Part D – Polymorphism

Virtual Functions

Workshop 8 (1.0 in_lab)

In this workshop, you will work with a set of classes in a hierarchy where the base is an abstract base class. Through these classes we will implement inclusion polymorphism that is, we will use virtual functions to enable dynamic binding. The classes used in this workshop will be the Writing Instruments such as a Pencil and Pen.

Learning Outcomes

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

- To define a pure virtual function
- To code an abstract base class
- To implement behavior specified in a pure virtual function
- To explain the difference between an abstract class and a concrete class
- To describe to your instructor what you have learned in completing this workshop

Submission Policy

The workshop is comprised of only 1 section;
in-lab - 100% of the total mark
To be completed before the end of the lab period and submitted from the lab.
The *in-lab* section is to be completed after the workshop is published, and before the end of the lab session. The *in-lab* is to be submitted during the workshop period form the lab.
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 workshop, you can submit the *in-lab* section along with your *at-home* section (see penalties below).

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 the Lab part of this workshop, 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 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.

Late Submission Penalties:

-*In-lab* portion submitted late
0 for *in-lab*.

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/WS08/in_lab -due<ENTER>

COMPILING AND TESTING YOUR PROGRAM

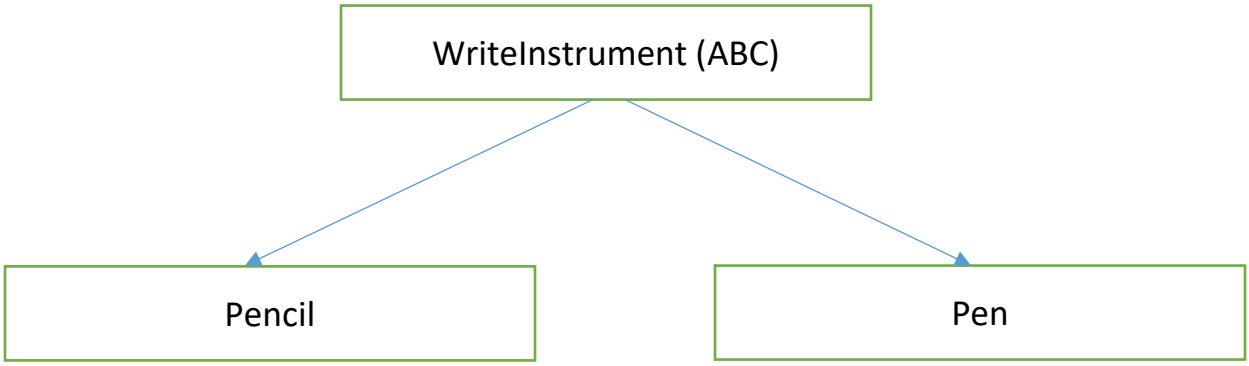
All your code should be compiled using this command on matrix:

`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 <ENTER>`

IN-LAB (100%)



In this workshop we will be establishing the above class hierarchy. The **WriteInstrument** class will serve as the **abstract base class** from where the **Pencil** and **Pen** classes will derive from. WriteInstrument will have pure virtual functions that are then overloaded by Pencil and Pen.

WRITE INSTRUMENT MODULE

The **WriteInstrument** module will consist of only a **header** file that contains an *abstract base class*. It will have no private members and only public pure virtual functions.

To accomplish the above follow these guidelines:

Design and code a class named `WriteInstrument`.

Follow the usual rules for creating a module: (i.e. Compilation safeguards for the header file, namespaces and coding styles your professor asked you to follow).

Create a class called **WriteInstrument**.

PUBLIC MEMBERS:

Destructor

The **WriteInstrument** class will require the presence of a **virtual** destructor. Note that this destructor while **virtual**, it isn't pure. Rather than making it pure, simply give it an empty function body instead via a pair of empty curly braces { }.

Pure Virtual Members

```
virtual void write(const char*);
This function will have the WriteInstrument write something.

virtual void refill(int);
This function will refill the WriteInstrument.

virtual bool filled() const;
This function will check if the WriteInstrument is filled
with material to write with (eg. ink for a Pen).

virtual ostream& display(ostream& os) const;
This function display details about the WriteInstrument.
```

PENCIL MODULE

The **Pencil** module will be the first of the two classes that derives and implements on top of the **WriteInstruments** abstract base class.

Create the following constants in the **Pencil** header:

> `HB_MAX` with a value of 2.
> This constant number is used to set the maximum length of the **HB_scale** data member.
> `GRAPHITE_MAX` with a value of 100.
> This constant number is used to set the maximum value of the **graphite** data member.

PRIVATE MEMBERS:

A **Pencil** will have the following data members:

- `HB_scale`
  This is a **character array** representing the HB value of the Pencil. The HB value indicates the how hard / soft the Pencil is. Utilize the HB_MAX constant to set the size of this array. Remember to consider the **nullbyte**.
- `graphite`
  This is an **integer** representing the amount of graphite left in the Pencil. In other words, how much material we have left to write with.

