Workshop #9: Derived Classes and Resources

Version 1.0

The Contact Class

Learning Outcomes

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

- Apply the rule of three for the derived classes with resources.
- Use your acquired skills throughout the semester to read a file into dynamically allocated memory.
- describe what you have learned in completing this workshop

Submission Policy

The workshop is divided into one coding part and one non-coding part:

- Part 1 (**LAB**): A step-by-step guided workshop, worth 100% of the workshop's total mark
  > Please note that the part 1 section is **not to be started in your first session of the week**. You should start it on your own before the day of your class and join the first session of the week to ask for help and correct your mistakes (if there are any).

- Part 2 (reflection): non-coding part. The reflection doesn't have marks associated with it but can incur a **penalty of max 40% of the whole workshop's mark** if your professor deems it insufficient (you make your marks from the code, but you can lose some on the reflection).

Due Dates

The Due dates depend on your section. Please choose the "-due" option of the submitter program to see the exact due date of your section:

```
~profname.proflastname/submit 2??/wX/pY_sss -due<ENTER>
```

- Replace **??** with your subject code (`00 or 44`)
- Replace **X** with Workshop number: [`1 to 10`]
- Replace **Y** with the part number: [`1 or 2`]
- Replace **sss** with the section: [`naa, nbb, nra, zaa, etc...`]

Late penalties

You are allowed to submit your work up to 2 days after the due date with a 30% penalty for each day. After that, the submission will be closed and the mark will be zero.

Citation

Every file that you submit must contain (as a comment) at the top:
your name, **your Seneca email**, **Seneca Student ID** and the **date** when you completed the work.

For work that is done entirely by you (ONLY YOU)

If the file contains only your work or the work provided to you by your professor, add the following message as a comment at the top of the file:

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

For work that is done partially by you.

If the file contains work that is not yours (you found it online or somebody provided it to you), **write exactly which part of the assignment is given to you as help, who gave it to you, or which source you received it from.** 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 wrongdoing.

> - Add the citation to the file in which you have the borrowed code
> - In the 'reflect.txt` submission of part 2 (DIY), add exactly what is added to which file and from where (or whom).

> ⚠️ This Submission Policy only applies to the workshops. All other assessments in this subject have their own submission policies.

If you have helped someone with your code

If you have helped someone with your code. Let them know of these regulations and in your 'reflect.txt' of part 2 (DIY), write exactly which part of your code was copied and who was the recipient of this code.
By doing this you will be clear of any wrongdoing if the recipient of the code does not honour these regulations.

Compiling and Testing Your Program

All your code should be compiled using this command on `matrix`:

```
g++ -Wall -std=c++11 -g -o ws file1.cpp file2.cpp ...
```

- `-Wall` : the compiler will report all warnings

- `-std=c++11` : the code will be compiled using the C++11 standard
- `-g` : the executable file will contain debugging symbols, allowing *valgrind* to create better reports
- `-o ws` : the compiled application will be named `ws`

After compiling and testing your code, run your program as follows to check for possible memory leaks (assuming your executable name is `ws`):

```
valgrind --show-error-list=yes --leak-check=full --show-leak-kinds=all --track-origins=yes ws
```

- `--show-error-list=yes` : show the list of detected errors
- `--leak-check=full` : check for all types of memory problems
- `--show-leak-kinds=all` : show all types of memory leaks identified (enabled by the previous flag)
- `--track-origins=yes` : tracks the origin of uninitialized values (`g++` must use `-g` flag for compilation, so the information displayed here is meaningful).

To check the output, use a program that can compare text files. Search online for such a program for your platform, or use *diff* available on `matrix` .

> Note: All the code written in workshops and the project must be implemented in the **sdds** namespace, unless instructed otherwise.

Custom code submission

If you have any additional custom code, (i.e. functions, classes etc) that you want to reuse in the workshop save them under a module called Utils (`Utils.cpp and Utils.h`) and submit them with your workshop using the instructions in the "Submitting Utils Module" section.

PART 1 (100%)

The Person class and Tools module (fully implemented)

The **Person** class encapsulates a person with first, middle and last name and is instantiated in an empty state.

