```cpp
/* Lab1A.cpp */
#include <iostream>
#include <string>

using namespace std;

struct object {
    string* s;
};

int main() {
    object A, B;
    A.s = new string;
    B.s = new string;
    *A.s = "Hello World"; // A has "Hello World"
    *B.s = *A.s; // B has "Hello World"
    *A.s = "Goodbye"; // A has "Goodbye"

    cout << *A.s << endl;
    cout << *B.s << endl;

    return 0;
}
```

```cpp
/* Lab1B.cpp */
#include <iostream>

using namespace std;

int main() {
    int score[10];
    for (int i=0; i < 10; i++) {
    score[i]=3*i;
    cout << score[i];
}

    return 0;
}
```

```cpp
/*Lab1C.cpp*/
#include <iostream>

using namespace std;
const double PI = 3.14159265359;
void GetRadius(double&);
void ShowResults(double, double, double);

int main() {
    cout << "Program computes surface area and "
     << "volume of a sphere.\n";

    double radius, // radius of sphere
    surfaceArea = 0, // its surface area
    volume = 0; // its volume
    GetRadius(radius);
    surfaceArea = 4.0 * PI * radius * radius;
    volume = surfaceArea * radius / 3.0;
    ShowResults(radius, surfaceArea, volume);

    return 0;
}

void GetRadius(double& rad) {
    cout << "Enter radius of sphere: ";
    cin >> rad;
}

void ShowResults(double rad, double area, double vol) {
    cout << "Radius of sphere is " << rad << " inches\n";
    cout << "Its surface area is " << area
     << "sq. inches\n" << "Its volume is " << vol
     << " cubic inches.\n\n";
}
```

Names: Yousef Zoumot, Morgan Gillis
Date: 1/5/16
Lab Section: Tues 9:15 am
Class: COEN 70

PROBLEM 1:
Object B was initially made to point to whatever A was pointing to, which meant that if you changed A, you changed B. We decided to change this so that B would point to its own memory location that had the same value as A's. We also had to declare B as a new string, similar to A.

PROBLEM 2:
The loop syntax was outside the bounds of the array. The problem was that i started at 1 and ended at 10, even though the indexes of the array went from 0 to 9. So we changed it to (i=0; i < 10; i++).

PROBLEM 3:
We had to declare the surface area formula before the volume one because the volume formula uses the value of the surface area to compute its own value.

```
//  Yousef Zoumot Ken Wakaba
//  main.cpp
//  Coen70Lab2
//
//  Created by Yousef Zoumot & Ken Wakaba on 1/12/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
using namespace std;

class Complex{
private:
    double a;
    double b;
    char imaginary='i';
public:
    double real(){return a;};
    double imagine(){return b;};
    void setReal(double x){a=x;};
    void setImaginary(double y){b=y;};
    Complex(double n1, double n2);
    Complex();
    friend istream& operator >>(istream& ins, Complex& og1);
    friend ostream& operator <<(ostream& ins, Complex og1);

};

Complex::Complex(double n1, double n2){
    a=n1;
    b=n2;
}

Complex::Complex(){
    a=0;
    b=0;
}

Complex operator +(Complex og1, Complex og2){
    Complex temp;
    temp.setReal( (og1.real()+og2.real()) );
    temp.setImaginary( (og1.imagine()+og2.imagine()) );
    return temp;
}

Complex operator *(Complex og1, Complex og2){
    Complex temp;
    temp.setReal( (og1.real()*og2.real()) -
(og1.imagine()*og2.imagine()) );
    temp.setImaginary( (og1.real()*og2.imagine())+
```

```
(og1.imagine()*og2.real()) );
    return temp;
}

ostream& operator <<(ostream& ins, Complex og1){
    ins<<og1.real()<<" + "<<og1.imagine()<<"i\n";
    return ins;
}

istream& operator >>(istream& ins, Complex& og1){
    double real, imagine;
    cout<<"Please enter real number \n";
    ins>>real;
    og1.setReal(real);
    cout<<"Please enter imaginary number \n";
    ins>>imagine;
    og1.setImaginary(imagine);
    return ins;



}



int main(int argc, const char * argv[]) {
    // insert code here...
