.print\_puzzle();

A.move\_right();

cout<<"EE441 "<<endl;A.print\_puzzle();

A.move\_down();

cout<<"HCN"<<endl;A.print\_puzzle();

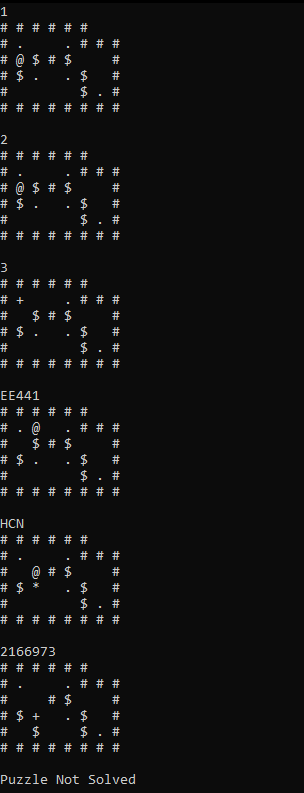
A.move\_down();

cout<<"2166973"<<endl;A.print\_puzzle();

if(A.is\_solved())

cout<<"Puzzle Solved"<<endl;

else



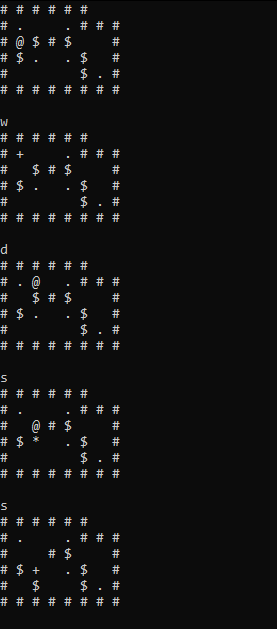
cout<<"Puzzle Not Solved"<<endl;

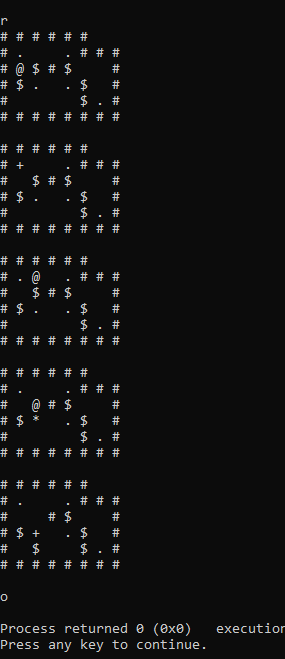
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//TEST RUN Q3orQ4//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

w, d, s, s, r, o key presses were performed.





**EE441 Programming Assignment I: *Classes, Stacks, Queues and Linked Lists***

**Q1)** **Implement a class Sokoban that represents a Sokoban puzzle.**

const int nRow = 6;

const int nColumn = 8;

const int nElement= nRow\*nColumn;

const int nSize=100;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban Declaration

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

class Sokoban{

private:

char mMap[nElement];//array that holds the game map.

int findPlayer() const;//Method that returns the player location such that row 3 column 3 will return 27

int moveCheck(int pLoc, int shift, int i=0);//method that determines if the move is valid.

void Edit(int pLoc, int shift, int i);//edits the game map according to provided parameters.

public:

//Constructors

Sokoban(const char Map[nElement]);//initializes the class from a char array

Sokoban();//default constructor

Sokoban(int i);// initializes the class from a text file.

