DYNAMIC MEMORY

Workshop 2 (10 marks – 3.75% of your final grade)

In this workshop, you are to allocate memory at run-time and deallocate that memory as soon as it is no longer required.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to:

- allocate and deallocate dynamic memory for an array of elements;
- resize the amount of dynamically allocated memory;
- overload a global function;
- explain the difference between statically and dynamically allocated memory;
- describe what you have learned in completing this workshop.

SUBMISSION POLICY

The *in-lab* section is to be completed during your assigned lab section. It is to be completed and submitted by the end of the workshop period.

If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the *in-lab* portion after the period. You must be present at the lab in order to get credit for the *in-lab* portion.

If you do not attend the lab, you can submit the *in-lab* section along with your *at-home* section (see penalties below). The *at-home* portion of the lab is due on the day that is four days after your scheduled *in-lab* workshop (@23:59) (even if that day is a holiday).

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to back up your work regularly.

LATE SUBMISSION PENALTIES:

- *In-lab* portion submitted late, with *at-home* portion: **0** for *in-lab*. Maximum of **7**/10 for the entire workshop.
- If any of *in-lab*, *at-home* or *reflection* portions is missing, the mark for the workshop will be **0**/10.

IN-LAB (30%)

Design and code a structure named `CellPhone` in the namespace `sict`. The structure should have two data members:

- `m_model`: a statically allocated array of characters of size 32 (including the null byte `'\0'`) that holds a C-style string containing the description of the cellphone model;
- `m_price`: a floating point number in double precision that stores the price of the cellphone.

Add to the `sict` namespace a global function called `display(...)` that returns nothing, receives as a parameter an unmodifiable reference to an object of type `CellPhone` and prints the object to the screen in the following format:

```
Phone MODEL costs $PRICE!<ENDL>
```

Put the structure's *definition* and the `display(...)` *declaration* in a header file named `CellPhone.h`. Put the *definition* of `display(...)` in an implementation file named `CellPhone.cpp`. Replace the ellipsis (...) with the proper parameter.

Complete the implementation of the `w2_in_lab.cpp` main module shown below (see the parts marked with **TODO**). You may use the `read(...)` function provided to accept input from the user. You do not need to write your own.

Below the source code is the expected output from your program. The **red color** identifies what you yourself should type as input to your program. The **green color** identifies what is generated by your program. The output of your program should match **exactly** the sample output shown below.

IN-LAB MAIN MODULE

```
// TODO: include headers

using namespace std;
using namespace sict;

void read(sict::CellPhone& phone);

int main()
{
	int count = 0;
	// TODO: declare the pPhones pointer here (don't forget to initialize it)
	cout << "==========" << endl
	     << "Input data" << endl
	     << "==========" << endl
	     << "Enter the number of phones: ";
	cin >> count;
	cin.ignore();

	if (count < 1) return 1;

	// TODO: allocate dynamic memory here for the pPhones pointer


	for (int i = 0; i < count; ++i) {
		cout << "Phone #" << i + 1 << ": " << endl;
		// TODO: add code to accept user input for Phone i

	}
	cout << "==========" << endl << endl;

	// testing that "display(...)" works
	cout << "------------------------------" << endl
	     << "The 1st phone entered is" << endl
	     << "------------------------------" << endl;
	sict::display(pPhones[0]);
	cout << "------------------------------" << endl << endl;

	// TODO: deallocate the dynamic memory here

	return 0;
}

// read accepts data for a Phone from standard input
void read(sict::CellPhone& phone)
{
	cout << "Enter the model of the Phone: ";
	cin.get(phone.m_model, 32, '\n');
	cin.ignore(2000, '\n');
	cout << "Enter the price for phone " << phone.m_model << ": ";
	cin >> phone.m_price;
	cin.ignore(2000, '\n');
}
```

IN-LAB EXPECTED OUTPUT

```
==========
Input data
==========
Enter the number of phones: 2
Phone #1:
Enter the model of the Phone: Pixel2
Enter the price for phone Pixel2: 759
Phone #2:
Enter the model of the Phone: iPhoneX
Enter the price for phone iPhoneX: 1284.99
==========
------------------------------
The 1st phone entered is
------------------------------
Phone Pixel2 costs $759!
------------------------------
```

In-Lab Submission

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix, upload `CellPhone.h`, `CellPhone.cpp` and `w2_in_lab.cpp` to your matrix account. Compile and run your code and make sure everything works properly.

Then, run the following script from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace `XXX`, i.e., SAA, SBB, etc.):

```
~profname.proflastname/submit 244XXX_w2_lab<ENTER>
```

and follow the instructions.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT-HOME (30%)

Copy your header and implementation files (`CellPhone.h` and `CellPhone.cpp`) from your `in-lab` directory to your `at-home` directory. Upgrade the files in your `at-home` directory as follows.

Overload `sict::display(...)` by adding a function of the same name that returns nothing and has two parameters: the first parameter receives the address of an unmodifiable array of `CellPhone` objects, and the second one receives an integer holding the number of elements in the array. This function finds the cheapest cellphone in the array and prints the array information to the screen in the following format:

```
------------------------------<ENDL>
Phones available at the mall:<ENDL>
------------------------------<ENDL>
1. Phone MODEL costs $PRICE!<ENDL>
2. Phone MODEL costs $PRICE!<ENDL>
3. Phone MODEL costs $PRICE!<ENDL>
------------------------------<ENDL>
The cheapest phone costs $SMALLEST_PRICE.<ENDL>
------------------------------<ENDL>
```

NOTE: This overload must be part of the `sict` namespace, have a declaration in `CellPhone.h` and an implementation in `CellPhone.cpp`.

