Part C – Encapsulation

Constructors

Workshop 4

In this workshop, you are to code a class that represents an enrollment and processes transactions on that enrollment.

LEARNING OUTCOMES

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

- code a class for simple objects
- define a constructor that sets an object to a safe empty state
- overload a constructor to receive information from a client

SUBMISSION POLICY

The workshop is divided into 2 sections;

lab - 50% of the total mark

To be completed before the end of the lab period and submitted from the lab.

DIY - 50% of the total mark

To be completed within **5 days** after the day of your lab (meaning it will be **due 2 days prior to following lab period for the next Workshop**).

The *lab* section is to be completed after the workshop is published, and before the end of the lab session. The *lab* is to be submitted during the workshop period from the lab.

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

If you do not attend the workshop, you can submit the *lab* section along with your *DIY* section (see Submission Penalties below). The *DIY* portion of the lab is due on the day that is **5 days** after your scheduled *lab* workshop (by 23:59 or 11:59PM) (even if that day is a holiday).

The *DIY* (Do It Yourself) section of the workshop is a task that utilizes the concepts you have done in the **lab** section. This section is open ended with no detailed instructions other than the required outcome.

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.

Ask your professor if there are any additional requirements for your specific section.

Citation and Sources

When submitting Workshops, Project and assignment deliverables, a file called **sources.txt** must be present. This file will be submitted with your work automatically.

You are to write either of the following statements in the file "**sources.txt**":

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

Then add your name and your student number as signature

OR:

Write exactly which part of the code of the workshops or the assignment are given to you as help and who gave it to you or which source you received it from.

You need to mention the workshop name or assignment name and also mention the file name and the parts in which you received the code for help.

Finally add your name and student number as signature.

By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

SUBMISSION PENALTIES:

The following are penalties associated with the workshop if the submission policies above regarding due dates and other requirements are not met. It is advised to pay close attention to these penalties in order to avoid them.

- If the **lab** is submitted past the due date but before the rejection date, it will be **worth half of its total weight** for the Workshop. If a lab is worth 50% then it will be worth 25%.
- If the **diy** is submitted past the due date but before the rejection date, it will be **worth half of its total weight** for the Workshop. If the diy is worth 50% then it will be worth 25%.
- If a submitted **reflection** (reflect.txt) is deemed to be insufficient in depth or does not demonstrate a strong understanding of the core concepts used in the Workshop, a **potential maximum reduction of 30%** may be applied the overall mark of the Workshop.
- If any of lab, diy or reflection is missing the total mark of the workshop will be **zero**.

WORKSHOP DUE DATES

You can see the exact due dates of all assignments by adding -due after the submission command:

Run the following script from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS04/lab -due<ENTER>
~profname.proflastname/submit 244/NXX/WS04/DIY -due<ENTER>

COMPILING AND TESTING YOUR PROGRAM

All your code should be compiled using this command on matrix:
g++ -Wall -std=c++11 -o ws (followed by your .cpp files)
After compiling and testing your code, run your program as follows to check for possible memory leaks: (assuming your executable name is "ws") **valgrind ws**

Utils Module

For those of you who would like to reuse their previously developed functions and classes in later workshops, an empty module is added to the workshop called `Utils` (`Utils.cpp` and `Utils.h`). These files are empty and will be compiled with your workshop.

If you don't have any interest in reusing your previous functions, leave these files as they are, and they will not have any effect in the workshop.

If you have anything that you would like to add to the project, place their code in the `".cpp"` file and the definitions and prototypes in the `".h"` file. This header file will be included in the tester program.

In any case when submitting your work, make sure that these two files (empty or not) are submitted along with your workshop.

LAB - 50%

Saiyan Class

Design and code a class named `Saiyan` that holds information about an alien race of warriors known as the mighty Saiyans.

Place your class definition in a header file named `Saiyan.h` and your function definitions in an implementation file named `Saiyan.cpp`. Include in your coding all of the statements necessary for your code to compile under a standard C++ compiler, and within the `sdds` namespace.

The class `Saiyan` has the following four data members:

- m_name: A C-style string holding the name of the Saiyan. You may assume that the string is no longer than 30 characters, excluding the null terminator.
- m_dob: An integer indicating the year in which the Saiyan was born**(<2020**).
- m_power: An integer indicating the strength of the Saiyan **(> 0**).
- m_super: A Boolean value to indicate whether the Saiyan has the ability to evolve into a Super Saiyan. (a value of `false` indicates that the Siayna has not achieved this ability yet).

