CS100
Introduction to Programming

Tutorial 5: Composition, Initializer lists, include guards, and inheritance

Problem 1

Basic implementation of a class

Implement class Screen

- The private member variables are

```
int m_width
int m_height
```

- The public interface has
  - One accessor/mutator procedure for each member variable (`GetWidth, SetWidth, GetHeight, SetHeight`)
  - Exactly one constructor which takes two integer parameters to give initial values to the width and height
  - One addition procedure to return the number of pixels on the screen (`int GetNumberPixels()`)
    - Note: The number of pixels is equal to width * height
- Separate the declaration and implementation into 2 files called Screen.cpp / Screen.hpp

Test class Screen

- Test your screen class using a separate file called main.cpp, which contains

```
#include <stdlib.h>
#include <stdio.h>
#include "Screen.hpp”

int main() {
  Screen myScreen( 640, 480 );
  printf( "The number of pixels on my screen ” );
  printf( "is %d\n", myScreen.GetNumberPixels() );
  return 0;
}
```

- Compile using the command

```
g++ Screen.cpp main.cpp –o main
```

Problem 2

Compositional Relationships (on initializer lists and include guards)

Implement class Computer

- Only one private member variable (compositional relationship!)

```
Screen m_screen
```

- The public interface has
  - One accessor/mutator procedure for **`m_screen`** (**`GetScreen, SetScreen`**)
  - Exactly one constructor which takes no parameters
- Separate the declaration and implementation in 2 files called Computer.cpp / Computer.hpp

Test class Computer

- Test your Computer class using a new main.cpp file which contains

```
#include <stdlib.h>
#include <stdio.h>
#include "Computer.hpp”
#include "Screen.hpp”

int main() {
  Computer myComputer;
  printf( "My computer's screen has %d pixels\n",
      myComputer.GetScreen().GetNumberPixels() );
  Screen myScreen(800,600);
  myComputer.SetScreen(myScreen);
  printf( "My computer's screen has %d pixels\n",
      myComputer.GetScreen().GetNumberPixels() );
  return 0;
}
```

Test class Computer

- Try to compile using the command
  `g++ Screen.cpp Computer.cpp main.cpp –o main`

- Questions:
  - What are the problems that you ran into?
  - How can we use an initializer list to construct computer?
  - How can we use include-guards to avoid double declarations?

Problem 3

Precedence of Overriding over Overloading

Extend class Computer

- Add a new member variable

```
int m_flops
```

- Extend/Modify the public interface
  - One additional accessor/mutator procedure for **`m_flops`** (`GetFlops, SetFlops`)
  - Modify constructor of **`Computer`** to take one **`int`** parameter to initialize **`m_flops`**

Implement class Laptop

- **`Laptop`** inherits from Computer
- Declare new member variable (divides flops if in battery mode)
  `int m_slowDownFactor`
- The public interface has
  - One getter/setter for **`m_slowDownFactor`**
    (**`GetSlowDownFactor, SetSlowDownFactor`**)
  - Exactly one constructor which takes two integer parameters for setting **`m_flops`** (in base class) and **`m_slowDownFactor`** (in child class)
  - One new overloading procedure
    `int GetFlops( int batteryMode )`
    - if **`batteryMode = 0`**, the laptop is powered and flops is equal to the original value
    - If **`batteryMode = 1`**, the laptop is on battery and flops is equal to **`m_flops / m_slowDownFactor`**

Test class Laptop

- Test your Laptop class using a new main.cpp file which contains

```
#include <stdlib.h>
#include <stdio.h>
#include "Computer.hpp”
#include "Screen.hpp”
#include "Laptop.hpp”
int main() {
  Laptop myLaptop( 1000, 2 );
  printf( "In powered mode, my laptop has %d flops\n",
      myLaptop.GetFlops(0) );
  printf( "In battery mode, my laptop has %d flops\n",
      myLaptop.GetFlops(1) );
  return 0;
}
```