    Complex c1;
    Complex c2(1.0, 2.0);
    c1.setReal((3.0));
    cout<<c1;
    //cin>>c2;
    cout<<c2;
    Complex c3= c1+c2;
    cout<<c3;
    Complex c4= c1*c2;
    cout<<c4;
    std::cout << "Hello, World!\n";
    return 0;
}
```

```
//  Gabrielle Tordillos & Yousef Zoumot
//  main.cpp
//  Coen70Lab3
//  Paul Thurston
//  Created by Yousef Zoumot on 1/19/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
#include <sstream>
#include <cassert>
#include <stdlib.h>//

using namespace std;

class Stack{
public:
    double express[100];
    void push(double x);
    double pop();
    double top();
    bool empty();
    void printValues();
private:
    int used;
    int last;

};

void Stack:: push(double x){
    assert(last!=used-1);
    express[last++]=x;
    used++;
    return;
}
bool Stack:: empty(){
    if(used==0)
        return true;
    else{
        return false;
    }
}
double Stack:: pop(){
    assert(!empty());
    double temp= express[--used];
    last--;
    return temp;
}

double Stack:: top(){
```

```
    return express[last];
}

void Stack:: printValues(){
    cout<<"The values in the stack are: ";
    for(int i=0; i<used; i++){
        cout<<express[i]<<"\n";
    }
}

int main(int argc, const char * argv[]) {

    string expr, token;
    Stack s1;
    double temp1, temp2;
    getline(cin, expr);
    istringstream stream(expr);
    while(stream >> token){
        if(token=="+"){
            temp1=s1.pop();
            temp2=s1.pop();
            s1.push(temp1+temp2);
        }
        if(token=="-"){
            temp1=s1.pop();
            temp2=s1.pop();
            s1.push(temp2-temp1);
        }
        if(token=="/"){
            temp1=s1.pop();
            temp2=s1.pop();
            s1.push(temp2/temp1);
        }
        if(token=="*"){
            temp1=s1.pop();
            temp2=s1.pop();
            s1.push(temp1*temp2);
        }
        if(token!="+" && token!="-" && token!="/" && token!="*"){
            double temp=atof(token.c_str());
            s1.push(temp);
        }

    }
    s1.printValues();
}
```

```
//
//  main.cpp
//  Lab4
//
//  Created by Yousef Zoumot on 1/26/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
#include <assert.h>

using namespace std;

class Queue{
    struct Node{
        Node *next;
        int data;
    };
    Node * head;
public:
    Queue();
    ~Queue();
    Queue(const Queue& source);
    void enqueue(int x);
    int dequeue();
    bool isEmpty() const;
};

Queue::Queue(){
    head=NULL;
}

Queue::~Queue(){
    while(!isEmpty())
        dequeue();

}

Queue::Queue(const Queue& source){
    if(this==&source)
        return;
    while(!isEmpty())
        dequeue();
    if(source.isEmpty()){
        return;
    }
    Node* temp=source.head;
    while(temp!=NULL){
        enqueue(temp->data);
        temp=temp->next;
```

```
    }
}

void Queue:: enqueue(int x){
    Node* tmp= new Node;
    tmp->data=x;
    if(head==NULL){
        head=tmp;
        head->next=NULL;
        return;
    }
    if(x>=head->data){
        tmp->next=head;
        head=tmp;
        return;
    }
    Node* cursor=head;
    while(cursor->next!=NULL && cursor->next->data>x){
        cursor=cursor->next;
    }
    tmp->next=cursor->next;
    cursor->next=tmp;
    return;

}

int Queue:: dequeue(){
    assert(!isEmpty());
    Node * tmp= head;
    head=tmp->next;
    int _data=tmp->data;
    delete tmp;
    return _data;
}

bool Queue:: isEmpty()const{
    return head==NULL;
}




int main(int argc, const char * argv[]) {
    Queue q1;
    q1.enqueue(1);
    q1.enqueue(3);
    q1.enqueue(2);
    q1.enqueue(4);
    Queue q2(q1);
```

```
    cout<<q1.dequeue()<<"\n";
    cout<<q1.dequeue()<<"\n";
    cout<<q1.dequeue()<<"\n";
    cout<<q1.dequeue()<<"\n";
    cout<<q2.isEmpty()<<"\n";
    cout<<q2.dequeue()<<"\n";
    cout<<q2.dequeue()<<"\n";
    cout<<q2.dequeue()<<"\n";
    cout<<q2.dequeue()<<"\n";
    return 0;
}
```

```
//
//  List.h
//  Lab5
//
//  Created by Yousef Zoumot on 2/2/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//


#pragma once
#include <iostream>
#include <stdlib.h>

using namespace std;

class List{
protected:
    struct Node{
        Node* _prev;
        Node* _next;
        int _data;
        Node(int data, Node* prev = NULL, Node* next = NULL){
