Part B – Foundations

Dynamic Memory

Workshop 2 (out of 10 marks - 3% 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
- to resize the amount of dynamically allocated memory
- to overload a global function
- to explain the difference between statically and dynamically allocated memory
- to describe to your instructor 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. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (with a penalty; see below). The “at-home” portion of the lab is due on the day that is two days before your next scheduled workshop (23:59).

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 `Kingdom` in the namespace `sict`. The 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>
```

Put the struct **definition** and the `display(...)` **declaration** in a header file named `Kingdom.h`. Put the definition of `display(...)` in an implementation file named `Kingdom.cpp`.

Complete the implementation of the `w2_in_lab.cpp` main module shown below (see the parts marked with **TODO**). Below the source code is the expected output from your program (the red color identifies what you should type as input to your program). The output of your program should match **exactly** the sample output shown below.

```
// OOP244 Workshop 2: Dynamic Memory
// File w2_in_lab.cpp
// Version 1.0

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

using namespace std;
```

```
using namespace sict;

void read(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 first kingdom entered is" << endl
         << "------------------------------" << endl;
    display(pKingdom[0]);
    cout << "------------------------------" << endl << endl;

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

------------------------------
The first kingdom entered is
------------------------------
The_Vale, population 234567
------------------------------
```

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 `Kingdom.h`, `Kingdom.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: (replace `profname.proflastname` with your professors Seneca userid)

`~profname.proflastname/submit 244_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%)

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 Kingdoms, 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 Kingdoms and prints the array information to the screen in the following format:

```
------------------------------<ENDL>
Kingdoms are<ENDL>
------------------------------<ENDL>
1. KINGDOM_NAME, population POPULATION<ENDL>
2. KINGDOM_NAME, population POPULATION<ENDL>
3. KINGDOM_NAME, population POPULATION<ENDL>
------------------------------<ENDL>
Total population of all Kingdoms: 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 the kingdom.

Upgrade the `w2_at_home.cpp` implementation file of the main module to expand the amount of dynamic memory allocated for the array of Kingdoms by one element after reading the original input. Then, read the data for the new element as shown below (see the parts marked with **TODO**, reuse the parts that you have done for the *in-lab* part). Below the source code is the expected output from your program (with red color is what you should type as input for your program). The output of your program should match **exactly** the sample output shown below.

```
// OOP244 Workshop 2: Dynamic Memory
// File w2_at_home.cpp
// Version 1.0

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

using namespace std;
using namespace sict;
```

```
void read(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 first 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);
    cout << endl;

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

------------------------------
The first kingdom entered is
------------------------------
The_Vale, population 234567
------------------------------

==========
Input data
==========
Kingdom #3:
Enter the name of the kingdom: The_Riverlands
Enter the number people living in The_Riverlands: 123456
==========

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

AT-HOME REFLECTION (40%)

Create a file named `reflect.txt` that contains answers to the following questions:

1) What happens to dynamic memory if it is not deallocated?
2) Why do you need to allocate new dynamic memory when you increase the size of an existing array of dynamically allocated memory?
3) The Kingdom structure stores the name 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. **Do we have a memory leak there?** Explain.
4) In your program, there are two `display(...)` functions. How does the compiler know which one should be called from your `main`? Explain.
5) Explain what have you 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).

Enter all the questions from the quiz for those you did not answer correctly.

Under each question, specify what your mistake was and what the correct answer is for that question. If you missed the last quiz, enter all the questions and the correct answer for each.

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`, `Kingdom.h`, `Kingdom.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: (replace `profname.proflastname` with your professor's Seneca userid)

`~profname.proflastname/submit 244_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.