Universidad Politecnica de Puerto Rico

Proyecto Final

Jorge A. Serrano

#121260

CECS – 2222 – Computer Programming II

Prof. Claudia Talavera

MyString

Tabla descriptiva atributos private:

|  |  |
| --- | --- |
| str | Variable data type char tipo puntero |
| len | Variable datatype int. Guarda la longitud de str |

Tabla descriptiva atributos public:

|  |  |
| --- | --- |
| MyString() | Default constructor |
| MyString(char\*) | Constructor con parametros |
| MyString(const MyString& right) | Compy constructor |
| ~MyString() | Destructor |
| int length() const; | Guarda la longitud de str atraves del uso de len |
| char\* data() const; | Guarda data puntero de str |
| operator=(const MyString&); | Operator Relacional = |
| operator==(MyString, MyString) | Operator Relacional == |
| operator!=(MyString, MyString); | Operador relacional != |
| operator<<(ostream&, MyString); | Operador stream << |
| operator>>(istream&, MyString&); | Operador streeam >> |

SearchableVector

Tabla descriptiva atributo public:

|  |  |
| --- | --- |
| SearchableVector() | Default constructor |
| SearchableVector(int size) | Constructor con parametros |
| SearchableVector(const SearchableVector&) | Copy constructor |
| int findItem(const T) | Accesor para encontrar “item” |

SimpleVector

Tabla descriptiva atributo protected:

|  |  |
| --- | --- |
| aptr | Variable data type template T tipo puntero. Apunta a la memoria guardad. |
| memError() | Maneja errores de alocacion |
| subError() | Maneja los errores |
| arraySize | Elementos adrentro del arreglo |
| itemCount | Cantidadde elementos adentro del arreglo |

Tabla descriptiva atributo public:

|  |  |
| --- | --- |
| SimpleVector() | Default constructor |
| SimpleVector(int) | Constructor parametrisado |
| SimpleVector(const SimpleVector&) | Copy constructor |
| ~SimpleVector() | Destructor |
| getArraySize() const | Retorma arraySize |
| getItemCount() const | Retorna itemCount |
| getElementAt(int position) | Esta funcion reggresa un valor del puntero |
| operator[](const int&) | Overload operator [] |
| add(const T& newEntry) | Funcion de adicion |
| remove(const T& anEntry) | Funcion para remover un aptr |
| getIndexOf(const T& targetEntry) const | Funcion de index |
| isEmpty() const | Retorna falso si el puntero esta vacio |
| display()const | Imprime el asrreglo dynamico |

SimpleVectorInterface

|  |  |
| --- | --- |
| ~SimpleVectorInterface() | Destructor incialisado |
| add(const T& newEntry) = 0; | Funcion add inicialisado |
| remove(const T& anEntry) = 0; | Funcion remove inicialisado |
| getIndexOf(const T& targetEntry) const = 0; | Funcion getIndex inicialisado |
| isEmpty() const = 0; | Funcion isEmpty inicialisado |
| display()const = 0; | Funcion display inicialisado |

Soccer

Tabla derscriptiva private:

|  |  |
| --- | --- |
| name | Variable tipo MyString |
| number | Variable tipo int |
| points | Vairable tipo int, guarda puntuacion |

Tabla descriptiva public:

|  |  |
| --- | --- |
| Soccer() | Default constructor |
| Soccer(MyString aname, int anumber, int apoints) | Parametrized Constructor |
| Soccer(const Soccer& obj) | Copy Constructor |
| ~Soccer() | Destructor |
| setName(MyString) | Guarda nombre del jugador |
| setNumber(int) | Guarda numero de jugador |
| setPoint(int) | Guarda puntos del jugador |
| getName() const | Retrorna nombre del jugador |
| getNumber() const | Retorna numero del jugador |
| getPoint() const | Retorna puntos del jugador |
| operator=(const Soccer& obj) | Operador relacional = |
| operator<(const Soccer& obj) | Operador relacional < |
| operator>(const Soccer& obj) | Operador relacional > |
| operator<<(ostream& strm, const Soccer& obj) | Operador stream << |
| operator>>(istream& strm, Soccer& obj) | Operador stream >> |