            this->_data = data;
            this->_prev = prev;
            this->_next = next;
        }
        int& data(){return _data;};
        Node*& next(){return _next;};
        Node*& prev(){return _prev;};

    };

    Node* cursor;
    int n;
public:
        List();
    List(const List& source);
    ~List();
    void start();
    void end();
    void advance();
    void reverse();
    int size();
    void insert(int data);
    void attach(int data);
    int current();
    void remove();
    void display();
    List& operator=(const List& other);
    friend ostream& operator<<(ostream &out, const List &other);
```

```
};
//
//  List.cpp
//  Lab5
//
//  Created by Yousef Zoumot on 2/2/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include "List.h"
#include <stdlib.h>
#include <assert.h>

using namespace std;
/*****************************************************/

List::List(){
    cursor = NULL;
    n = 0;
} /* end constructor */

List::~List(){}

/*****************************************************/

List::List(const List& source){
    operator=(source);
}

/*****************************************************/

void List::start(){
    while(cursor->prev() != NULL){
        cursor = cursor->prev();
    }
}

/*****************************************************/

void List::end(){
    while(cursor->next() != NULL){
        cursor = cursor->next();
    }
}

/*****************************************************/

void List::advance(){
    if(cursor->next() != NULL){
```

```
        cursor = cursor->next();
    }
}

/*****************************************************/

void List::reverse(){
    if(cursor->prev() != NULL){
        cursor = cursor->prev();
    }
}

/*****************************************************/

int List::size(){
    return n;
}

/*****************************************************/

int List::current(){
    assert(n != 0);
    return cursor->data();
}

/*****************************************************/

std::ostream& operator<<(std::ostream &out, const List& other){
    List::Node* tmp = other.cursor;
    while(tmp != NULL){
        out << tmp->data() << std::endl;
        tmp = tmp->next();
    }
    return out;
}


/*****************************************************/

List& List::operator=(const List& other){
    if(this != &other){
        while(size() != 0){
            remove();
        }
        Node* tmp = other.cursor;
        while(tmp->prev()!=NULL)
            tmp=tmp->prev();
        while(tmp != NULL){
            insert(tmp->data());
            tmp = tmp->next();
```

```
        }

    }
    return *this;
}

/*****************************************************/

void List::insert(int data){
    if(n == 0){
        cursor= new Node(data);
        n++;
    }
    else{
        Node* tmp = new Node(data);
        tmp->next() = cursor;
        tmp->prev() = NULL;
        cursor->prev() = tmp;
        cursor = tmp;
        n++;
    }
}

/*****************************************************/

void List::attach(int data){
    if(n == 0){
        cursor= new Node(data);
        n++;
    }
    else{
        Node* tmp = new Node(data);
        cursor->next() = tmp;
        tmp->prev() = cursor;
        tmp->next() = NULL;
        cursor = tmp;
        n++;
    }
}

/*****************************************************/

void List::remove(){
    Node* tmp = cursor;
    while(tmp != NULL){
        //if(tmp->data() == target){
            if(cursor->next() == NULL){
                cursor = cursor->prev();
                //cursor->next();
                delete tmp;
```

```
                n--;
            } else{
                cursor->prev()->next() = cursor->next();
                cursor->next()->prev() = cursor->prev();
                delete tmp;
                n--;
            }
        //}
    }
}

/*****************************************************/


void List::display(){
    start();
    Node* tmp = cursor;
    while(tmp != NULL){
        std::cout << tmp->data() << "\n";
        tmp = tmp->next();
    }
    std::cout << "The size of the list is: " << n << endl;
}
//
//  Order.h
//  Lab5
//
//  Created by Yousef Zoumot on 2/2/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#pragma once
#include <iostream>
#include "list.h"

