Part A:  
Write two functions CreateDList(), that creates a doubly linked list of five nodes and   
DisplayAllNodes(),   
that prints all the nodes values in the sequence.  
Node 1 value: 10  
Node 2 Value: 20  
Node 3 value: 30

#include <iostream>

using namespace std;

class node{

public:

int roll;

node\* next;

node\* prev;

node\* create\_node(){

node\* newnode = new node;

return newnode;

}

void read\_node(node\* newnode){

cout<<"ENTER THE ROLL NO : ";cin>>newnode->roll;cout<<endl;

}

};

node\* head = NULL;

class list: public node{

public:

void insert\_at\_head(node\* newnode){

if(head==NULL){

head = newnode;

newnode->next = NULL;

}

else{

ascend\_from\_head(newnode);

}

}

node\* find\_last(){

node\* temp = head;

while(temp->next!=NULL){

temp = temp->next;

}

node\* last = temp;

return last;

}

void ascend\_from\_head(node\* newnode) {

node\* last = find\_last();

if(newnode->roll>last->roll){

last->next = newnode;

newnode->prev = last;

newnode->next = NULL;

}

else if(newnode->roll<head->roll){

newnode->next = head;

head->prev = newnode;

head = newnode;

}

else{

node\* first = head->next;

node\* second = first->next;

if(newnode->roll<first->roll){

node\* first\_prev = first->prev;

first\_prev->next = newnode;

newnode->next = first;

first->prev = newnode;

newnode->prev = first\_prev;

}

else{

first = first->next;

second = second->next;

}

}

}

void display(){

if(head==NULL){

cout<<"NOTHING TO DISPLAY"<<endl;

}

else{

node\* temp = head;

while(temp!=NULL){

cout<<temp->roll<<endl;

temp = temp->next;

}

}

}

};

int main(){

int option;

list l;

while(1){

cout<<"ENTER 1 TO INSERT AT HEAD"<<endl;

cout<<"ENTER 10 TO DISPLAY"<<endl;

cin>>option;cout<<endl;

switch(option){

case 1:

{

node\* newnode = l.create\_node();

l.read\_node(newnode);

l.insert\_at\_head(newnode);

break;

}

case 10:

{

l.display();

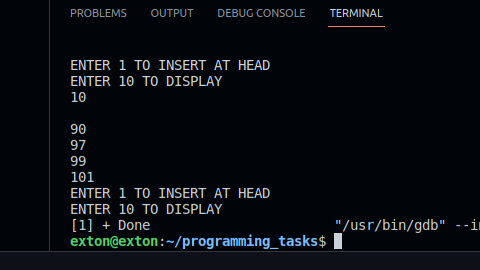
break;

}

}

}

}



Part B:  
Write a program to find the maximum value from the doubly linked list and delete it.   
Create three functions DeleteNode() and Maxvalue().

#include <iostream>

using namespace std;

class node{

public:

int roll;

node\* next;

node\* prev;

node\* create\_node(){

node\* newnode = new node;

return newnode;

}

void read\_node(node\* newnode){

cout<<"ENTER THE ROLL NO : ";cin>>newnode->roll;cout<<endl;

}

};

node\* head = NULL;

class list: public node{

public:

void insert\_at\_head(node\* newnode){

if(head==NULL){

head = newnode;

newnode->next = NULL;

}

else{

newnode->next = head;

head->prev = newnode;

head = newnode;

}

}

node\* find\_last(){

node\* temp = head;

while(temp->next!=NULL){

temp = temp->next;

}

node\* last = temp;

return last;

}

node\* find\_max(){

if(head==NULL){

cout<<"THE LIST IS EMPTY"<<endl;

}

else{

node\* temp = head->next;

node\* max\_node = NULL;

int max\_val = head->roll;

while(temp!=NULL){

if(temp->roll>max\_val){

max\_val = temp->roll;

max\_node = temp;

temp = temp->next;

}

else{

temp = temp->next;