SoccerTeam

Tabla descriptiva private:

|  |  |
| --- | --- |
| team | Variable puntero guarda informacion del equipo |

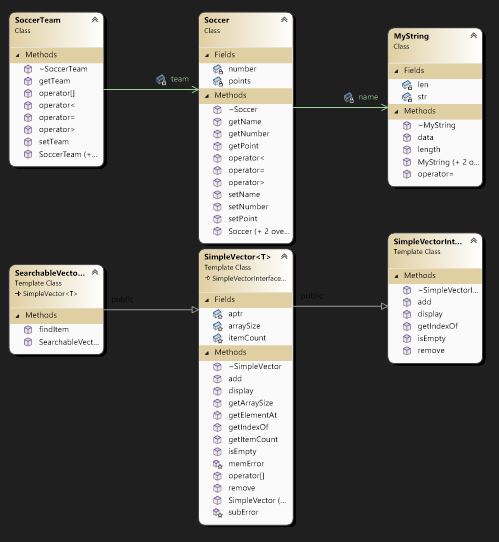
Tabla descriptiva public:

|  |  |
| --- | --- |
| SoccerTeam() | Default constructor |
| SoccerTeam(Soccer\*) | Parametrized Constructor |
| SoccerTeam(const SoccerTeam& obj) | Copy Constructor |
| ~SoccerTeam() | Destructor |
| setTeam(Soccer\*) | Guarda variable team |
| operator>(const SoccerTeam& obj) | Operador relacional > |
| operator<(const SoccerTeam& obj) | Operador relacional < |
| getTeam() const | Retorna team |
| operator [](int) | Overload operator [] |
| operator=(const SoccerTeam& obj) | Operador relacional = |
| operator<<(ostream& strm, SoccerTeam& obj) | Operador stream << |
| operator>>(istream& strm, SoccerTeam& obj) | Operador stream >> |

SortableVector

|  |  |
| --- | --- |
| SortableVector(int so) | Constructor con parametros |
| SortableVector(SimpleVector<T>& obj) | Copy constructor |
| sortAscending() | Sorts in ascending order |

UML



MyString.h

#pragma once

#ifndef MYSTRING\_H

#define MYSTRING\_H

#include <iostream>

#include <string.h>

#include <cstdlib>

using namespace std;

class MyString {

private:

char\* str;

int len;

public:

// Constructors

MyString();

MyString(char\*);

MyString(const MyString& right);

// Destructor

~MyString();

// Various member functions and operators

int length() const;

char\* data() const;

MyString& operator=(const MyString&);

// Various overloaded operators

friend bool operator==(MyString, MyString);

friend bool operator!=(MyString, MyString);

friend ostream& operator<<(ostream&, MyString);

friend istream& operator>>(istream&, MyString&);

};

#endif

MyString.cpp

#include "MyString.h"

#include <cstring>

MyString::MyString()

{

str = NULL;

len = 0;

}

MyString::MyString(char\* sptr)

{

len = strlen(sptr);

str = new char[len + 1];

strcpy\_s(str, len + 1, sptr);

}

MyString::MyString(const MyString& obj)

{

str = new char[obj.len + 1];

strcpy\_s(str, obj.len + 1, obj.str);

len = obj.len;

}

MyString::~MyString()

{

if (len != 0)

delete[] str;

}

// Various member functions and operators

int MyString::length() const

{

return len;

}

char\* MyString::data() const

{

return str;

}

MyString& MyString::operator=(const MyString& obj)

{

char\* temp = str;

str = new char[len + obj.len + 1];

strcpy\_s(str, len + obj.len + 1, obj.str);

//strcat\_s(str, len + obj.len + 1, );

if (len != 0)

{

delete[] temp;

}

len += obj.len;

return \*this;

}

ostream& operator<<(ostream& strm, MyString obj)

{

strm << obj.str;

return strm;

}

istream& operator>>(istream& strm, MyString& obj)

{

strm.ignore();

// Read the MyString

char buffer[256];

strm.getline(buffer, 256);

// Invoke the convert constructor and overloaded assignment

obj = buffer;

return strm;

}

SearchableVector.h

#ifndef SEARCHABLEVECTOR\_H

#define SEARCHABLEVECTOR\_H

#include "SimpleVector.h"

template <class T>

class SearchableVector : public SimpleVector<T>

{

public:

// Default constructor

SearchableVector() : SimpleVector<T>() {}

// Constructor

SearchableVector(int size) : SimpleVector<T>(size) {}

// Copy constructor

SearchableVector(const SearchableVector&);

// Accessor to find an item

int findItem(const T);

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Copy constructor \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

SearchableVector<T>::SearchableVector(const SearchableVector& obj) :

SimpleVector<T>(obj.size())