Sokoban(const Sokoban &obj);//a copy constructor

Sokoban& operator=(const Sokoban &rhs);//copy assignment operator

//move methods

bool move\_up();

bool move\_down();

bool move\_left();

bool move\_right();

bool is\_solved();//checks puzzle if it is solved.

void print\_puzzle() const;//prints current puzzle map

};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban Method Definitions

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban default constructor

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Sokoban::Sokoban(){}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban constructor (initializes the class from a char array)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Sokoban::Sokoban(const char Map[nElement]){

//copies each array element one by one

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

\*(mMap+i)=\*(Map+i);}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban constructor (initializes the class from a text file.)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Sokoban::Sokoban(int i){//int i is used to have parameter difference between constructors

string filename = "sample\_puzzle.txt";

char\* fileloc = &filename[0];//char \* is solved the problem I had with the file.open(fileloc) line

//used the provided code with minor changes.

char data[6][8];

char dummy;

ifstream file;

file.open(fileloc);

for(int i=0; i<6; ++i){

for(int j=0; j<8; ++j){

file >> noskipws >> data[i][j];

mMap[i\*8+j]=data[i][j];//assigns char values to dynamic memory.

}

file >> noskipws >> dummy;

}file.close(); }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban copy constructor

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Sokoban::Sokoban(const Sokoban &obj){

//copies each array element one by one

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

\*(mMap+i) = \*(obj.mMap+i); }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban = operator definition

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Sokoban& Sokoban::operator=(const Sokoban &rhs){

//copies each array element one by one

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

\*(mMap+i)=\*(rhs.mMap+i);

return \*this;//this object is returned

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban puzzle print function

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void Sokoban::print\_puzzle() const{//prints all 48 elements of the mMap[48]

for(int i=0; i<6; ++i){

for(int j=0; j<8; ++j){

cout<<mMap[i\*8+j]<<" ";}

cout<<endl;}

cout<<endl;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban is\_solved function

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

bool Sokoban::is\_solved(){

for(int i=1; i<nRow-1; ++i){

for(int j=1; j<nColumn-1; ++j){

if(mMap[i\*8+j]=='.' || mMap[i\*8+j]=='+')

return false;}}

return true;//returns true if there are no target location or the player at a target location is found

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban findPlayer function

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int Sokoban::findPlayer() const{

// returns the location of "the player" or "the player at a target location"

for(int i=1; i<nRow-1; ++i){

for(int j=1; j<nColumn-1; ++j){

if(mMap[i\*8+j]=='@' || mMap[i\*8+j]=='+')

return i\*8+j;

}

}

cerr<<"Error: Player not found."<<endl;

return -1;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban moveCheck function

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int Sokoban::moveCheck(int pLoc, int shift, int i){ //int i=0 by default.

//Shift= 1 for move right, -8 for //move up. pLoc is player location

if(mMap[pLoc+shift]=='#') //if there is a wall ahead

return 0; //move is invalid

else if(mMap[pLoc+shift]=='$' || mMap[pLoc+shift]=='\*') //if there is a box ahead

if(i==0) //if moveCheck called for the first time

return moveCheck(pLoc+shift, shift, 1); //check what is behind the box

else //another box is found ahead at 2nd call of the function

return 0; //move is invalid

else //move is valid returns 2 if function is called once, 3 if called twice

return 2+i; //valid move.

}//returning integer lets us know the how many elements will be edited in the mMap[]

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban Edit function(edits game map)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void Sokoban::Edit(int pLoc, int shift, int i){

//pLoc player location shift== 1 for move right, -8 for move up. i = moveCheck()(2 or 3)

//called only if the move is valid.

for(int k=0; k<i; k++){

// pLoc+k\*shift -> k=0 player location k=1 location front of the player...

// here comments start with if player means if there is a player at the pLoc+k\*shift

//if it is true then under the statement I commented what will that location will be converted to

if(mMap[pLoc+k\*shift]=='@')//if player //can be true only for k=0

mMap[pLoc+k\*shift]=' ';// empty cell

else if(mMap[pLoc+k\*shift]=='+')//if player on target//can be true only for k=0

mMap[pLoc+k\*shift]='.';// target location.