All the name fields are stored dynamically and are read from istream through comma-separated data entry.

> The C-string manipulation and dynamic memory allocation logic are provided by the Tools module.

A Person is valid if it has valid first and last names. The middle name is optional, therefore the data entry for a Person can be done in the following two ways:

1- A Person with a middle name
Data entry:

```
Homer,Jay,Simpson<ENTER>
```

Displayed as follows:

```
Homer Jay Simpson
```

2- A Person without a middle name
Data entry:

```
Lisa,,Simpson<ENTER>
```

Displayed as follows

```
Lisa Simpson<ENTER>
```

Walk through the Person class using the `nameTester.cpp` program; study and understand it.

The Contact class

Create a Class Module called **Contact**, derived from the **Person** class. A contact is a person with an address.

The Contact class stores the address in four attributes: 1- Address (Dynamic, unknown length) 2- City (Dynamic, unknown length) 3- Province: stored by two characters (for the province code, like ON, AL, BC, etc...) 4- Postal code: stored in six characters.

Like a **Person**, **Contact** is instantiated in an empty state and later is populated from istream using comma-separated values. All the attributes defined in Contact are mandatory and can not be empty or not provided.

A Contact is read from istream as follows:

```
      Homer,Jay,Simpson<ENTER>
      70 the pond road,North York,ON,M3J3M6<ENTER>
OR
      Homer,,Simpson<ENTER>
      70 the pond road,North York,ON,M3J3M6<ENTER>
```

And it is displayed as follows:

```
      Homer Jay Simpson
      70 the pond road
      North York ON
      M3J 3M6
OR
      Homer Simpson
      70 the pond road
      North York ON
      M3J 3M6
```

If any data other than the middle name is missing or exceeds the field length, the Contact should be put in an invalid state.

Displaying an invalid **Contact** object will be quietly ignored and no action will be taken.

the Contact class implementation

- Initialize all the attributes to their default empty states in the class declaration. (as a result your defualt constructor should be empty)
- Use the Tools functions for your C-string work and dynamic memory allocation. (do not include `<cstring>`)
- Invoke the base class's constructors and Methods in copy construction and copy assignment to make sure the base class's resources are managed properly.
- Override all the virtual functions and virtual operators of the Person class and implement the rule of three.
- Apart from the overridden methods and operators, you can create any additional method to help you implement this class.

Tester program

```
/*****************************************************************
// OOP244 Workshop 9:
// File w9_tester.cpp
// Version 1.0
// Date 2021/11/19
// Author   Fardad Soleimanloo
// Description
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
/////////////////////////////////////////////////////////////////
*****************************************************************/

#include <iostream>
#include <fstream>
#include "Contact.h"

using namespace std;
using namespace sdds;
Contact readContact(ifstream& ifstr) {
   Contact fC;
   ifstr >> fC;
   return fC;
}

int main() {
```

```
Contact C;
ifstream file("contacts.txt");
cout << "Empty Contact: >" << C << "<" << endl;
cout << "Enter the following:" << endl
   << "Homer,Jay,Simpson" << endl
   << "70 the pond road,North York,ON,M3J3M6" << endl << endl;
cout << "Name and address" << endl << "> ";
cin >> C;
if (cin)
   cout << "OK!" << endl;
else {
   cout << "Date entry implemented incorrectly" << endl;
   cin.clear();
   cin.ignore(1000, '\n');
}
cout << "Contact:" << endl << C << endl << endl;
//--------------------------------------
cout << "Enter the following:" << endl
   << "Homer,Jay,Simpson" << endl
   << "70 the pond road,North York,ONT,M3J3M6" << endl << endl;
cout << "Name and address" << endl << "> ";
cin >> C;
if (cin)
   cout << "Date entry implemented incorrectly" << endl;
else {
   cin.clear();
   cin.ignore(1000, '\n');
}
cout << "Empty Contact: >" << C << "<" << endl << endl;
//--------------------------------------
```

```
cout << "Enter the following:" << endl
   << "Homer,Jay,Simpson" << endl
   << "70 the pond road,North York,ON,M3J 3M6" << endl << endl;
cout << "Name and address" << endl << "> ";
cin >> C;
if (!cin){
   cin.clear();
   cin.ignore(1000, '\n');
}
cout << "Empty Contact: >" << C << "<" << endl << endl;
//--------------------------------------
cout << "Enter the following:" << endl
   << "Homer,Jay,Simpson" << endl
   << "70 the pond road,,ON,M3J3M6" << endl << endl;
cout << "Name and address" << endl << "> ";
cin >> C;
if (!cin){
   cin.clear();
   cin.ignore(1000, '\n');
}
cout << "Empty Contact: >" << C << "<" << endl << endl;
//--------------------------------------
cout << "Enter the following:" << endl
   << "Homer,Jay,Simpson" << endl
   << ",North York,ON,M3J3M6" << endl << endl;
cout << "Name and address" << endl << "> ";
cin >> C;
if (!cin){
   cin.clear();
   cin.ignore(1000, '\n');
```

```
    }
    cout << "Empty Contact: >" << C << "<" << endl << endl;
    cout << "Name and addresses in file: " << endl;
    while (file) {
       C = readContact(file);
       if (file) cout << C << endl;
    }
    return 0;
}
```