class Ordered_List: public List {
public:
    Ordered_List();
    Ordered_List(const Ordered_List& source);
    ~Ordered_List();
    Ordered_List& operator=(const Ordered_List& other);
    friend ostream& operator<<(ostream &out, const Ordered_List
&other);
    void insert(int x);
    void attach(int x);
    void display();
};
//
//  Ordered_List.cpp
//  Lab5
```

```
//
//  Created by Yousef Zoumot on 2/2/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include "Ordered_List.h"
#include <stdlib.h>

using namespace std;

Ordered_List::Ordered_List(){}

/************************************************/

Ordered_List::Ordered_List(const Ordered_List& source){
    if(this != &source){
        while(size() != 0){
            remove();
        }
        Node* tmp = source.cursor;
        while(tmp->prev()!=NULL)
            tmp=tmp->prev();
        while(tmp != NULL){
            insert(tmp->data());
            tmp = tmp->next();
        }

    }
}

/************************************************************************
******/

Ordered_List::~Ordered_List(){
}

/************************************************************************
******/

Ordered_List& Ordered_List:: operator=(const Ordered_List&
other){
    if(this != &other){
        while(size() != 0){
            remove();
        }
        Node* tmp = other.cursor;
        while(tmp->prev()!=NULL)
            tmp=tmp->prev();
        while(tmp != NULL){
            insert(tmp->data());
```

```
            tmp = tmp->next();
        }

    }
    return *this;
}

/*****************************************************************
*********/

std::ostream& operator<<(std::ostream &out, const Ordered_List
&other){
    out<<(List&)other;
    return out;
}

/*****************************************************************
********/

void Ordered_List::insert(int x){
    Node* tmp = cursor;
    Node* tmp_data= new Node(x);

    //tmp_data->data() = x;

    if(n == 0 && cursor == NULL){
        cursor = tmp_data;
        n++;
        return;
    }
    while(tmp->prev() != NULL){
        tmp = tmp->prev();
    }
    while(x >= tmp->data() && tmp->next()!=NULL){
        tmp = tmp->next();
    }
    if(tmp->prev()!=NULL){
        tmp->prev()->next() = tmp_data;
        tmp_data->next()=tmp;
        tmp->prev()=tmp_data;
    }
    if(tmp->next()==NULL && tmp->data()<=x){
        tmp->next()=tmp_data;
        tmp_data->next()=NULL;
        tmp_data->prev()=tmp;
    }
    else{
        tmp_data->next() = tmp;
        tmp_data->prev() = tmp->prev();
        tmp->prev() = tmp_data;
```

```
    }
    n++;
    return;
}



/*****************************************************************
********/

void Ordered_List:: attach(int x){
    Node* tmp = cursor;
    Node* tmp_data;
    tmp_data->data() = x;

    while(tmp->prev() != NULL){
        tmp = tmp->prev();
    }
    if(n == 0 && cursor == NULL){
        cursor = tmp_data;
        n++;
        return;
    }
    while(x >= tmp->data()){
        tmp = tmp->next();
    }
    tmp->prev()->next() = tmp_data;
    tmp_data->next() = tmp;
    tmp_data->prev() = tmp->prev();
    tmp->prev() = tmp_data;
    n++;
    return;
}

void Ordered_List::display(){
    start();
    Node* tmp = cursor;
    while(tmp != NULL){
        std::cout << tmp->data() << "\n";
        tmp = tmp->next();
    }
    std::cout << "The size of the list is: " << n << endl;
}
```

```
//  Yousef Zoumot Joseph Phan Lab 6
//  main.cpp
//  Lab6
//
//  Created by Yousef Zoumot on 2/9/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
#include <stdlib.h>
#include <cstdlib>
#include <time.h>

using namespace std;

class Player{
public:
    Player(){}
    virtual ~Player(){}
    virtual int getGuess()=0;
    // Returns the players next guess, an integer between
    // 0 and 99 (inclusive)
    virtual void lastWasTooHigh(bool tooHigh)=0;
    // Called to inform the player that their last guess
    // was too high (if the tooHigh argument is true) or
    // too low (if the tooHigh argument is false)
    virtual void opponentsGuess(int guess, bool tooHigh)=0;
    // Called to tell the player what the opponent's last
    // guess was, and whether it was too high or too low.
    virtual void reset()=0;
    // Called to reset the player at the end of the game.