else if(mMap[pLoc+k\*shift]=='$') // if box //can be true only for k>0

mMap[pLoc+k\*shift]='@';//player

else if(mMap[pLoc+k\*shift]=='\*')//if box on target //can be true only for k>0

mMap[pLoc+k\*shift]='+';//player on target

else if(mMap[pLoc+k\*shift]==' ')//if empty cell //can be true only for k>0

if(k==1)//if the first location further the Player Location

mMap[pLoc+k\*shift]='@';//player

else// if the second location further the Player Location

mMap[pLoc+k\*shift]='$';//box

else if(mMap[pLoc+k\*shift]=='.')//if target //can be true only for k>0

if(k==1)//if the first location further the Player Location

mMap[pLoc+k\*shift]='+';//player on target

else// if the second location further the Player Location

mMap[pLoc+k\*shift]='\*';//box on target

else;//line is added to eliminate ambiguity of else statement.

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban move\_up function//move functions have same structure.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

bool Sokoban::move\_up(){

int pLoc = findPlayer();//returns player location

int moveCase = moveCheck(pLoc, -1\*nColumn );//returns 0, 2 or 3

if(moveCase==0)//invalid move

return false;

else{//valid move

Edit(pLoc, -1\*nColumn, moveCase);//edits the map according to shift amount and case2or3

return true;

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban move\_down function//only 2nd parameter of edit and movecheck function differs

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

bool Sokoban::move\_down(){

int pLoc = findPlayer();//returns player location

int moveCase = moveCheck(pLoc, nColumn );//returns 0, 2 or 3

if(moveCase==0)//invalid move

return false;

else{//valid move

Edit(pLoc, nColumn, moveCase);//edits the map according to shift amount and case2or3

return true;

}}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban move\_left function//only 2nd parameter of edit and movecheck function differs

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

bool Sokoban::move\_left(){

int pLoc = findPlayer();//returns player location

int moveCase = moveCheck(pLoc, -1 );//returns 0, 2 or 3

if(moveCase==0)//invalid move

return false;

else{//valid move

Edit(pLoc, -1, moveCase);//edits the map according to shift amount and case2or3

return true;

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Sokoban move\_right function//only 2nd parameter of edit and moveCheck function differs

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

bool Sokoban::move\_right(){

int pLoc = findPlayer();//returns player location

int moveCase = moveCheck(pLoc, 1 );

if(moveCase==0)//invalid move

return false;

else{//valid move

Edit(pLoc, 1, moveCase);//edits the map according to shift amount and case2or3

return true;

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//TEST RUN//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(){

string Example="###### #. .####@$#$ ##$. .$ ## $.#########";

char\* Ex=&Example[0];

Sokoban B=Sokoban(Ex);

cout<<"1"<<endl;

B.print\_puzzle();

Sokoban A=Sokoban(1);

cout<<"2"<<endl;A.print\_puzzle();

A.move\_up();

cout<<"3"<<endl;A.print\_puzzle();

A.move\_right();

cout<<"EE441 "<<endl;A.print\_puzzle();

A.move\_down();

cout<<"HCN"<<endl;A.print\_puzzle();

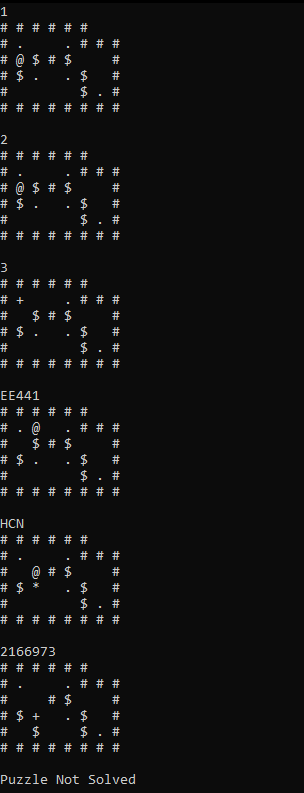
A.move\_down();

cout<<"2166973"<<endl;A.print\_puzzle();

if(A.is\_solved())

cout<<"Puzzle Solved"<<endl;

else



cout<<"Puzzle Not Solved"<<endl;