PUBLIC MEMBERS:

Constructors/Destructors

Pencil objects will be making use of **constructors** as well to handle their creation. There are two **constructors** that are required:

1. Default constructor – This constructor should set a Pencil object to **a safe empty state**. The notion of what the safe empty state will be left up to your design but choose reasonable values.
2. 2 Argument Constructor – This constructor will take in 2 parameters:
   a. A **constant character pointer** that represents the **HB_scale** of the Pencil
   b. An **integer** representing the **graphite** amount in the Pencil.

   A little bit of validation needs to occur in the 2 arg constructor. **If the provided constant character pointer** for the HB_scale is either **nullptr** or an **empty string** then set the Pencil to **a safe empty state**.

   Otherwise set the data members to be the values of the parameters as they are. For the HB_scale simply copy over **HB_MAX** characters. Remember to close off the string with a **nullbyte** at the last index.

If the **graphite** value provided from the parameter is less than 1 or greater than 100 then set the data member **graphite to** the value of GRAPH-ITE_MAX.

Other Members

The other member functions will be overloading the inherited **pure virtual functions** from the **WriteInstrument** class by giving them implementation.

```
virtual void write(const char*);
```
This member function will do the following if **graphite** is available (non-zero):

1. Print out:

```
"Writing out the following: [text] with a Pencil!" <newline>
```

Where [text] is the value of the const char* parameter of this function.

2. Reduce the amount of **graphite** in the Pencil by the number of **spaces** found in the **text times 5**. In order to do this portion, consider applying a loop on the **text** and determine those number of spaces. If the amount of **graphite** is reduced to less than zero set it to zero instead.

If the amount of graphite is zero then print out the following instead:

```
"We can’t write without graphite!" <newline>
```

```
virtual void refill(int);
```
This member function will refill the amount of **graphite** in the Pencil with the amount from the parameter. If this refill causes the amount of graphite to exceed GRAPHITE_MAX then set it to GRAPHITE_MAX instead.

It will also print out the following message:

```
"Refilling the pencil with graphite” <newline>
```

```
virtual bool filled() const;
```

This member function will return true if the Pencil has **graphite** to write with and false otherwise.

```
virtual ostream& display(ostream& os) const;
```
This member function will display the current details of the Pencil by utilizing the parameter as the output stream object. If the Pencil is in an **empty** state the following will be printed:

```
"This is an empty Pencil" <newline>
```

If the Pencil isn't empty then it will print:

```
"Pencil Details" <newline>
"HB Value: [HB_scale]" <newline>
"Graphite Remaining: [graphite]" <newline>
```

Return **os** at the end of the function.

Refer to the sample output for details.

PEN MODULE

The **Pen** module will be the second of the two classes that derives and implements on top of the **WriteInstruments** abstract base class. It is very similar to the Pencil class.

Create the following constants in the **Pen** header:

> `INK_MAX` with a value of 50.
> This constant number is used to set the maximum value of the **ink** data member.

PRIVATE MEMBERS:

A **Pen** will have the following data members:

- `style`
  This is a **character pointer** representing the style of the Pen. There is no particular maximum left for this data member. Utilize dynamic memory for it.
- `ink`
  This is an **integer** representing the amount of ink left in the Pen. In other words, how much material we have left to write with.

PUBLIC MEMBERS:

Constructors/Destructors

Pen objects will be making use of **constructors** as well to handle their creation. There are two **constructors** that are required:

1. Default constructor – This constructor should set a Pen object to **a safe empty state**. The notion of what the safe empty state will be left up to your design but choose reasonable values.
2. 2 Argument Constructor – This constructor will take in 2 parameters:
   a. A **constant character pointer** that represents the **style** of the Pen
   b. An **integer** representing the **ink** amount in the Pen.

   A little bit of validation needs to occur in the 2 arg constructor. **If the provided constant character pointer** for the style is either **nullptr** or an **empty string** then set the Pencil to **a safe empty state**.

   Otherwise set the data members to be the values of the parameters as they are. Allocate as much memory is as needed for the **style** data member. Don't forget the **nullbyte**.

   If the **ink** value provided from the parameter is less than 1 or greater than 50 then set the data member **ink to** the value of INK_MAX.

As there is dynamic memory in play, remember to make sure there are no memory leaks.

Other Members

The other member functions will be overloading the inherited **pure virtual functions** from the **WriteInstrument** class by giving them implementation.

```
virtual void write(const char*);
```
This member function will do the following if **ink** is available (non-zero):

3. Print out:

```
"Writing out the following: [text] with a Pen!" <newline>
```

Where [text] is the value of the const char* parameter of this function.

4. Reduce the amount of **ink** in the Pen by the number of **non-space characters** found in the **text times 2**. In order to do this portion, consider applying a loop on the **text** and determine those number of non-spaces. If the amount of **ink** is reduced to less than zero set it to zero instead.

If the amount of ink is zero then print out the following instead:

```
"We can’t write without ink!" <newline>
```

```
virtual void refill(int);
```
This member function will refill the amount of **ink** in the Pen with the amount from the parameter. If this refill causes the amount of ink to exceed INK_MAX then set it to INK_MAX instead.