}

}

cout<<"MAX : "<<max\_val<<endl;

node\* temp\_prev = max\_node->prev;

node\* temp\_next = max\_node->next;

node\* temp\_s = max\_node;

temp\_prev->next = temp\_next;

temp\_next->prev = temp\_prev;

delete(temp\_s);

return max\_node;

}

}

void display(){

if(head==NULL){

cout<<"NOTHING TO DISPLAY"<<endl;

}

else{

node\* temp = head;

while(temp!=NULL){

cout<<temp->roll<<endl;

temp = temp->next;

}

}

}

};

int main(){

int option;

list l;

while(1){

cout<<"ENTER 1 TO INSERT AT HEAD"<<endl;

cout<<"ENTER 2 TO FIND MAX"<<endl;

cout<<"ENTER 10 TO DISPLAY"<<endl;

cin>>option;cout<<endl;

switch(option){

case 1:

{

node\* newnode = l.create\_node();

l.read\_node(newnode);

l.insert\_at\_head(newnode);

break;

}

case 2:

{

node\* max\_val = l.find\_max();

//l.delete\_max(max\_val);

break;

}

case 10:

{

l.display();

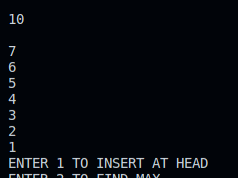
break;

}

}

}

}



Task 2  
Write a program to insert a node at the end of the doubly linked list without using   
findLast() function.

#include <iostream>

using namespace std;

class node{

public:

int roll;

node\* next;

node\* prev;

node\* create\_node(){

node\* newnode = new node;

return newnode;

}

void read\_node(node\* newnode){

cout<<"ENTER THE ROLL NO : ";cin>>newnode->roll;cout<<endl;

}

};

node\* head = NULL;

class list: public node{

public:

void insert\_at\_head(node\* newnode){

if(head==NULL){

head = newnode;

newnode->next = NULL;

}

else{

newnode->next = head;

head->prev = newnode;

head = newnode;

}

}

void insert\_at\_last(node\* newnode){

if(head==NULL){

head = newnode;

head->next = NULL;

}

else{

node\* temp = head;

while(temp->next!=NULL){

temp = temp->next;

}

temp->next = newnode;

newnode->prev = temp;

newnode->next = NULL;

}

}

node\* find\_last(){

node\* temp = head;

while(temp->next!=NULL){

temp = temp->next;

}

node\* last = temp;

return last;

}

node\* find\_max(){

if(head==NULL){

cout<<"THE LIST IS EMPTY"<<endl;

}

else{

node\* temp = head->next;

node\* max\_node = NULL;

int max\_val = head->roll;

while(temp!=NULL){

if(temp->roll>max\_val){

max\_val = temp->roll;

max\_node = temp;

temp = temp->next;

}

else{

temp = temp->next;

}

}

/\*cout<<"MAX : "<<max\_val<<endl;

node\* temp\_prev = max\_node->prev;

node\* temp\_next = max\_node->next;

node\* temp\_s = max\_node;

temp\_prev->next = temp\_next;

temp\_next->prev = temp\_prev;

delete(temp\_s);

return max\_node;\*/

}

}

void display(){

if(head==NULL){

cout<<"NOTHING TO DISPLAY"<<endl;

}

else{

node\* temp = head;

while(temp!=NULL){

cout<<temp->roll<<endl;

temp = temp->next;

}

}

}

};

int main(){

int option;

list l;

while(1){

cout<<"ENTER 1 TO INSERT AT HEAD"<<endl;

cout<<"ENTER 2 TO FIND MAX"<<endl;

cout<<"ENTER 3 TO INSERT AT END"<<endl;

cout<<"ENTER 10 TO DISPLAY"<<endl;

cin>>option;cout<<endl;

switch(option){

case 1:

{

node\* newnode = l.create\_node();

l.read\_node(newnode);

l.insert\_at\_head(newnode);

break;

}

case 2:

{

node\* max\_val = l.find\_max();

//l.delete\_max(max\_val);

break;

}

case 3:

{

node\* newnode = l.create\_node();

l.read\_node(newnode);

l.insert\_at\_last(newnode);

break;

}

case 10:

{

l.display();

break;

}

}

}

}