**Q2)** **Implement a mixed StackQueue template class.**

const int nSize=100;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue Declaration

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

class StackQueue{

private:

T mSQ[nSize];//memory element

int mTop;//indicates the state of the stack

public:

StackQueue();//default constructor

StackQueue(const StackQueue<T>& obj);//copy constructor

StackQueue<T>& operator=(const StackQueue<T>& rhs);//assignment operator

void Push\_front(const T& item);//Push method

T Pop\_front();//pop method

T Pop\_rear();//pop rear method

T Peek() const;//peek method

bool Empty() const;//true if stack is empty

bool Full() const;//true if stack is full

};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//**class StackQueue Method Definitions**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class **StackQueue** **default constructor**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

StackQueue<T>::StackQueue():mTop(0){}//default constructor assigns 0 to mTop

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue **copy constructor**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

StackQueue<T>::StackQueue(const StackQueue<T>& obj){

mTop=obj.mTop;//copy the state of the obj stackqueue

if(!obj.Empty())//checks if the obj was empty

for(int i=0;i<mTop;i++)//copy each element till the the mTop^th element

mSQ[i]=obj.mSQ[i];}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue **overloading assignment operator**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

StackQueue<T>& StackQueue<T>::operator=(const StackQueue<T>& rhs){

mTop=rhs.mTop;//copy the state of the obj stackqueue

if(!rhs.Empty())//checks if the obj was empty

for(int i=0;i<mTop;i++)//copy each element till the the mTop^th element

mSQ[i]=rhs.mSQ[i];

return\* this;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue **empty method**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

bool StackQueue<T>::Empty()const{

if(mTop==0)//only checking the state of the stack queue using mTop

return true;

else

return false;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue **full method**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

bool StackQueue<T>::Full()const{

if(mTop==nSize-1)//similar to empty method checking state only using mTop

return true;

else

return false;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue **push front method**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

void StackQueue<T>::Push\_front(const T& item){

if(!Full()){//action taken if the stack is not full

mSQ[mTop]=item;//item is added to stack

mTop++;}//state of the stack is changed

else

cout<<"Error: SQ is full!"<<endl; }//prints out a message if the stack is full

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue **pop front method**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

T StackQueue<T>::Pop\_front(){

if(!Empty()){//action taken for non-empty stack

mTop--;//state of the stack is changed

return mSQ[mTop];}//popped item returned

else{//if empty

T UNDEF;//uninitialized variable

cout<<"Error: SQ is empty!"<<endl;//prints out a message if the stack is empty

return UNDEF;}}//stack returnes declared but uninitialized variable

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue **pop rear method**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

T StackQueue<T>::Pop\_rear(){

if(!Empty()){//action taken for non-empty stack

T temp = mSQ[0];//return first added element

for(int i=0;i<mTop-1;i++)//shifting each variable in the stack

mSQ[i]=mSQ[i+1];

mTop--;//state of the stack is changed

return temp;}//popped item returned

else{

T UNDEF;//uninitialized variable

cout<<"Error: SQ is empty!"<<endl;//prints out a message if the stack is empty

return UNDEF;} }//stack returnes declared but uninitialized variable

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue **peek method(front)**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

T StackQueue<T>::Peek()const{

if(!Empty()){//action taken for non-empty stack

return mSQ[mTop-1];}//return last added element

else{

T UNDEF;//uninitialized variable

cout<<"Error: SQ is empty!"<<endl;//prints out a message if the stack is empty

return UNDEF;}}//stack returnes declared but uninitialized variable

**Q3)** **In your main function, instantiate a Sokoban class and a StackQueue class with the template argument Sokoban. You can use the puzzle in Q1 as the initial state. Write a code to respond to given key press events.**

I have made an infinite loop for this question. Key press events are handled with ‘cin>>’ command.

int main()

{

Sokoban A=Sokoban(1);

