Part B – Foundations

Dynamic Memory

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

In this workshop, you 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 (struct) named `Kingdom` in the namespace `sict`. Your structure should have two data members:

`m_name`: a statically allocated array of characters of size 32 (including `'\0'`) that holds the name of the kingdom;
`m_population`: an integer that stores the number of people living in the kingdom.

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

```
   KINGDOM_NAME, population POPULATION<ENDL>
```

Store your struct **definition** and the `display(...)` **declaration** in a header file named `Kingdom.h`. Store your `display(...)` **definition** in an implementation file named `Kingdom.cpp`. Replace the ellipsis `(...)` with the proper parameter as appropriate.

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 `read(...)` function. The expected output from your program is listed below the source code listing. The **scarlet 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.

```
// Workshop 2: Dynamic Memory
// File: w2_in_lab.cpp
// Author: Cornel
// Date: 2019/05/06

#include <iostream>
#include "Kingdom.h"
#include "Kingdom.h"// intentional

using namespace std;
using namespace sict;

void read(sict::Kingdom&);

int main() {
```

```
    int count = 0; // the number of kingdoms in the array

    // TODO: declare the pKingdom pointer here (don't forget to initialize it)

    cout << "==========\n"
         << "Input data\n"
         << "==========\n"
         << "Enter the number of Kingdoms: ";
    cin >> count;
    cin.ignore();

    if (count < 1) return 1;

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

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

    // testing that "display(...)" works
    cout << "------------------------------" << endl
         << "The 2nd kingdom entered is" << endl
         << "------------------------------" << endl;
    sict::display(pKingdom[1]);
    cout << "------------------------------" << endl << endl;

    // TODO: deallocate the dynamic memory here

    return 0;
}

// read accepts data for a Kingdom from standard input
//
void read(sict::Kingdom& kingdom) {

    cout << "Enter the name of the Kingdom: ";
    cin.get(kingdom.m_name, 32, '\n');
    cin.ignore(2000, '\n');
    cout << "Enter the number of people living in " << kingdom.m_name << ": ";
    cin >> kingdom.m_population;
    cin.ignore(2000, '\n');
}
```

OUTPUT SAMPLE:

```
==========
Input data
==========
Enter the number of Kingdoms: 2
Kingdom #1:
Enter the name of the Kingdom: The_Vale
```

```
Enter the number of people living in The_Vale: 234567
Kingdom #2:
Enter the name of the Kingdom: The_Reach
Enter the number of people living in The_Reach: 567890
==========

------------------------------
The 2nd kingdom entered is
------------------------------
The_Reach, population 567890
------------------------------
```

To complete your coding

a) remove all comments that have been included to assist you with your tasks (marked with **TODO**).
b) Include in each file appropriate header comments uniquely identify the file (as shown above)
c) Preface each function definition in the implementation file with a function header comment explaining what the function does in a single phrase (as shown above).

IN-LAB SUBMISSION

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

Upload `Kingdom.h`, `Kingdom.cpp` and `w2_in_lab.cpp` to your matrix account. Recompile and rerun your code to make sure that everything works properly on matrix also.

To submit, run the following command 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 generated by the command and your program.

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 (`Kingdom.h`, `Kingdom.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 `Kingdom`s, and the *second* one receives an integer holding the number of elements in the array. This function calculates the total number of people living in all of the `Kingdom`s and prints the array information to the screen (in the reverse order) in the following format:

```
   ------------------------------<ENDL>
   Kingdoms of SICT<ENDL>
   ------------------------------<ENDL>
   3. KINGDOM_NAME, population POPULATION<ENDL>
   2. KINGDOM_NAME, population POPULATION<ENDL>
   1. KINGDOM_NAME, population POPULATION<ENDL>
   ------------------------------<ENDL>
   Total population of SICT: TOTAL_POPULATION<ENDL>
   ------------------------------<ENDL>
```

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

NOTE: this overload must call the `sict::display(...)` function you created for the *in-lab* part in order to display the name and the population of each kingdom in the array of kingdoms.