{

for (int count = 0; count < this->size(); count++)

this->operator[](count) = obj[count];

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// findItem function \*

// This function searches for item. If item is found \*

// the subscript is returned. Otherwise -1 is returned. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

int SearchableVector<T>::findItem(const T item)

{

//initialize min and max

int min = 0, max = this->size() - 1, mid;

//loop until min and max overlap

while (min <= max)

{

//update mid after each iteration

mid = (min + max) / 2;

if (this->getElementAt(mid) == item)

return mid;

else if (this->getElementAt(mid) > item)

//update max

max = mid - 1;

else

//update min

min = mid + 1;

}

return -1;

}

#endif

SimpleVector.h

#pragma once

// check page 997

#ifndef SIMPLEVECTOR\_H

#define SIMPLEVECTOR\_H

#include <iostream>

#include <new> // Needed for bad\_alloc exception

#include <cstdlib> // Needed for the exit function

using namespace std;

#include "SimpleVectorInterface.h"

template <class T>

class SimpleVector :public SimpleVectorInterface<T>

{

protected:

T\* aptr;

void memError();

void subError();

int arraySize;

int itemCount;

public:

// Default constructor

SimpleVector();

// Constructor declaration

SimpleVector(int);

// Copy constructor declaration

SimpleVector(const SimpleVector&);

// Destructor declaration

virtual ~SimpleVector();

// Accessor to return the array size

int getArraySize() const;

// Accessor to return a specific element

int getItemCount() const;

T getElementAt(int position);

// Overloaded [] operator declaration

T& operator[](const int&);

virtual bool add(const T& newEntry);

virtual bool remove(const T& anEntry);

virtual int getIndexOf(const T& targetEntry) const;

virtual bool isEmpty() const;

virtual void display()const;

};

template <class T>

SimpleVector<T>::SimpleVector()

{

aptr = 0;

arraySize = 0;

itemCount = 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Constructor for SimpleVector class. Sets the size of the \*

// array and allocates memory for it. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

SimpleVector<T>::SimpleVector(int s)

{

arraySize = s;

itemCount = 0;

// Allocate memory for the array.

try

{

aptr = new T[s];

}

catch (bad\_alloc)

{

memError();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Copy Constructor for SimpleVector class. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

SimpleVector<T>::SimpleVector(const SimpleVector& obj)

{

// Copy the array size.

arraySize = obj.arraySize;

itemcount = obj.itemCount;

// Allocate memory for the array.

aptr = new T[arraySize];

if (aptr == 0)

memError();

// Copy the elements of obj's array.

for (int count = 0; count < getItemCount(); count++)

\*(aptr + count) = \*(obj.aptr + count);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Destructor for SimpleVector class. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

SimpleVector<T>::~SimpleVector()

{

if (itemCount > 0)

delete[] aptr;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// memError function. Displays an error message and \*

// terminates the program when memory allocation fails. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

void SimpleVector<T>::memError()

{

cout << "ERROR:Cannot allocate memory.\n";

exit(EXIT\_FAILURE);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// subError function. Displays an error message and \*

// terminates the program when a subscript is out of range. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

void SimpleVector<T>::subError()

{

cout << "ERROR: Subscript out of range.\n";

exit(EXIT\_FAILURE);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// getElementAt function. The argument is a subscript. \*

// This function returns the value stored at the \*

// subcript in the array. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

T SimpleVector<T>::getElementAt(int sub)

{

if (sub < 0 || sub > getItemCount())

subError();

return aptr[sub];

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Overloaded [] operator. The argument is a subscript. \*

// This function returns a reference to the element \*

// in the array indexed by the subscript. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <class T>

T& SimpleVector<T>::operator[](const int& sub)

{

if (sub < 0 || sub > getItemCount())

subError();

return aptr[sub];

}

template <class T>

int SimpleVector<T>::getArraySize() const {

return arraySize;

}

template <class T>

int SimpleVector<T>::getItemCount() const {

return itemCount;

}

template <class T>

bool SimpleVector<T>::add(const T& newEntry)

{

bool hasRoomToAdd = (itemCount < getArraySize());

if (hasRoomToAdd) {

aptr[itemCount] = newEntry;

itemCount++;

}//end if

return hasRoomToAdd;

}

template <class T>

int SimpleVector<T>::getIndexOf(const T& targetEntry) const {

bool found = false;

int result = -1;

int searchIndex = 0;

while (!found && (searchIndex < getItemCount())) {

if (aptr[searchIndex] == targetEntry) {

found = true;

result = searchIndex;