// cout<<"2"<<endl;

A.print\_puzzle();

StackQueue<Sokoban> S1;//S1 stackqueue of Sokoban template class

S1.Push\_front(A);//added A to S1

int i = 1;//initialize i=1

char X;//declared input character

while(i==1){//infinite loop

cin>>X;

**// Movement keys: w, a, s, d; Go Back: z; Replay: r; Quit: o**.

if(X=='o')//pressing 'o' will cause exit the program

return 0;

else if(X=='w'&& A.move\_up()){

A.print\_puzzle();

S1.Push\_front(A);

}

else if(X=='a'&& A.move\_left()){

A.print\_puzzle();

S1.Push\_front(A);

}

else if(X=='s'&& A.move\_down()){

A.print\_puzzle();

S1.Push\_front(A);

}

else if(X=='d'&& A.move\_right()){

A.print\_puzzle();

S1.Push\_front(A);

}

else if(X=='z'){

S1.Pop\_front();

if(!S1.Empty())

A=S1.Peek();// A is used as a temp variable

else

S1.Push\_front(A);//if S1 is empty then A is loaded again

A.print\_puzzle();//so S1 is never becomes empty outside of this else if statement

}

else if(X=='r'){

StackQueue<Sokoban> S2(S1);

//S2=S1;

while(!S2.Empty())

S2.Pop\_rear().print\_puzzle();

return 0;

}

if(A.is\_solved())

cout<<"Puzzle Solved"<<endl;//printed if the puzzle is solved

}

return 0;

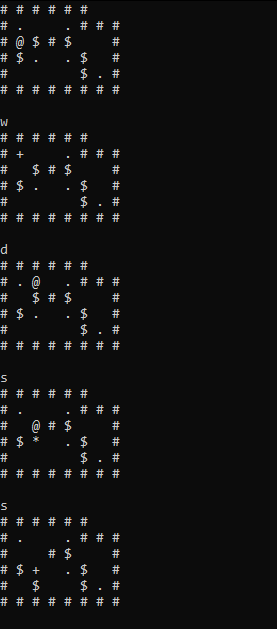
}

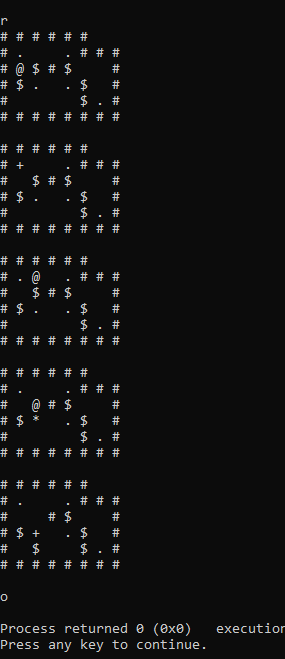
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//TEST RUN//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

w, d, s, s, r, o key presses were performed.





**Q4)** **Implement a doubly-linked list class.**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node Decleration

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template<class T>

class node

{

private:

node<T>\* mNext;//pointer for next node

node<T>\* mPrev;//pointer for previous node

public:

T mData;//data(memory element)

node();//constructor

node(const T& x);//alternative constructor

node \*getprev();//returns pointer to previous node

node \*getnext();//returns pointer to next node

void InsertF(node<T> \*p);//inserts a node poited by p to front of the current node

void InsertB(node<T> \*p);//inserts a node poited by p to back of the current node

node<T>\* DeleteF();//deletes the node front of the current node

node<T>\* DeleteB();//deletes the node back of the current node

};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node default constructor

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>//initally both pointers of the node points the node itself

node<T>::node() : mNext(this), mPrev(this){}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node alternative constructor

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>//initally both pointers of the node points the node itself

node<T>::node(const T& x) : mNext(this), mPrev(this), mData(x) {}//node also have data member initialized