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

Your tasks include expanding the amount of dynamic memory allocated for the array of `Kingdom`s by one element after reading the original input, and then reading

the data for the new element as shown below. The test input and the expected output from your program are listed below this source code (the input to your program is shown in red). The output of your program should match **exactly** the sample output shown below.

```
// Workshop 2: Dynamic Memory
// File: w2_at_home.cpp
// Author: Cornel
// Date: 2019/05/06

#include <iostream>
#include "Kingdom.h"
#include "Kingdom.h"// intentional

using namespace std;
using namespace sict;

void read(sict::Kingdom&);

int main() {
    int count = 0; // the number of kingdoms in the array

    // TODO: declare the pKingdom pointer here (don't forget to initialize it)

    cout << "==========\n"
         << "Input data\n"
         << "==========\n"
         << "Enter the number of Kingdoms: ";
    cin >> count;
    cin.ignore();

    if (count < 1) return 1;

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

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

    // testing that "display(...)" works
    cout << "------------------------------" << endl
         << "The 1st kingdom entered is" << endl
         << "------------------------------" << endl;
    display(pKingdom[0]);
    cout << "------------------------------" << endl << endl;

    // expand the array of Kingdoms by 1 element
    {
        // TODO: allocate dynamic memory for count + 1 Kingdoms
        // 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 pKingdom pointer
        // add the new Kingdom
        cout << "==========\n"
             << "Input data\n"
             << "==========\n"
             << "Kingdom #" << count + 1 << ": " << endl;
        // TODO: accept input for the new element in the array
        count++;
        cout << "==========\n" << endl;
    }

    // testing that the overload of "display(...)" works
    display(pKingdom, count);

    // TODO: deallocate the dynamic memory here

    return 0;
}

// read accepts data for a Kingdom from standard input
//
void read(Kingdom& kingdom) {
    cout << "Enter the name of the Kingdom: ";
    cin.get(kingdom.m_name, 32, '\n');
    cin.ignore(2000, '\n');
    cout << "Enter the number of people living in " << kingdom.m_name << ": ";
    cin >> kingdom.m_population;
    cin.ignore(2000, '\n');
}
```

OUTPUT SAMPLE:

```
==========
Input data
==========
Enter the number of Kingdoms: 2
Kingdom #1:
Enter the name of the Kingdom: The_Vale
Enter the number of people living in The_Vale: 234567
Kingdom #2:
Enter the name of the Kingdom: The_Reach
Enter the number of people living in The_Reach: 567890
==========

------------------------------
The 1st Kingdom entered is
------------------------------
The_Vale, population 234567
------------------------------

==========
Input data
```

```
==========
Kingdom #3:
Enter the name of the Kingdom: The_Riverlands
Enter the number of people living in The_Riverlands: 123456
==========

------------------------------
Kingdoms of SICT
------------------------------
3. The_Riverlands, population 123456
2. The_Reach, population 567890
1. The_Vale, population 234567
------------------------------
Total population of SICT: 925913
------------------------------
```

To complete your coding

a) remove all comments that have been included to assist you with your tasks (marked with **TODO**).
b) Include in each file appropriate header comments that uniquely identify the file (as shown above)
c) Preface each function definition in the implementation file with a function header comment explaining what the function does in a single phrase (as shown above).

REFLECTION (40%)

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 topics 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 when you increase the size of an existing array of dynamically allocated memory?
2) The `Kingdom` structure stores the name of the kingdom 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 name itself. Why don't we need to use the `delete` e operator on this array itself?

3) There are two `display(...)` function definitions. How does the compiler know which definition to call from your `main` function?
4) Describe what have you have learned in this workshop.

Quiz Reflection

Add a section to `reflect.txt` called Quiz X Reflection. Replace the X with the number of the last quiz that you received and list the numbers of all questions that you answered incorrectly.

Then for each incorrectly answered question write your mistake and the correct answer to that question. If you have missed the last quiz, then write all the questions and their answers.

At-Home Submission

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

Upload `reflect.txt`, `Kingdom.h`, `Kingdom.cpp` and `w2_at_home.cpp` to your matrix account. Recompile and rerun your code and make sure everything works properly on matrix.

To submit, run the following command 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 generated by the command and your program.

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.