Saiyan has the following member functions (methods) that you need to implement in **Saiyan.cpp.** (make sure you reuse your code and DO NOT re-implement the logics already implemented)

- **bool isSuper() const** – a query that returns the value of **super**.
- **bool isValid()const** - returns **true** when the object is NOT in a safe empty state.
- **void setEmpty()** - sets the object to a safe empty state.
- **void display() const** - a query that displays the Saiyan's name and stats. This function displays **Invalid Saiyan!** if the object is in a safe empty state (when it is invalid);
- **void set(const char* name, int dob, int power, bool super = false)-** a setter function that stores valid data in a Saiyan object. The first parameter receives the address of a C-style string containing the Saiyan name, followed by 2 integers receiving dob and power. Last parameter receives a Boolean super. If the last argument is not provided, super_ is set to false.
  If any of the data is invalid, this function will set the object to a safe empty state.

- **bool hasLost(const Saiyan& other) const-** a query that accepts an unmodifiable reference to the other **Saiyan** object and returns whether the current object will win the battle or not. A battle is won when the power of the current objects exceeds the power of the other object. Moreover, if any of the objects are invalid this function returns false.

- Include two constructors in your class: a no-argument constructor that sets the object to a safe empty state and a three-argument constructor that receives the address of a C-style string containing the saiyan's name and 2 integers indicating date of birth and the power of the Saiyan.The three-argument constructor should set the value indicating that the Saiyan super ability to **false**.

Note that you must check the validity of users' input. If null pointer arguments are provided, or validation fails, the object is set to a ***safe empty state***.

CLIENT MODULE

Here is a sample of implementation file for the **SaiyanMain.cpp** main module that you should use to test your implementation:

```
// OOP244 Workshop 4: Constructors
// File: SaiyanMain.cpp
// Version: 1.0
```

```
// Date: 1/24/2020
// Author: Mohammad Shamas
// Description:
// This file tests lab section of your workshp
/////////////////////////////////////////////

#include <iostream>
#include "Saiyan.h"
#include "Saiyan.h"  // this is on purpose
using namespace std;
using namespace sdds;

int main() {
    // constructors
    Saiyan s1("Goku", 1990, 23000);
    Saiyan s2, s3;
    Saiyan badData[] = {
      Saiyan("Nappa", 2025, 1),
      Saiyan("Nappa", 2018, -1),
      Saiyan(nullptr, 2015, 1),
      Saiyan("", 2018, 5),
    };

    cout << "Testing Saiyan objects" << endl << endl;

    // testing two invalid Saiyans
    s3.display();
    s2.display();

    cout << endl;

    // testing Valid Saiyan
    s1.display();

    // setting the second one to Vegeta
    s2.set("Vegeta", 1989, 22000);

    s2.display();

    // setting the Saiyan Goku to SuperSaiyan;
    s1.set("Goku", 1990, 23000, true);

    s1.display();

    // Checking hasLost;

    cout << "S1 attacking S2, Battle " << (!(s1.hasLost(s2)) ? "Won" : "Lost")<< endl;

    cout << "S2 attacking S1, Battle " << (!(s2.hasLost(s1)) ? "Won" : "Lost") << endl;


    // making sure all the conditions are met for an inValid Saiyan.
    cout << endl << "None should be Valid:" << endl;
    for (int i = 0; i < 4; i++) {
        cout << "index: " << i << " - " << (badData[i].isValid() ? "V" : "Not v") <<
"alid." << endl;
    }
    return 0;
```

```
}

/* output:
Testing Saiyan objects

Invalid Saiyan!
Invalid Saiyan!

Goku
DOB: 1990 Power: 23000
Super Saiyan Ability: Not super Saiyan.

Vegeta
DOB: 1989 Power: 22000
Super Saiyan Ability: Not super Saiyan.

Goku
DOB: 1990 Power: 23000
Super Saiyan Ability: Super Saiyan.

S1 attacking S2, Battle Won
S2 attacking S1, Battle Lost

None should be Valid:
index: 0 - Not valid.
index: 1 - Not valid.
index: 2 - Not valid.
index: 3 - Not valid.

*/
```

To complete your coding

a) Include in each file an appropriate header comments uniquely identify the file (include your name, section, id and date of file was last modified)

b) Make sure your function prototypes have meaningful argument names to help understand what the function does.

LAB SUBMISSION

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

If not on matrix already, upload `Saiyan module` and the `SaiyanMain.cpp` program to your `matrix` account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

`~profname.proflastname/submit 244/NXX/WS04/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.

DIY (DO IT YOURSELF) – 50%

The **Saiyan** module will be updated to include a new data data member:

- m_level: An integer indicating the level of super saiyan **(> 0)**.

In this part you need to get rid of the static size for the Saiyan's name and use dynamic memory allocation when appropriate to allocate enough memory to set the name. In other words allocate the memory in the constructor and free the memory in the desctructor and make sure your set function does not cause memory leak.