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node getprev method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>//returns pointer to previous node

node<T>\* node<T>::getprev(){return mPrev;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node getnext method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>//returns pointer to next node

node<T>\* node<T>::getnext(){return mNext;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node insertfront method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>//inserts another node front in 4 steps

void node<T>::InsertF(node<T> \*p){

p->mNext=mNext;

p->mPrev=this;

p->getnext()->mPrev=p;

mNext=p;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node InsertBack method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>//inserts another node at back in 4 steps

void node<T>::InsertB(node<T> \*p){

p->getprev=mPrev;

p->mNext=this;

p->getPrev->mNext=p;

mPrev=p;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node DeleteFront Method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

node<T>\* node<T>::DeleteF(){//deletes node at front

node\* temp=getnext();//save the memory location to delete the node later

if (temp!=this){//deletion will not happen if this is a header for the stackqueue class.

getnext()->getnext()->mPrev=this;//2 pointers edited for skipping an element in the list

mNext=getnext()->mNext;}

return temp;}//pointer to node is returned so that it can be deleted

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class Node DeleteBack Method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

node<T>\* node<T>::DeleteB(){//deletes node at back

node\* temp=getprev();//save the memory location to delete the node prior

// T tempdata=temp->mData;//data stored in the node back is saved

if (temp!=this){//deletion will not happen if this is a header for the stackqueue class.

getprev()->getprev()->mNext=this;//2 pointers edited for skipping an element in the list

mPrev=getprev()->mPrev;}

return temp; }//data hold in the deleted node is returned

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class **StackQueue Decleration**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

class StackQueue{

private:

int mTop;//holds number of memory elements in SQ

node<T> mHead;//defined the head node

node<T>\* mSQHead;//=&mHead;//head node pointer

void ClearSQ();//clears the stackqueue

public:

StackQueue();//default constructor

~StackQueue();//destructor

StackQueue(const StackQueue<T>& obj);//copy constructor

StackQueue<T>& operator=(const StackQueue<T>& rhs);//overloading assignment operator

void Push\_front(const T& item);//method for Push operation

T Pop\_front();//popfront method

T Pop\_rear();//poprear method

T Peek() const;//peek method

bool Empty() const;//checks if the stackqueue is empty

//bool Full() const;//not necessary anymore

};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue default constructor

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

StackQueue<T>::StackQueue():mTop(0),mSQHead(&mHead){}//size is set to zero

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue destructor

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

StackQueue<T>::~StackQueue(){ClearSQ();}//deletes all dynamically allocated nodes

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue copy constructor

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

StackQueue<T>::StackQueue(const StackQueue<T>& obj){

mTop=0;//size set to zero

node<T>\* temp = obj.mSQHead;//temporary pointer is initialized for iteration

if(!obj.Empty())//if object is not empty

for(int i=0;i<obj.mTop;i++){//number of iterations determined by the size

temp=temp->getprev();//because the list is circular temp(obj.mSQHead) is iterated backwards

Push\_front(temp->mData);}}//nodes pushed to the current object

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue overloaded assignment operator

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

StackQueue<T>& StackQueue<T>::operator=(const StackQueue<T>& rhs){

ClearSQ();//clearing SQ to make sure memory is freed clears dynamically allocated nodes.

if(mTop!=0) cout<<"Error"<<endl;

node<T>\* temp = rhs.mSQHead;//temporary pointer is initialized for iteration

if(!rhs.Empty())//if object is not empty

for(int i=0;i<rhs.mTop;i++){//number of iterations determined by the size

temp=temp->getprev();//because the list is circular temp is iterated backwards

Push\_front(temp->mData);}//nodes pushed to the current object

return\* this;}//returns current object

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue empty method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

bool StackQueue<T>::Empty()const{

if(mTop==0)//checks the size of the stackqueue

return true;

else

return false;}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue clearSQ method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

void StackQueue<T>::ClearSQ(){

while(!Empty()) Pop\_front();}//popfront method also deletes the allocated memory of a node

