2. #include<bits/stdc++.h>

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int\* key;

Node\* next;

Node(){next = NULL;}

Node(int value){

key = new int (value);

next = NULL;

}

~Node(){}

};

void pushFront(int key);

int\* topFront();

void print();

void numberOfElements();

Node\* head = NULL;

Node\* tail = NULL;

int main(){

int choice;

int key;

int\* p;

pushFront(5);

pushFront(4);

pushFront(3);

pushFront(2);

pushFront(1);

print();

//write your code

//deleteFirst();

numberOfElements();

}

void pushFront(int key){

Node\* node = new Node(key);

node->next = head;

head = node;

if(tail == NULL)

tail = head;

}

int\* topFront(){

return head->key;

}

void print(){

Node\* p = head;

if(head==NULL){

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

return;

}

while(p!= NULL){

cout << \*(p->key) << " -> ";

p = p->next;

}

cout << " NULL " << endl;

}

void numberOfElements(){

int c=0;

Node\* p=head;

while(p!=NULL){

c++;

p=p->next;

}

cout<<c<<endl;

}

3 #include<bits/stdc++.h>

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int\* key;

Node\* next;

Node(){next = NULL;}

Node(int value){

key = new int (value);

next = NULL;

}

~Node(){}

};

void pushFront(int key);

int\* topFront();

void print();

void minusElements();

Node\* head = NULL;

Node\* tail = NULL;

int main(){

int choice;

int key;

int\* p;

pushFront(5);

pushFront(4);

pushFront(3);

pushFront(2);

pushFront(1);

print();

//write your code

minusElements();

}

void pushFront(int key){

Node\* node = new Node(key);

node->next = head;

head = node;

if(tail == NULL)

tail = head;

}

int\* topFront(){

return head->key;

}

void print(){

Node\* p = head;

if(head==NULL){

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

return;

}

while(p!= NULL){

cout << \*(p->key) << " -> ";

p = p->next;

}

cout << " NULL " << endl;

}

void minusElements(){

Node\*p=head;

int a;

while(p->next!=NULL){

a=\*(p->key)-\*(p->next->key);

cout<<a<<" ";

p=p->next;

}

}

\*\*\*\*\*\*\*\*\*\*\*SIMPLE LIST ADT

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

char\* key;

Node\* next;

Node(){next = NULL;}

Node(int value){

key = new char (value);

next = NULL;

}

~Node(){}

};

class Stack{

public:

Node\* head;

Node\* tail;

Stack(){

head = NULL;

tail = NULL;

}

void pushFront(char key);

char\* topFront();

void print();

void popFront();

};

int main(){

int choice;

char key;

int\* p;

Stack\* list1 = new Stack();

list1->pushFront('5');

list1->pushFront('4');

list1->pushFront('3');

list1->pushFront('2');

list1->pushFront('1');

list1->print();

Stack\* list2 = new Stack();

list2->pushFront('5');

list2->pushFront('4');

list2->pushFront('3');

list2->pushFront('2');

list2->pushFront('1');

list2->print();

list1->popFront();

list1->print();

// write your code

}

void Stack::pushFront(char key){

Node\* node = new Node(key);

node->next = head;

head = node;

if(tail == NULL)

tail = head;

}

char\* Stack::topFront(){

return head->key;

}

void Stack::print(){

Node\* p = head;

if(head==NULL){

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

return;

}

while(p!= NULL){

cout << \*(p->key) << " -> ";

p = p->next;

}

cout << " NULL " << endl;

}

void Stack::popFront(){

if(head==NULL){

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

}

head=head->next;

}

\*\*\*\*\*\*\*SIMPLE LIST

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int\* key;

Node\* next;

Node(){next = NULL;}

Node(int value){

key = new int (value);

next = NULL;

}

~Node(){}

};

void pushFront(int key);

int\* topFront();

void print();

void deleteFirst();

Node\* head = NULL;

Node\* tail = NULL;

int main(){

int choice;

int key;

int\* p;

pushFront(5);

pushFront(4);

pushFront(3);

pushFront(2);

pushFront(1);

print();