output

```
Empty Contact: ><
Enter the following:
Homer,Jay,Simpson
70 the pond road,North York,ON,M3J3M6

Name and address
> Homer,Jay,Simpson
70 the pond road,North York,ON,M3J3M6
OK!
Contact:
Homer Jay Simpson
70 the pond road
North York ON
M3J 3M6


Enter the following:
Homer,Jay,Simpson
70 the pond road,North York,ONT,M3J3M6

Name and address
> Homer,Jay,Simpson
70 the pond road,North York,ONT,M3J3M6
Empty Contact: ><

Enter the following:
Homer,Jay,Simpson
```

```
70 the pond road,North York,ON,M3J 3M6

Name and address
> Homer,Jay,Simpson
70 the pond road,North York,ON,M3J 3M6
Empty Contact: ><

Enter the following:
Homer,Jay,Simpson
70 the pond road,,ON,M3J3M6

Name and address
> Homer,Jay,Simpson
70 the pond road,,ON,M3J3M6
Empty Contact: ><

Enter the following:
Homer,Jay,Simpson
,North York,ON,M3J3M6

Name and address
> Homer,Jay,Simpson
,North York,ON,M3J3M6
Empty Contact: ><

Name and addresses in file:
Homer Jay Simpson
70 the pond road
North York ON
M3J 3M6
```

```
Fred Soley
1 York Gate Blvd
North York ON
M3N 3A1

John Al Doe
1750 Finch Ave E
North York ON
M2J 2X5
```

Files needed for submission

```
Tools.h
Tools.cpp
Person.h
Person.cpp
Contact.h
Contact.cpp
w9_tester.cpp
contacts.txt
```

PART 1 Submission

Upload your source code and data file to your `matrix` account. Compile and run your code using the `g++` compiler as shown above and make sure that everything works properly.

Then, run the following command from your account

- replace `profname.proflastname` with your professor's Seneca userid
- replace **??** with your subject code (2**00** or 2**44**)
- replace **#** with the workshop number
- replace **X** with the workshop part number (**1** or **2**)

```
~profname.proflastname/submit 2??/w#/pX
```

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. Re-submissions will attract a penalty.

Part 2: Reflection

Study your final solutions for each deliverable of the workshop **and the most recent milestones of the project**, 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 **the project milestones** and mention any issues that caused you difficulty.

Submission Process:

Upload the files listed above to your `matrix` account. Compile and run your code using the `g++` compiler as shown in Compiling and Testing Your Program and make sure that everything works properly.

Then, run the following command from your matrix account

```
~profname.proflastname/submit 2??/wX/pY_sss  <ENTER>
```

- Replace **??** with your subject code (`00 or 44`)
- Replace **X** with Workshop number: [`1 to 10`]
- Replace **Y** with the part number: [`1 or 2`]
- Replace **sss** with the section: [`naa, nbb, nra, zaa, etc...`]

and follow the instructions.