Saiyan's member functions (methods) also need to be updated to meet the following specs:

- **`void powerup()`** – will power up the Saiyan by multiplying their level + 1 by the power only if they have super Saiyan abilities. *(New)*
- **`void set(const char* name, int dob, int power, int level = 0, bool super = false)-`**. This function will work same as before but will use dynamic memory allocation to set the name of the Saiyan if the provided parameter is valid. Make sure this does not cause memory leak.This function will also set the level only if the Saiyan has super Saiyan powers. (Modified)
- **`bool hasLost(Saiyan& other)`** – a query that accepts a reference to the other **`Saiyan`** object. This function needs to power up each Saiyan

before it returns whether the current object will win the battle or not. A battle is won when the power of the current objects exceeds the power of the other object. Moreover, if any of the objects are invalid this function returns false. (Modified)

MAIN PROGRAM:

SaiyanMain.cpp

```
// OOP244 Workshop 4: Constructors
// File: SaiyanMain.cpp
// Version: 1.0
// Date: 1/24/2020
// Author: Mohammad Shamas
// Description:
// This file tests lab section of your workshp
///////////////////////////////////////////

#include <iostream>
#include "Saiyan.h"
#include "Saiyan.h"  // this is on purpose
using namespace std;
using namespace sdds;

int main() {
    // constructors
    Saiyan s1("Goku", 1990, 20000);
    Saiyan s2, s3;
    Saiyan badData[] = {
      Saiyan("Nappa", 2025, 1),
      Saiyan("Nappa", 2018, -1),
      Saiyan(nullptr, 2015, 1),
      Saiyan("", 2018, 5),
    };

    cout << "Testing Saiyan objects" << endl << endl;

    // testing two invalid Saiyans
    s2.display();
    s3.display();

    cout << endl;

    // testing Valid Saiyan
    s1.display();

    // setting the second one to Vegeta
    s2.set("Vegeta", 1989, 22000);

    s2.display();

    // Checking hasLost;

    cout << "S1 attacking S2, Battle " << (!(s1.hasLost(s2)) ? "Won" : "Lost") << endl;
```

```
    cout << "S2 attacking S1, Battle " << (!(s2.hasLost(s1)) ? "Won" : "Lost") << endl;

    cout << endl;

    // setting the Saiyan Goku to SuperSaiyan;
    s1.set("Goku", 1990, 20000, 1, true);

    s1.display();

    // Checking hasLost;

    cout << endl;

    cout << "S1 attacking S2, Battle " << (!(s1.hasLost(s2)) ? "Won" : "Lost")<< endl;

    cout << "S2 attacking S1, Battle " << (!(s2.hasLost(s1)) ? "Won" : "Lost") << endl;

    // setting the Saiyan Vegeta to SuperSaiyan;

    cout << endl;

    s2.set("Vegeta", 1990, 22000, 1, true);

    s2.display();

    // setting the Saiyan Goku to SuperSaiyan;

    cout << endl;

    s1.set("Goku", 1990, 20000, 2, true);

    s1.display();

    // Checking hasLost;

    cout << endl;

    cout << "S1 attacking S2, Battle " << (!(s1.hasLost(s2)) ? "Won" : "Lost") << endl;

    cout << "S2 attacking S1, Battle " << (!(s2.hasLost(s1)) ? "Won" : "Lost") << endl;

    // making sure all the conditions are met for an inValid Saiyan.
    cout << endl << "None should be Valid:" << endl;
    for (int i = 0; i < 4; i++) {
        cout << "index: " << i << " - " << (badData[i].isValid() ? "V" : "Not v") <<
"alid." << endl;
    }
    return 0;
}

/* output:
Testing Saiyan objects

Invalid Saiyan!
Invalid Saiyan!

Goku
```

```
DOB: 1990 Power: 20000
Super Saiyan Ability: Not super Saiyan.

Vegeta
DOB: 1989 Power: 22000
Super Saiyan Ability: Not super Saiyan.

S1 attacking S2, Battle Lost
S2 attacking S1, Battle Won

Goku
DOB: 1990 Power: 20000
Super Saiyan Ability: Super Saiyan.
Super Saiyan level is: 1

S1 attacking S2, Battle Won
S2 attacking S1, Battle Lost

Vegeta
DOB: 1990 Power: 22000
Super Saiyan Ability: Super Saiyan.
Super Saiyan level is: 1

Goku
DOB: 1990 Power: 20000
Super Saiyan Ability: Super Saiyan.
Super Saiyan level is: 2

S1 attacking S2, Battle Won
S2 attacking S1, Battle Lost

None should be Valid:
index: 0 - Not valid.
index: 1 - Not valid.
index: 2 - Not valid.
index: 3 - Not valid.
*/
```

REFLECTION

Study your final solutions for each deliverable of the Workshop, 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 and the result is suggested to be at least 150 words in length.**

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 and how you solved them. Add any other comments you wish to make. Submit this file along with your "DIY" submission.

DIY SUBMISSION

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

If not on matrix already, upload `SaiyanMain.cpp` program and the updated `Saiyan` module to your `matrix` account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

`~profname.proflastname/submit 244/NXX/WS04/diy`<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.