List

-> [ ]->[ ]. .

A.)Insert 5 in front of the list

List

->[20|]->[30|]->. . .

P

->[5|] //connect this to the beginning of the list above

Struct node

{

Int info;

node \*next

};

node \*p

p=new(node);

p->info=5;

i.) p->next=list;

ii.) list=p;

//list is now [5|]->[20|]->[30|]

B.) insert 50 at the rear of the list

list

-> [5|]->[20|]->[30|]->[10|]

p = new(node); //p->[50|] we want this to be the last node

p->info=50;

p->next=NULL; //we want it to be 0 because it is the end and we want to insert it in the rear

node \*q;

q=list; //we using q to go through the lists

while(q->next != NULL) //q now starts at the first pointer of list which is [5|]

{

q=q->next; //now points to the next address in the list

}

//now we want to connect the last node of q to p bec p will be last

q->next=p;

c.) -------------------delete the first node----------------

list

->[20|]->[30|]->[10|]->[5|]

remove(delete) the first node.

//we need a pointer to the first node

node \*p;

p=list; //p is now pointing to first node.

//delete 20 and make list point to 30 and make that one first

i.) list=list->next; //now list is pointing to 30

//predefined function delete is to get rid of that location

delete(p); //p exists but the location is gone or rather we made that location empty .. we made that //room available for future use like an apartment

d.)-------------------delete the last node in the ff list---------------------

list

->[35|]->[50|]->[40|]->[50|] //don’t look for the value of the last node bec there might be duplicates

//we use a apointer to point to the last node.

//we use NULL to compare the pointers to last

//we set pointer to point to 40

//we create another pointer to p oint to 40 to put null at the end.. we use q

//we start p and q to point at first node .. we want p to point to last node.

//we do while p->next != NULL

//when p moves forward q will follow p from behind this trick when we want 2 pointers to follow eother refer to while loop below

p=q=list; //all p q list are pointing to same location

//for p we know what condition p->next!=NULL

while(p->next!=NULL)

{

q=p;// q will follow p now

}

delete(p);

q->next=NULL; //now q qbecomes the last node while the last pointer location p is deleted

//the order doesn’t matter if u do delete p or q next null

----------------------pointer implementation of stack-----------------------

class stack\_ptr

{

private:

struct node

{

int info;

node \*next;

}//end struct

node \*stackl //stack is a pointer to node

//in public we need a member to do insertion, add and remove to the list

//insertions are done at the beginning of the list

public:

stack\_ptr(){stack=NULL;} //when we create an object it initially points to null. this constructor

void push\_s(int x)

{

node \*p;

p=new(node);

p->info=x;

//we now have p->[x|] rather than it being empty

p->next=NULL; //last of stack so far

if(stack==NULL)

{

stack=p;

}//endif

else

{

p->next=stack;

stack=p;

}//endelse

//all we need to add something to stack

}//end push\_s

bool empty\_s()

{

//we know it’s empty when it’s NULL;

if(stack==NULL)return true;

else return false;

}//end empty\_s

int pop\_s()

{

//delete the first item in the stack by pointing to is

node \*p; int x=stack->info;

p=stack; //p is pointing to the first node

stack=stack->next;

delete(p);

return x;

}//end pop\_s

}//end class stack\_ptr

int main()

{

stack\_ptr S; //once we have this we call constructor and S->[ |/] is pointing to NULL

S.push\_s(10); //p->[10|/] //is stack == NULL? we execute the while loop

//S is now pointing to first node S->[10|/]

S.push(15); //so far S->[10|] pointing to 10 so now create a new node P->[15|/] since // p->next is NULL since p connected to s now S->[15|]

//we have p->[15|]->[10|/]

//--------------display the stack--------------------

//as long as s is not empty

while(!S.empty\_s())

{

int r=s.pop();

cout<<r<<”->”;

}//endwhile

cout<<”NULL”<<endl; //output will be 10->15->NULL

}

//--------------to change to template------------

template<class T> //we don’t need the size/length like before the n

class stack\_ptr

{

private:

struct node

{

int info;

node \*next;

