Samin Amanat

Pentcheva

COMSC-210

September 18, 2019

#include <iostream>

using namespace std;

template <class T> //Template class that can make dynamic array of any type

class DynamicArray

{

private:

T \* dynamicArray; //Initializing the array of T-Type values

int SIZE; //Setting up the size of the array

public:

//Prototypes

DynamicArray(); //Constructor

DynamicArray(const DynamicArray<T>&); //Copy Constructor

int getSize() const; //Get size of the array

void addEntry(const T); //Add entry to array

bool deleteEntry(T); //Delete entry from array

T getEntry(int) const; //Find entry in array at an index

~DynamicArray(); //Destructor

};

template <class T>

DynamicArray<T>::DynamicArray() //Constructor

{

SIZE = 0; //Initialize size to zero

dynamicArray = new T[SIZE]; //Make the array with zero values so that an array "exists"

}

template <class T> //Copy constructor

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

{

SIZE = obj.getSize();

dynamicArray = new T[SIZE];

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

{

dynamicArray[i] = obj.getEntry(i); //To avoid pointer issues, each value in the array is coppied over one by one by value.

}

}

template <class T>

int DynamicArray<T>::getSize() const //Return size of the array

{

return SIZE;

}

template <class T>

void DynamicArray<T>::addEntry(T const s)

{

T \* dynamicArray2; //Make another "dynamicArray"

dynamicArray2 = new T[SIZE + 1]; //New array is 1 value larger

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

{

dynamicArray2[i] = dynamicArray[i]; //Copy over the values

}

dynamicArray2[SIZE] = s; //Add the new value

SIZE++; //Increment the size of the array held by the class

delete[] dynamicArray; //"Delete" original array

dynamicArray = dynamicArray2; //Set original array to the new array

}

template <class T>

bool DynamicArray<T>::deleteEntry(T s) //Delete an entry

{

bool flag = false; //Entry not found

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

{

if (dynamicArray[i] == s) flag = true; //Entry found

}

if (flag)

{

//Same story as the adding value but deleting and incrementing the size downwards

T \* dynamicArray2;

dynamicArray2 = new T[SIZE - 1];

int x = 0;

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

{

if (dynamicArray[i] != s)

{

dynamicArray2[x] = dynamicArray[i];

}

else if (dynamicArray[i] == s)

{

x--;

}

x++;

}

SIZE--;

delete[] dynamicArray;

dynamicArray = dynamicArray2;

return true;

}

else return false;

}

template <class T>

T DynamicArray<T>::getEntry(int x) const //has to be const so compiler

//doesn't think that this function will effect the original object

//being coppied from

{

if (x > SIZE || x < 0)

{

cout << "Index out of bounds." << endl;

return 0;

}

else

{

return dynamicArray[x];

}

}

template <class T>

DynamicArray<T>::~DynamicArray() //Basic destructor

{

delete[] dynamicArray; //array is dead (-:

}

//Provided main function

int main()

{

DynamicArray<string> names;

// List of names

names.addEntry("Frank");

names.addEntry("Wiggum");

names.addEntry("Nahasapeemapetilon");

names.addEntry("Quimby");

names.addEntry("Flanders");

// Output list

cout << "List of names:" << endl;

for (int i = 0; i < names.getSize(); i++)

cout << names.getEntry(i) << endl;

cout << endl;

// Add and remove some names

names.addEntry("Spuckler");

cout << "After adding a name:" << endl;

for (int i = 0; i < names.getSize(); i++)

cout << names.getEntry(i) << endl;

cout << endl;

names.deleteEntry("Nahasapeemapetilon");

cout << "After removing a name:" << endl;

for (int i = 0; i < names.getSize(); i++)

cout << names.getEntry(i) << endl;

cout << endl;

names.deleteEntry("Skinner");

cout << "After removing a name that isn't on the list:" << endl;

for (int i = 0; i < names.getSize(); i++)

cout << names.getEntry(i) << endl;

cout << endl;

names.addEntry("Muntz");

cout << "After adding another name:" << endl;

for (int i = 0; i < names.getSize(); i++)

cout << names.getEntry(i) << endl;

cout << endl;

// Remove all of the names by repeatedly deleting the last one

while (names.getSize() > 0) {

names.deleteEntry(names.getEntry(names.getSize() - 1));

}

cout << "After removing all of the names:" << endl;

for (int i = 0; i < names.getSize(); i++)

cout << names.getEntry(i) << endl;

cout << endl;

names.addEntry("Olivia");

cout << "After adding a name:" << endl;

for (int i = 0; i < names.getSize(); i++)

cout << names.getEntry(i) << endl;

cout << endl;

cout << "Testing copy constructor" << endl;

DynamicArray<string> names2(names);

// Remove Olivia from names

names.deleteEntry("Olivia");

cout << "Copied names:" << endl;

for (int i = 0; i < names2.getSize(); i++)

cout << names2.getEntry(i) << endl;

cout << endl;

cout << "Testing assignment" << endl;

DynamicArray<string> names3 = names2;

// Remove Olivia from names2

names2.deleteEntry("Olivia");

cout << "Copied names:" << endl;

for (int i = 0; i < names3.getSize(); i++)

cout << names3.getEntry(i) << endl;

cout << endl;

cout << "Testing dynamic array of ints" << endl;

DynamicArray<int> nums;

nums.addEntry(10);

nums.addEntry(20);

nums.addEntry(30);

for (int i = 0; i < nums.getSize(); i++)

cout << nums.getEntry(i) << endl;

cout << endl;

cout << "Enter a character to exit." << endl;

char wait;

cin >> wait;

return 0;

}