It will also print out the following message:

```
"Refilling the pen with ink” <newline>
```

```
virtual bool filled() const;
```
This member function will return true if the Pen has **ink** to write with and false otherwise.

```
virtual ostream& display(ostream& os) const;
```
This member function will display the current details of the Pen by utilizing the parameter as the output stream object. If the Pen is in an **empty** state the following will be printed:

```
"This is an empty Pen" <newline>
```

If the Pencil isn't empty then it will print:

```
"Pen Details" <newline>
"Style: [style]" <newline>
"Ink Remaining: [ink]" <newline>
```

Return **os** at the end of the function.

Refer to the sample output for details.

IN-LAB MAIN MODULE

```
/*************************************************************************
// OOP244 Workshop 8: Virtual Functions
// File WriteTester.cpp
// Version 1.0
// Date       2019/11/11
// Author     Hong Zhan (Michael) Huang
// Description
// Tests the WriteInstrument and its derived classes via virtual functions
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Michael
/////////////////////////////////////////////////////////////////
*************************************************************************/

#include <iostream>
#include "Pen.h"
#include "Pen.h"
#include "Pencil.h"
#include "Pencil.h"

using namespace std;
using namespace sdds;

ostream& line(int len, char ch) {
  for (int i = 0; i < len; i++, cout << ch);
  return cout;
}
ostream& number(int num) {
  cout << num;
  for (int i = 0; i < 9; i++) {
    cout << " - " << num;
  }
  return cout;
}
```

```
int main() {

  cout << "Pencil & Pen default constr" << endl;
  line(64, '-') << endl;
  number(1) << endl;
  Pencil pe1;
  Pen pp1;
  pe1.display(cout) << endl;
  pp1.display(cout) << endl;

  cout << "Pencil & Pen 2 arg constr invalid" << endl;
  line(64, '-') << endl;
  number(2) << endl;
  Pencil pe2("HB", -15);
  Pen pp2("Fountain", -12);
  pe2.display(cout) << endl;
  pp2.display(cout) << endl;

  cout << "Use WriteInstrument pointers to hold Pencils and Pens" << endl;
  line(64, '-') << endl;
  number(3) << endl;
  WriteInstrument* writers[3];
  writers[0] = new Pencil("9A", 15);
  writers[1] = new Pen("Feather", 14);
  writers[2] = new Pencil("2B", 50);

  for (int i = 0; i < 3; ++i)
    writers[i]->display(cout) << endl;

  cout << "Use each Writing Instrument to write some text" << endl;
  line(64, '-') << endl;
  number(4) << endl;
  string strs[3] = { "Still my heart is blazing", "I am a cat", "A I U E O" };
  for (int i = 0; i < 3; ++i) {
    writers[i]->write(strs[i].c_str()); cout << endl;
    writers[i]->display(cout) << endl;
  }

  cout << "Write with an empty Pencil & Pen" << endl;
  line(64, '-') << endl;
  number(5) << endl;
  writers[0]->write("Get em to Rock, get em to Burst!"); cout << endl;
  writers[1]->write("Grasp the Truth! Hold it High!"); cout << endl;

  cout << "Refill an empty Pencil" << endl;
  line(64, '-') << endl;
  number(6) << endl;
  writers[0]->refill(150);
  writers[0]->display(cout);
}
```

EXECUTION EXAMPLE Red values are user entry

```
Pencil & Pen default constr
----------------------------------------------------------------
1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
This is an empty Pencil
```

```
This is an empty Pen

Pencil & Pen 2 arg constr invalid
----------------------------------------------------------------
2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2
Pencil Details
HB Value: HB
Graphite Remaining: 100

Pen Details
Style: Fountain
Ink Remaining: 50

Use WriteInstrument pointers to hold Pencils and Pens
----------------------------------------------------------------
3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3
Pencil Details
HB Value: 9A
Graphite Remaining: 15

Pen Details
Style: Feather
Ink Remaining: 14

Pencil Details
HB Value: 2B
Graphite Remaining: 50

Use each Writing Instrument to write some text
----------------------------------------------------------------
4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4
Writing out the following: Still my heart is blazing with a Pencil!

Pencil Details
HB Value: 9A
Graphite Remaining: 0

Writing out the following: I am a cat with a Pen!

Pen Details
Style: Feather
Ink Remaining: 0

Writing out the following: A I U E O with a Pencil!

Pencil Details
HB Value: 2B
Graphite Remaining: 30

Write with an empty Pencil & Pen
----------------------------------------------------------------
5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5
We can't write without graphite!

We can't write without ink!

Refill an empty Pencil
----------------------------------------------------------------
```

```
6 - 6 - 6 - 6 - 6 - 6 - 6 - 6 - 6 - 6
Refilling the pencil with graphite
Pencil Details
HB Value: 9A
Graphite Remaining: 100
```

IN-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 WriteInstrument, Pencil and Pen modules and the WriteTester.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/WS08/in_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.