} //end if

else {

searchIndex++;

}//end else

}//end whule

return result;

}

template <class T>

bool SimpleVector<T>::isEmpty() const {

bool result;

if (getItemCount() == 0)

result = true;

else result = false;

return result;

}

template <class T>

bool SimpleVector<T>::remove(const T& anEntry) {

int locatedIndex = getIndexOf(anEntry);

bool canRemoveItem = !isEmpty() && (locatedIndex > -1);

if (canRemoveItem) {

itemCount--;

aptr[locatedIndex] = aptr[itemCount];

}//end if

return canRemoveItem;

}

template <class T>

void SimpleVector<T>::display() const {

cout << "El contenido dentro del arreglo es:";

for (int index = 0; index < getItemCount(); index++) {

cout << aptr[index] << endl;

}//end for

}

#endif

SimpleVectorInterface.h

#pragma once

#ifndef \_SIMPLE\_VECTOR\_INTERFACE\_

#define \_SIMPLE\_VECTOR\_INTERFACE\_

template<class T>

class SimpleVectorInterface

{

public:

~SimpleVectorInterface() {}

virtual bool add(const T& newEntry) = 0;

virtual bool remove(const T& anEntry) = 0;

virtual int getIndexOf(const T& targetEntry) const = 0;

virtual bool isEmpty() const = 0;

virtual void display()const = 0;

};

#endif

Soccer.h

#ifndef \_SOCCER\_H

#define \_SOCCER\_H

#include "MyString.h"

#include<iostream>

using namespace std;

class Soccer {

private:

MyString name;

int number;

int points;

public:

Soccer();

Soccer(MyString aname, int anumber, int apoints);

Soccer(const Soccer& obj);

~Soccer();

void setName(MyString);

void setNumber(int);

void setPoint(int);

MyString getName() const;

int getNumber() const;

int getPoint() const;

Soccer& operator=(const Soccer& obj);

bool operator<(const Soccer& obj);

bool operator>(const Soccer& obj);

friend ostream& operator<<(ostream& strm, const Soccer& obj);

friend istream& operator>>(istream& strm, Soccer& obj);

};

#endif

Soccer.cpp

#include "Soccer.h"

Soccer::Soccer()

{

setNumber(0);

setPoint(0);

}

Soccer::Soccer(MyString aname, int anumber, int apoints)

{

setName(aname);

setNumber(anumber);

setPoint(apoints);

}

Soccer::Soccer(const Soccer& obj)

{

setName(obj.name);

setNumber(obj.number);

setPoint(obj.points);

}

Soccer::~Soccer()

{

}

void Soccer::setName(MyString aname)

{

name = aname;

}

void Soccer::setNumber(int anumber)

{

number = anumber;

}

void Soccer::setPoint(int apoints)

{

points = apoints;

}

MyString Soccer::getName() const

{

return name;

}

int Soccer::getNumber() const

{

return number;

}

int Soccer::getPoint() const

{

return points;

}

Soccer& Soccer::operator=(const Soccer& obj)

{

setName(obj.name);

//this->name = name;

setNumber(obj.number);

setPoint(obj.points);

return(\*this);

}

bool Soccer::operator<(const Soccer& obj)

{

if (getPoint() < obj.getPoint())

{

return true;

}

else {

return false;

}

}

bool Soccer::operator>(const Soccer& obj)

{

if (points > obj.points)

{

return true;

}

else {

return false;

}

}

ostream& operator<<(ostream& strm, const Soccer& obj)

{

cout << "The Most Valuable Player:\n" << endl;

strm << "Player name: " << obj.getName() << endl;

strm << "ID: #" << obj.getNumber() << endl;

strm << "Goals: " << obj.getPoint() << endl;

return strm;

}

istream& operator>>(istream& strm, Soccer& obj)

{

MyString aname;

//int anumber, apoints;

cout << "Enter name: ";

strm >> obj.name;

//obj.setName(aname);

cout << "Enter player number: ";

strm >> obj.number;

cout << "Enter total of goals: ";

strm >> obj.points;

return strm;

}

SoccerTeam.h

#ifndef \_SOCCERTEAM\_H

#define \_SOCCERTEAM\_H

#include "Soccer.h"

class SoccerTeam

{

private:

Soccer\* team;

public:

SoccerTeam();

SoccerTeam(Soccer\*);