NOTE: This overload must call the `sict::display(...)` function you created for the *in-lab* part in order to display the model and the price of the cellphone.

Complete the `w2_at_home.cpp` implementation file of the main module shown below (see the parts marked with **TODO**; reuse the code that you have completed for the *in-lab* part).

Your tasks include expanding the amount of dynamic memory allocated for the array of cellphones by one element after reading the original input, and then reading the data for the new element as shown below.

Below the source code is the expected output from your program. The **red color** identifies what you yourself should type as input to your program. The **green color** identifies what is generated by your program. The output of your program should match **exactly** the sample output shown below.

AT-HOME MAIN MODULE

```
// TODO: include headers

using namespace std;
using namespace sict;

void read(sict::CellPhone& phone);

int main()
{
	int count = 0;
	// TODO: declare the pPhones pointer here (don't forget to initialize it)


	cout << "==========" << endl
	     << "Input data" << endl
	     << "==========" << endl
	     << "Enter the number of phones: ";
	cin >> count;
	cin.ignore();

	if (count < 1) return 1;

	// TODO: allocate dynamic memory here for the pPhones pointer


	for (int i = 0; i < count; ++i) {
		cout << "Phone #" << i + 1 << ": " << endl;
		// TODO: add code to accept user input for Phone i
	}
	cout << "==========" << endl << endl;

	// testing that "display(...)" works
	cout << "------------------------------" << endl
	     << "The 1st phone entered is" << endl
	     << "------------------------------" << endl;
	sict::display(pPhones[0]);
	cout << "------------------------------" << endl << endl;

	// expand the array of Phones by 1 element
	{
	    // TODO: allocate dynamic memory for count + 1 Phones
	    // TODO: copy elements from original array into this newly allocated array
	    // TODO: deallocate the dynamic memory for the original array
	    // TODO: copy the address of the newly allocated array into pPhones pointer
	}

	// add the new Phone
	cout << "==========\n"
	     << "Input data\n"
	     << "==========\n"
	     << "Phone #" << count + 1 << ": " << endl;
	// TODO: accept input for the new element in the array
```

```
       count++;
       cout << "==========\n" << endl;

       // testing that the overload of "display(...)" works
       sict::display(pPhones, count);
       cout << endl;

       // TODO: deallocate the dynamic memory here


       return 0;
}


// read accepts data for a Phone from standard input
void read(CellPhone& phone)
{
       cout << "Enter the model of the Phone: ";
       cin.get(phone.m_model, 32, '\n');
       cin.ignore(2000, '\n');
       cout << "Enter the price for phone " << phone.m_model << ": ";
       cin >> phone.m_price;
       cin.ignore(2000, '\n');
}
```

AT-HOME EXPECTED OUTPUT

```
==========
Input data
==========
Enter the number of phones: 2
Phone #1:
Enter the model of the Phone: Pixel2
Enter the price for phone Pixel2: 759
Phone #2:
Enter the model of the Phone: iPhoneX
Enter the price for phone iPhoneX: 1284.99
==========

------------------------------
The 1st phone entered is
------------------------------
Phone Pixel2 costs $759!
------------------------------

==========
Input data
==========
Phone #3:
Enter the model of the Phone: GalaxyS9
Enter the price for phone GalaxyS9: 719.99
==========
------------------------------
Phones available at the mall:
------------------------------
1. Phone Pixel2 costs $759!
2. Phone iPhoneX costs $1284.99!
3. Phone GalaxyS9 costs $719.99!
------------------------------
The cheapest phone costs $719.99.
------------------------------
```

REFLECTION

Study your final solution, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time.**

Create a file named `reflect.txt` that contains your detailed description of the topic that you have learned in completing this workshop and mention any issues that caused you difficulty. Include in your explanation—but do not limit it to—the following points:

1) Why do you need to allocate new dynamic memory for the entire array when you increase the size of an existing array of dynamically allocated memory? Why not allocate memory only for the new element?
2) The `CellPhone` structure stores the model of the cellphone in an array of characters. At the end of the program, we do not use the `delete` operator to deallocate the memory occupied by the model. Why don't we need to use the `delete` operator on this array itself? Explain and compare with the array of cellphones.
3) There are two `display(...)` function definitions. How does the compiler know which definition to call from your main function?

QUIZ REFLECTION

Add a section to `reflect.txt` called "**Quiz X Reflection:**" (replace the '**X**' with the number of the last quiz).

Identify all of the questions from the quiz that you did not answer correctly. Under each question, provide the correct answer. If you missed the last quiz, enter all of the questions and the correct answer for each question.

AT-HOME SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload `reflect.txt`, `CellPhone.h`, `CellPhone.cpp` and `w2_at_home.cpp` to your matrix account. Compile and run your code and make sure everything works properly.

Then, run the following script from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace XXX, i.e., SAA, SBB, etc.):

```
~profname.proflastname/submit 244XXX_w2_home<ENTER>
```

and follow the instructions.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.