}//end struct

node \*stackl //stack is a pointer to node

//in public we need a member to do insertion, add and remove to the list

//insertions are done at the beginning of the list

public:

stack\_patr(){stack=NULL;} //when we create an object it initially points to null. this constructor

void push\_s(T x)

{

node \*p;

p=new(node);

p->info=x;

//we now have p->[x|] rather than it being empty

p->next=NULL; //last of stack so far

if(stack==NULL)

{

stack=p;

}//endif

else

{

p->next=stack;

stack=p;

}//endelse

//all we need to add something to stack

}//end push\_s

bool empty\_s()

{

//we know it’s empty when it’s NULL;

if(stack==NULL)return true;

else return false;

}//end empty\_s

T pop\_s()

{

//delete the first item in the stack by pointing to is

node \*p; T x=stack->info;

p=stack; //p is pointing to the first node

stack=stack->next;

delete(p);

return x;

}//end pop\_s

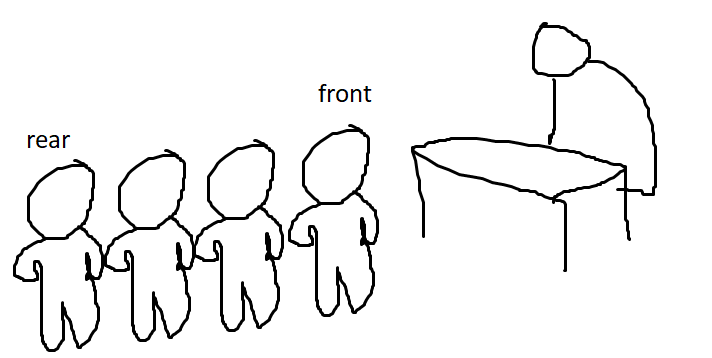
}//end class stack\_ptr

//in main we go stack\_ptr<int> //and we don’t need the size of the stack

--------------POINTER IMPLEMENTATION OF QUEUE--------------------

FIRST IN FIRST OUT (FIFO) //for queue the difference is you delete the last item) ie. insert in front of the list, but remove from the rear of the list

//we’ll have the pointer to point in front of queue



//when one person is there he is both front and rear.

when another person comes he becomes the rear while original person is still front

class queue\_ptr

{

private:

struct node\_q

{

int info;

node \*next;

};//end node\_q

~~node \*queue;~~ node \*front, \*rear;

public:

queue\_ptr(){front=rear=NULL;} //constructor

void push\_q(int x) //

{

node \*p;

p=new(node);

p->info=x;

p->next=NULL;

if(front==NULL) //means theres nothing there yet front=rear=p

{

front=rear=p;

}//endif

else

{

rear->next=p; //now p->[x|] is the last one and we want it to be rear

rear=p;

}//endelse

}//end push\_q

bool empty\_q()

{

//it’s empty if theres nothing in the front

if(front==NULL) return true; else return false;

}//empty\_q

int pop\_q()

{

node \*p;

p=front;

int x=front->info; //front and p ->[x |]->[ |]

front=front->next;

delete(p);

return x;

}//end pop\_q

}//end queue\_ptr

int main()

{

queue\_ptr Q; //front and rear ->[ |/]

Q.push\_q(10); //front and rear ->[ |/] and p->[10|/]

//now we have front p and rear ->[10|/]

Q.push\_q(5); //p is now pointing to new node 5 p->[5|/]

//rear->next=p;

//rear->[5|/]

}//end main

//----------to make it to a template-------------

template<class G>

//pmuch repeat same thing from above rename the ints to G

**for 5 extra points BOIIIII**

**TYPE A STACK AND QUEUE**

**GIVEN arrays**

**int a[5]={9,3,2,8,10};**

**string days[7]={“Mon”, “Tue”, .. . “Sun”};**

**write a program to use pointer implementation of stack and queue to display array a in reverse order**

**and array days in same order**

**remember: STACK IS LAST IN FIRST OUT LIFO AND QUEUE IS FIRST IN FIRST OUT FIFO**