SoccerTeam(const SoccerTeam& obj);

~SoccerTeam();

void setTeam(Soccer\*);

bool operator>(const SoccerTeam& obj);

bool operator<(const SoccerTeam& obj);

Soccer\* getTeam() const;

Soccer& operator [](int);

SoccerTeam& operator=(const SoccerTeam& obj);

friend ostream& operator<<(ostream& strm, SoccerTeam& obj);

friend istream& operator>>(istream& strm, SoccerTeam& obj);

};

#endif

SoccerTeam.cpp

#include "SoccerTeam.h"

int SIZE = 12;

SoccerTeam::SoccerTeam()

{

team = new Soccer[SIZE];

}

SoccerTeam::SoccerTeam(Soccer\* ateam)

{

setTeam(ateam);

}

SoccerTeam::SoccerTeam(const SoccerTeam& obj)

{

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

{

team[i] = obj.team[i];

}

}

SoccerTeam::~SoccerTeam()

{

delete[] team;

}

void SoccerTeam::setTeam(Soccer\* ateam)

{

team = new Soccer[SIZE];

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

{

team[i] = ateam[i];

}

}

Soccer\* SoccerTeam::getTeam() const

{

return team;

}

Soccer& SoccerTeam::operator[](int index)

{

return team[index];

}

SoccerTeam& SoccerTeam::operator=(const SoccerTeam& obj)

{

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

{

this->team[i] = obj.team[i];

}

return (\*this);

}

bool SoccerTeam::operator>(const SoccerTeam& obj)

{

if (team > obj.team) {

return true;

}

else {

return false;

}

}

bool SoccerTeam::operator<(const SoccerTeam& obj)

{

if (team < obj.team) {

return true;

}

else {

return false;

}

}

ostream& operator<<(ostream& strm, SoccerTeam& obj)

{

strm << obj.getTeam();

return strm;

}

istream& operator>>(istream& strm, SoccerTeam& obj)

{

for (int count = 0; count < SIZE; count++)

{

cout << "Player" << count + 1 << ": \n";

strm >> obj.team[count];

cout << endl;

}

return strm;

}

SortableVector.h

#pragma once

#ifndef SORTABLEVECTOR\_H

#define SORTABLEVECTOR\_H

#include <iostream>

#include "SimpleVector.h"

#include <algorithm>

template <class T>

class SortableVector : public SimpleVector<T>

{

public:

SortableVector(int so) : SimpleVector<T>(so)

{

}

SortableVector(SimpleVector<T>& obj) : SimpleVector<T>(obj)

{

}

void sortAscending();

};

template <class T>

void SortableVector<T>::sortAscending()

{

bool Swap;

do

{

Swap = false;

for (int i = 0; i < (this->size() - 1); i++)

{

if (this->operator[](i) > this->operator[](i + 1))

{

swap(this->operator[](i), this->operator[](i + 1));

Swap = true;

}

}

} while (Swap);

}

#endif

Main.cpp

#include <iostream>

#include "SoccerTeam.h"

#include "SimpleVector.h"

#include "SearchableVector.h"

#include "SortableVector.h"

using namespace std;

void selectionSort(int array[], int size);

int main()

{

int MAX\_PLAYER;

SoccerTeam team;

int total\_goals = 0;

SearchableVector<Soccer> searchV(MAX\_PLAYER);

SimpleVector<Soccer> simpleV(MAX\_PLAYER);

SortableVector<Soccer> sortV2(MAX\_PLAYER);

cout << "Enter player amount: ";

cin >> MAX\_PLAYER;

for (int i = 0; i < MAX\_PLAYER; i++)

{

cin >> team;

for (int k = 0; k < MAX\_PLAYER; k++)

{

total\_goals += team[k].getPoint();

}

}

for (int i = 0; i < 1; i++)

{

for (int count = 1; count < MAX\_PLAYER; count++)

{

if (team[i] < team[count]) {

team[i] = team[count];

}

}

}

cout << endl << team[0] << "Team Total Goals: " << total\_goals;

return 0;

}

void selectionSort(int array[], int size)

{

int startScan, minIndex, minValue;

for (startScan = 0; startScan < (size - 1); startScan++)

{

minIndex = startScan;

minValue = array[startScan];

for (int index = startScan + 1; index < size; index++)

{

if (array[index] < minValue)

{

minValue = array[index];

minIndex = index;

}

}

array[minIndex] = array[startScan];

array[startScan] = minValue;

}

}