    // Any stored state about the last guess should be
    // cleared


};

class HumanPlayer: public Player{
private:
    int current_guess;
public:
    HumanPlayer():Player(){}
    virtual ~HumanPlayer(){}
    virtual int getGuess();
    // Returns the players next guess, an integer between
    // 0 and 99 (inclusive)
    virtual void lastWasTooHigh(bool tooHigh);
    // Called to inform the player that their last guess
    // was too high (if the tooHigh argument is true) or
    // too low (if the tooHigh argument is false)
```

```
    virtual void opponentsGuess(int guess, bool tooHigh);
    // Called to tell the player what the opponent's last
    // guess was, and whether it was too high or too low.
    virtual void reset();
    // Called to reset the player at the end of the game.
    // Any stored state about the last guess should be
    // cleared.
};


void HumanPlayer::reset(){}

void HumanPlayer:: lastWasTooHigh(bool tooHigh){
    if(tooHigh){
        cout<<"Answer was too High"<<endl;
    }
    else
        cout<<"Answer was too Low"<<endl;

}

int HumanPlayer:: getGuess(){
    cout<<"Please enter your guess: ";
    int x;
    cin>>x;
    current_guess = x;
    return x;
}

void HumanPlayer::opponentsGuess(int guess, bool tooHigh){
    cout<<"opponent's guess: " << guess<< endl;
    if(tooHigh){
        cout<<"The guess was too high. "<<"\n";
    }
    else{
        cout<<"The guess was too Low. "<< "\n";
    }
}

class ComputerPlayer: public Player{
private:
    int min;
    int max;
    int current_guess;
public:
    ComputerPlayer():Player(){
        min = 0;
        max = 100;
    }
    virtual ~ComputerPlayer(){}
```

```
    virtual int getGuess();
    // Returns the players next guess, an integer between
    // 0 and 99 (inclusive)
    virtual void lastWasTooHigh(bool tooHigh);
    // Called to inform the player that their last guess
    // was too high (if the tooHigh argument is true) or
    // too low (if the tooHigh argument is false)
    virtual void opponentsGuess(int guess, bool tooHigh);
    // Called to tell the player what the opponent's last
    // guess was, and whether it was too high or too low.
    virtual void reset();
    // Called to reset the player at the end of the game.
    // Any stored state about the last guess should be
    // cleared.

};

void ComputerPlayer:: lastWasTooHigh(bool tooHigh){
    if(tooHigh)
        min = current_guess + 1;
    else
        max = current_guess - 1;
}

int ComputerPlayer:: getGuess(){
    int mid = (min + max)/2;
    current_guess = mid;
    return mid;
}

void ComputerPlayer::reset(){
    min = 0;
    max = 100;
}

void ComputerPlayer::opponentsGuess(int guess, bool tooHigh){
    if(tooHigh && (max>guess))
        max = guess-1;
    if(!tooHigh && (min<guess))
       min = guess+1;
}


int main(int argc, const char * argv[]) {
    // insert code here...
