Screenshots:

Graphical user interface, text

Description automatically generated

**Code:**

# Source.cpp

/\*

Edward Alvarado

Class: CIS247C

Date: 4/13/2021

Week 7 Lab- Templated ArrayList

\*/

#include <iostream>

#include <string>

#include <conio.h>

#include "ArrayList.h"

using namespace std;

/// Entry point to the applicatrion

int main(void)

{

// check for memory leaks

#if defined(DEBUG) | defined(\_DEBUG)

\_CrtSetDbgFlag(\_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF);

#endif

// create an int ArrayList

ArrayList<int> intList;

intList.add(45);

intList.add(44);

intList.add(54);

intList.add(100);

intList.add(5);

intList.add(41);

for (int i = 0; i < intList.getCount(); i++)

{

cout << intList.get(i) << ", ";

}

cout << "\n" << endl;

cout << "Count: " << intList.getCount() << endl;

cout << "Capacity: " << intList.getCapacity() << endl;

cout << "\n\n" << endl;

// create a string ArrayList

ArrayList<string> stringList;

stringList.add("Bob");

stringList.add("James");

stringList.add("Ed");

stringList.add("Skeleton");

for (int i = 0; i < stringList.getCount(); i++)

{

cout << stringList.get(i) << ", ";

}

cout << "\n" << endl;

cout << "Count: " << stringList.getCount() << endl;

cout << "Capacity: " << stringList.getCapacity() << endl;

cout << "\n\n" << endl;

// Pause

cout << "\nPress any key to continue...";

\_getch();

return 0;

}

# ArrayList.h

#pragma once

template <class T>

class ArrayList

{

private:

//attributes

const static int DEFAULT\_SIZE = 5;

T\* list; // pointer to the array

int count; // number of items in the list

int capacity; // current size in memory

public:

//constructors

ArrayList(void)

{

this->list = new T[DEFAULT\_SIZE];

this->capacity = DEFAULT\_SIZE;

this->count = 0;

}

ArrayList(int initialCapacity)

{

this->list = new T[initialCapacity];

this->capacity = initialCapacity;

this->count = 0;

}

//destructor

~ArrayList(void)

{

//delete the array pointer

if (this->list != nullptr)

{

//delete[] this->list;

this->list = nullptr;

}

}

/// determine if the arraylist is empty

bool isEmpty(void)

{

return count == 0; // array is empty if it has zero items

}

/// get the item at the given position

T get(int position)

{

if (position < count)

return list[position];

else

return NULL;

}

// add an item to the ArrayList

void add(T data)

{

//if the array is full, double the size

if (count == capacity)

{

//create bigger array

capacity = 2 \* capacity;

T\* temp = new T[capacity];

// copy items from current array to bigger array

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

{

temp[i] = list[i];

}

// delete the current array

delete[] list;

//rename the bigger array to the current array name

list = temp;

}

// add the data item to the array

list[count] = data;

//increment the count

count++;

}

void removeAt(int position)

{

//replace every item from that position on with the next item

for (int i = position; i < count - 1; i++) // notice "count-1" to copy last item to next to last position

{

list[i] = list[i + 1];

}

// decrease the item count

count--;

}

// get the count of items in the ArrayList

int getCount(void)

{

return count;

}

// get the ArrayList current capacity

int getCapacity()

{

return capacity;

}

};