//write your code

deleteFirst();

}

void pushFront(int key){

Node\* node = new Node(key);

node->next = head;

head = node;

if(tail == NULL)

tail = head;

}

int\* topFront(){

return head->key;

}

void print(){

Node\* p = head;

if(head==NULL){

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

return;

}

while(p!= NULL){

cout << \*(p->key) << " -> ";

p = p->next;

}

cout << " NULL " << endl;

}

void deleteFirst(){

Node\* p=head->next;

if(head==NULL){

cout<<"";

}

else{

while(p!=NULL){

cout<<\*(p->key)<<" ";

p=p->next;

}

}

cout<<endl;

}

//void NumberOfElement(){

// Node\* p =head;

//while()

//}

\*\*\*\*SATCK adt bALANCE PARENTETHIS

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

void\* key;

Node\* next;

};

class Stack{

public:

Node\* head;

Stack(){

head=NULL;

}

void push(void\* key, int bytes);

void pop();

void\* top();

bool empty();

};

void Stack::push(void\* key, int bytes){

Node\* node = new Node;

node->key = malloc(bytes);

memcpy(node->key, (char\*) key, bytes);

node->next = head;

head = node;

//cout << "item pushed" << endl;

}

void Stack::pop(){

if(head==NULL){

cout << "Stack is empty" << endl;

return;

}

Node\* temp = head;

head = head->next;

delete temp;

//cout << "item popped" << endl;

}

void\* Stack::top(){

return head->key;

}

bool Stack::empty(){

if(head==NULL)

return true;

else

return false;

}

template<typename T>

void print(Stack\*, T\*);

int main(){

//Stack\* stack1 = new Stack;

Stack\* stack=new Stack;

char value[1024];

cin.getline(value,1024);

int i=0;

while(value[i]!='\0'){

if(value[i]=='('||value[i]=='{'||value[i]=='['||value[i]==')'||value[i]=='}'||value[i]==']'){

stack->push(&value[i],1); //1 byte for character

}

i++;

}

int j=0;

while(value[j]!='\0'){

if(value[0]==')'||value[0]=='}'||value[0]==']'){

//cout<<"Not balanced"<<endl;

break;

}

else if(value[j]=='(' && \*(char\*)(stack->top())==')'){

stack->pop();

}

else if(value[j]=='{' && \*(char\*)(stack->top())=='}'){

stack->pop();

}

else if(value[j]=='[' && \*(char\*)(stack->top())==']'){

stack->pop();

}

else if(value[j]==')' && \*(char\*)(stack->top())=='('){

stack->pop();

}

else if(value[j]=='}' && \*(char\*)(stack->top())=='{'){

stack->pop();

}

else if(value[j]==']' && \*(char\*)(stack->top())=='['){

stack->pop();

}

j++;

}

if(stack->empty()==true){

cout<<"Balanced"<<endl;

}

else if(stack->empty()==false){

cout<<"Not balanced"<<endl;

}

//char key;

// reading a character array

//char name[1024];

//cin.getline(name, 1024);

// reading a string

//string name;

//getline(cin, name);

/\* cout << "This program will create a CHARACTER stack" << endl;

// creating a stack

cout << "Enter items to push (x to stop)" << endl;

while(true){

cin >> key;

if(key == 'x')

break;

stack1->push(&key, 1);

}

// popping two elements

stack1->pop();

stack1->pop();

// printing the top after popping two elements

cout << "Top: " << \*(char \*)stack1->top() << endl;

// printing the complete stack using a template function

print(stack1, &key);\*/

}

// using a template fucntion (Generic methods in Java)

// if you don't understand, write your own print function accordingly

template <typename T>

void print(Stack\* stack, T\* a){

Node\* p = stack->head;

if(p==NULL){

cout << "Stack is empty" << endl;

return;

}

while(p!= NULL){

cout << \*(T\*) (p->key)<< " ";

p = p->next;

}

cout << endl;

}