    srand(time(NULL));
    HumanPlayer p1;
    ComputerPlayer p0;
    int num_players = 2;
    int answer= rand()%101;
```

```
    int player_guess;
    int play_again;
    int counter=0;
    bool wasHigh;
    bool end_game = false;
    while(true){
        counter++;
        for(int i=0; i<num_players; i++){
            //get guesses from human player
            player_guess = p1.getGuess();

            if(player_guess==answer){
                  cout<<"You win with" << counter << "
guesses."<<endl;
                cout<<"would you like to play again? 0 for no, 1
for yes"<< endl;
                cin>>play_again;
                if(play_again==1){
                    p0.reset();
                    p1.reset();
                    counter = 0;
                    answer= rand() % 101;
                    break;
                }
                else{
                    end_game = true;
                    break;
                }
            }

            wasHigh = (player_guess > answer);
            p1.lastWasTooHigh(wasHigh);

            p0.opponentsGuess(player_guess, wasHigh);
            player_guess = p0.getGuess();
            if(player_guess==answer){
                cout<<"You lost opponent won in " << counter << "
guesses."<<endl;
                cout<<"would you like to play again? 0 for no, 1
for yes"<< endl;
                cin>>play_again;
                if(play_again==1){
                    p0.reset();
                    p1.reset();
                    answer= rand()%101;
                    counter = 0;
                    break;
                }
                else{
                    end_game = true;
```

```
                    break;
                }
            }

            wasHigh = (player_guess > answer);
            p1.opponentsGuess(player_guess, wasHigh);


        }
        if(end_game)
            break;
    }
    return 0;
}
```

```
//
//  Map.h
//  Coen70Lab7
//
//  Created by Yousef Zoumot on 2/16/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <utility>
#include <list>

using namespace std;

template < class K, class V >
class Map{
    list< pair<K,V> >  mList;
public:
    void insert (K key, V value);
    bool contains_key(K key);
    V value_of(K key);
    void remove_key(K key);
    void printVales();
};



template <class K, class V>
bool Map<K,V>:: contains_key(K key){
    typename list< pair<K, V> >:: iterator it;
    for(it=mList.begin(); it!= mList.end(); it++){
        if(it->first==key)
            return true;
    }
    return false;
}

template <class K, class V>
void Map<K,V>:: insert(K key, V value){
    if(contains_key(key))
        return;
    mList.push_back(pair<K,V> (key, value));
    return;
}

template <class K, class V>
V Map<K,V>:: value_of(K key){
    typename list< pair<K, V> >:: iterator it;
    for(it=mList.begin(); it!= mList.end(); it++){
        if(it->first==key)
            return it->second;
```

```
    }
    //cout<<"Does not compute... Please put a valid key in...\n";
    return NULL;
}

template < class K, class V>
void Map<K,V>:: remove_key(K key){
    /*  this works too but it is messier
     typename list< pair<K, V> >:: iterator it;
    for(it=mList.begin(); it!= mList.end(); it++){
        if(it->first==key)
            mList.remove(pair<K,V>(key, it->second));
    }
    return;*/
    if(contains_key(key))
        mList.remove( pair<K,V>(key,value_of(key)));

}

template<class K, class V>
void Map<K,V>:: printVales(){
    typename list< pair<K, V> >:: iterator it;
    for(it=mList.begin(); it!= mList.end(); it++){
        cout<<"Key: "<<it->first;
        cout<<"        Value: "<<it->second<<"\n";
    }
    cout<<"\n";
    return;
}
```

```
/*
#ifndef Coen70Lab7_Map_h
#define Coen70Lab7_Map_h


#endif
*/
```

```
//  Yousef Zoumot   Kelly Wesley
//  main.cpp
//  Coen70Lab8
//
//  Created by Yousef Zoumot on 2/23/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
#include <string>//
#include <sstream>//
#include <stack>//


using namespace std;


class ExpTree{
    struct Node{
        int value;
        char op;
        Node *left, *right;
    };
    Node *root;
    void print_inorder(Node* node);
    void print_preorder(Node* node);
    void print_postorder(Node* node);
    int evaluateR(Node * node);
    void destroyT(Node* node);
public:
    ExpTree();
    ~ExpTree();
    void build(string expr);