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue Push\_Front method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

void StackQueue<T>::Push\_front(const T& item){

mTop++;

node<T>\* ptr = new node<T>(item);//node is allocated

mSQHead->InsertF(ptr);}//node is placed in front of the head node like a stack

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue Pop\_front method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

T StackQueue<T>::Pop\_front(){

if(!Empty()){//if list is not empty

mTop--;

node<T>\* temp= mSQHead->DeleteF();//returned pointer is saved to free the allocated memory

T tempdata= temp->mData;// data in the node is saved to be returned

delete temp;

return tempdata;}

else{

cout<<"Error: SQ is empty!"<<endl;

return mSQHead->mData;}}//data hold by the head node is returned

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue Pop\_rear method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

T StackQueue<T>::Pop\_rear(){

if(!Empty()){//if list is not empty

mTop--;

node<T>\* temp= mSQHead->DeleteB();//returned pointer is saved to free the allocated memory

T tempdata= temp->mData;// data in the node is saved to be returned

delete temp;

return tempdata;}

else{

cout<<"Error: SQ is empty!"<<endl;

return mSQHead->mData;}}//data hold by the head node is returned

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//class StackQueue Peek Method

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

template <class T>

T StackQueue<T>::Peek()const{

if(!Empty()){//if list is not empty

return mSQHead->getnext()->mData;}//returns the content of first node

else{

cout<<"Error: SQ is empty!"<<endl;

return mSQHead->mData;}}//data hold by the head node is returned

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//MAIN FUNCTION

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(){

string Example="###### #. .####@$#$ ##$. .$ ## $.#########";

char\* Ex=&Example[0];

Sokoban A=Sokoban(Ex);

cout<<"1"<<endl;

A.print\_puzzle();

Sokoban B=Sokoban(1);

cout<<"2"<<endl;B.print\_puzzle();

B.move\_up();

cout<<"3"<<endl;B.print\_puzzle();

B.move\_right();

cout<<"EE441 "<<endl;B.print\_puzzle();

B.move\_down();

cout<<"HCN"<<endl;B.print\_puzzle();

B.move\_down();

cout<<"2166973"<<endl;B.print\_puzzle();

if(A.is\_solved())

cout<<"Puzzle Solved"<<endl;

else

cout<<"Puzzle Not Solved"<<endl;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//Question 3&4 as an infinite loop// Movement keys: w, a, s, d;

//Go Back: z; Replay&Quit: r; Quit: q.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

StackQueue<Sokoban> S1;//S1 stackqueue of Sokoban template class

S1.Push\_front(A);//added A to S1

A.print\_puzzle();

int i = 1;//initialize i=1

char X;//declared input character

while(i==1){//infinite loop

cin>>X;

// Movement keys: w, a, s, d; Go Back: z; Replay: r; Quit: o.

if(X=='q')//pressing 'o' will cause exit the program

return 0;

else if(X=='w'&& A.move\_up()){

A.print\_puzzle();

S1.Push\_front(A);

}

else if(X=='a'&& A.move\_left()){

A.print\_puzzle();

S1.Push\_front(A);

}

else if(X=='s'&& A.move\_down()){

A.print\_puzzle();

S1.Push\_front(A);

}

else if(X=='d'&& A.move\_right()){

A.print\_puzzle();

S1.Push\_front(A);

}

else if(X=='z'){

S1.Pop\_front();

if(!S1.Empty())

A=S1.Peek();// A is used as a temp variable

else

S1.Push\_front(A);//if S1 is empty then A is loaded again

A.print\_puzzle();//so S1 is never becomes empty outside of this else if statement

}

else if(X=='r'){

StackQueue<Sokoban> S2;//(S1);

S2=S1;

while(!S2.Empty())

S2.Pop\_rear().print\_puzzle();

return 0;

}

if(A.is\_solved())

cout<<"Puzzle Solved"<<endl;//printed if the puzzle is solved

}

return 0;

}