    int evaluate();
    void printInOrder();
    void printPreOrder();
    void printPostOrder();

};

ExpTree:: ExpTree(){
    root=new Node;
    root->left=NULL;
    root->right=NULL;
}
ExpTree:: ~ExpTree(){
    destroyT(root);
}

void ExpTree:: destroyT(Node* node){
```

```
    if(node==NULL)
        return;
    destroyT(node->left);
    destroyT(node->right);
    delete node;
}



void ExpTree:: build(string expr){
    istringstream stream(expr);
    string token;
    stack<Node*> nodes;
    while(stream>>token){
        if(token=="+" || token=="-" || token=="/" || token=="*"){
            Node* second= nodes.top();
            nodes.pop();
            Node* first=nodes.top();
            nodes.pop();
            Node* oper= new Node;
            oper->op=token[0];
            oper->left=first;
            oper->right=second;
            nodes.push(oper);
        }
        else{
            int temp=atoi(token.c_str());
            Node* tmp=new Node;
            tmp->value=temp;
            tmp->left=NULL;
            tmp->right=NULL;
            nodes.push(tmp);
        }
    }
    root=nodes.top();
}

int ExpTree:: evaluateR(Node* node){
    if(node->left==NULL && node->right==NULL){
        return node->value;
    }
    if(node->op== '+'){
        return evaluateR(node->left) + evaluateR(node->right);
    }
    if(node->op== '-'){
        return evaluateR(node->left) - evaluateR(node->right);
    }
    if(node->op== '/'){
        return evaluateR(node->left) / evaluateR(node->right);
    }
```

```
    if(node->op== '*'){
        return evaluateR(node->left) * evaluateR(node->right);
    }
    return 0;
}

int ExpTree:: evaluate(){
    return evaluateR(root);

}

void ExpTree:: printInOrder(){
    print_inorder(root);
}

void ExpTree::print_inorder(Node* node){
    if(node==NULL)
        return;
    print_inorder(node->left);
    if(node->op=='+' || node->op=='-' || node->op=='/' || node-
>op=='*')
        cout<< node->op<< " ";
    else
        cout<<node->value<< " ";
    print_inorder(node->right);
}

void ExpTree:: printPreOrder(){
    print_preorder(root);
}

void ExpTree::print_preorder(Node* node){
    if(node==NULL)
        return;
    if(node->op=='+' || node->op=='-' || node->op=='/' || node-
>op=='*')
        cout<< node->op<< " ";
    else
        cout<<node->value<< " ";
    print_preorder(node->left);
    print_preorder(node->right);
}

void ExpTree:: printPostOrder(){
    print_postorder(root);
}

void ExpTree::print_postorder(Node* node){
    if(node==NULL)
        return;
```

```
    print_postorder(node->left);
    print_postorder(node->right);
    if(node->op=='+' || node->op=='-' || node->op=='/' || node-
>op=='*')
        cout<< node->op<< " ";
    else
        cout<<node->value<< " ";
}

int main(int argc, const char * argv[]) {
    // insert code here...
    ExpTree t1;
    t1.build("5 3 + 2 5 * 9 3 / - +");
    cout<<t1.evaluate();
    cout<<"\n";
    t1.printInOrder();
    cout<<"\n";
    t1.printPostOrder();
    cout<<"\n";
    t1.printPreOrder();
    cout<<"\n";
    return 0;
}
```

```
// Gabrielle Tordillos Yousef Zoumot
//  main.cpp
//  Coen70Lab9
//
//  Created by Yousef Zoumot on 3/1/16.
//  Copyright (c) 2016 Yousef Zoumot. All rights reserved.
//

#include <iostream>
#include <vector>

using namespace std;

template<typename T>
class BinomialHeap{
public:
    BinomialHeap();
    ~BinomialHeap();

    BinomialHeap& operator=(const BinomialHeap&);

    void push(T value);


    T top() const;
    void pop();

    void merge(BinomialHeap &other);

    int count(){return _count;};
private:
    struct BinTree{
        T value;
        BinTree *child, *sibling;
        BinTree(T value, BinTree *child, BinTree *sibling);
        BinTree(BinTree &other);
        static void deletion(BinTree* rec);
    };
public:
    void printV(BinTree* rec);

    void printValues();
private:
    std::vector<BinTree*> trees;
    int _count;



    BinTree* mergeTrees(BinTree *lhs, BinTree *rhs){
        if(lhs == NULL)
```

```
            return rhs;
        if(rhs == NULL)
            return lhs;

        if(rhs->value < lhs->value)
            std::swap(lhs, rhs);

        rhs->sibling = lhs->child;
        lhs->child = rhs;
        return lhs;
    }

    void mergeHeaps(std::vector<BinTree*> &lhs,
std::vector<BinTree*> &rhs){
        std::vector<BinTree *> result;
        BinTree *carry = NULL;
        size_t max_order = std::max(lhs.size(), rhs.size());
        lhs.resize(max_order);
        rhs.resize(max_order);

        for(int order = 0; order < max_order; order++){

            std::vector<BinTree *> tmp;
            if(carry) tmp.push_back(carry);
            if(lhs[order]) tmp.push_back(lhs[order]);
            if(rhs[order]) tmp.push_back(rhs[order]);


            if(tmp.empty()){
                //Case 0: All trees are NULL (0 + 0 = 0)
                result.push_back(NULL);
            }else if(tmp.size() == 1){
                //Case 1: One tree is not NULL (1 + 0 = 0)
                result.push_back(tmp[0]);
                carry=NULL;
            }else if(tmp.size() == 2){
                //Case 2: Two trees are not NULL (1 + 1 = 0 carry
1)
                //Case 3: 3 trees are not NULL
                carry=mergeTrees(tmp[0], tmp[1] );
                result.push_back(NULL);
            }else if(tmp.size() == 3){
                //Case 4: Three trees are not NULL (1 + 1 with
carry = 1 carry 1)
                carry=mergeTrees(tmp[0], tmp[1] );
                result.push_back(tmp[2]);
            }
        }
        if(carry)
            result.push_back(carry);
```

```
        rhs.clear();
        trees = result;
    }
};

template< class T >
BinomialHeap<T>::BinomialHeap(){
    _count=0;
}

template< class T >
BinomialHeap<T>:: ~BinomialHeap(){
    for(int i=0; i<trees.size(); i++){
        BinTree::deletion(trees[i]);
    }
    while(trees.size()!=0){
        trees.pop_back();
    }

}

template< class T >
void BinomialHeap< T >:: push(T value){
    vector< BinTree* > tmp;
    BinTree* temp= new BinTree(value, NULL, NULL);
    tmp.push_back(temp);
    mergeHeaps(trees, tmp);
}

template < class T >
T BinomialHeap<T>:: top() const{
    T result=NULL;
    for(int i=0; i<trees.size() ; i++){
        if(trees[i]!=NULL && (result==NULL || trees[i]<result))
            result=trees[i];
    }
    return result;

}

template< class T >
void BinomialHeap<T>:: pop(){
    BinTree* result=NULL;
    int order=0;
    for(int i=0; i<trees.size() ; i++){
        if(trees[i]!=NULL && (result==NULL || trees[i]<result)){
            result=trees[i];
            order=i;
        }
```

```
    }
    vector<BinTree* > temp;
    temp.resize(order);
    BinTree* p=result->child;
    while(p !=NULL){
        temp.push_back(p);
        p=p->sibling;
    }
    for(int h=0; h<temp.size(); h++){
        if(temp[h]!=NULL)
            temp[h]->sibling=NULL;

    }
    trees[order]=NULL;
    mergeHeaps(trees, temp);
}
template<class T >
void BinomialHeap<T>:: merge(BinomialHeap &other){
    mergeHeaps(&trees, &other.trees);
    _count= _count + other.count();
}

template< class T >
BinomialHeap<T>:: BinTree:: BinTree(BinTree &other){
    value=other.value;
    if(other.child==NULL)
        child=NULL;
    else{
        child=new BinTree(other.child);
    }
    if(other.sibling==NULL){
        sibling=NULL;
    }
    else{
        sibling=new BinTree(other.sibling);
    }


}

template< class T >
BinomialHeap<T>& BinomialHeap< T >:: operator=(const
BinomialHeap<T>& other){
    vector<BinTree* > temp;
    for(int i=0; i<other.trees.size(); i++){
        temp.push_back(BinTree(other.trees[i]));
    }
    trees=temp;
    _count=other.count();
    return *this;
```

```
}

template< class T>
void BinomialHeap< T >:: BinTree:: deletion( BinTree* rec){
    if(rec==NULL)
        return;
    deletion(rec->sibling);
    deletion(rec->child);
    delete rec;
}

template<class T>
BinomialHeap<T>:: BinTree:: BinTree(T data, BinTree* c, BinTree*
s){
    value=data;
    child =c;
    sibling = s;
}

template<class T>
void BinomialHeap<T>:: printV(BinTree* rec){
    if(rec==NULL)
        return;
    printV(rec->child);
    cout<< rec->value << "\n";
    printV(rec->sibling);
}

template<class T>
void BinomialHeap<T>:: printValues(){
    for(int i=0; i<trees.size(); i++){
        printV(trees[i]);
    }
    cout<<"\n";
}

int main(int argc, const char * argv[]) {
    // insert code here...
    BinomialHeap<int> bh1, bh2;
    bh1.push(1);
    bh1.push(2);
    bh1.push(3);
    bh1.push(4);
    bh1.push(5);
    bh1.push(6);
    bh1.push(7);
    bh1.push(8);
    bh1.printValues();
    bh1.pop();
```

```
    bh1.printValues();
    bh1.pop();
    bh1.printValues();
    bh1.pop();
    bh1.printValues();



    return 0;
}
```