Test class Laptop

- Try to compile using the command
  `g++ Screen.cpp Computer.cpp Laptop.cpp main.cpp –o main`

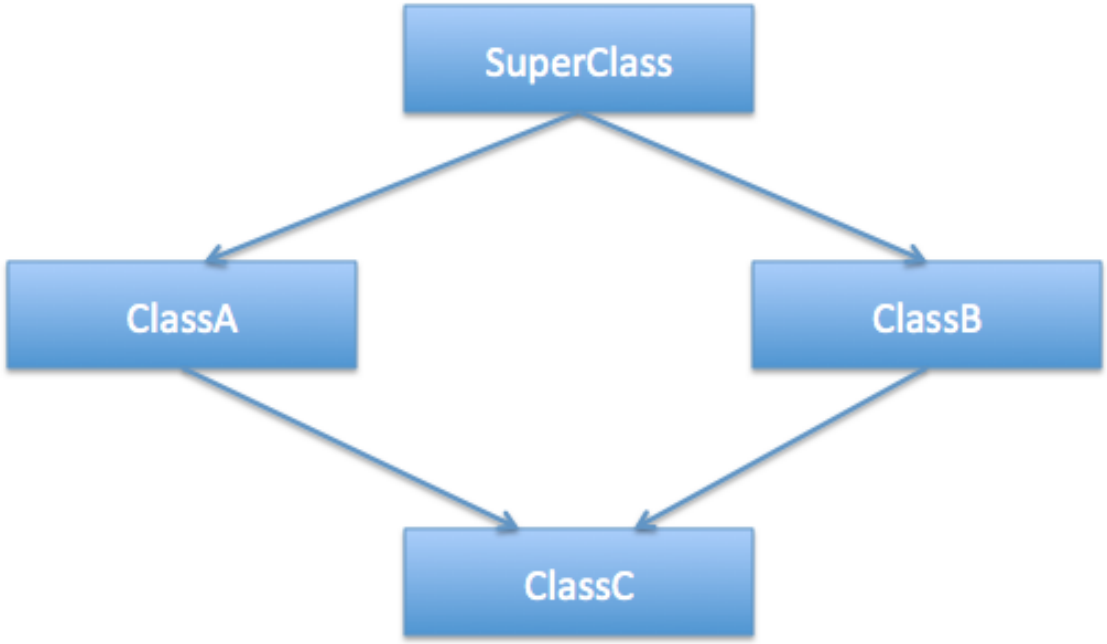
- Questions:
  - What are the problems that you ran into?
  - How can we use an initializer list to construct the members of the parent class?
  - By overloading the GetFlops procedure, can we still access the original GetFlops from the base class?
  - How to solve the problem?

Problem 4

Complex inheritance

Diamond problem

- Originates from multiple inheritance



Example

```
#include <iostream>

class LivingThing {
public:
    void breathe() {
        printf("I'm breathing as a living thing.\n”);
    }
};

class Animal : public LivingThing {
public:
    void breathe() {
        printf("I'm breathing as an animal.\n”);
    }
};

class Reptile : public LivingThing {
public:
    void breathe() {
        printf("I'm breathing as a reptile.\n”);
    }
};

class Snake : public Animal, public Reptile {
public:
    void breathe() {
        printf("I'm breathing as a snake.\n”);
    }
};

int main() {
    Snake snake;
    snake.breathe();
    return 0;
}
```

Example

- Example has been added to your gitlab project repository, in a sub-folder called **DiamondCase**

Does it compile and run?

```
g++ test0.cpp -o main
```

What about this?

```
#include <iostream>

class LivingThing {
public:
    void breathe() {
        printf("I'm breathing as a living thing.\n");
    }
};

class Animal : public LivingThing {
public:
    void breathe() {
        printf("I'm breathing as an animal.\n");
    }
};

class Reptile : public LivingThing {
public:
    void breathe() {
        printf("I'm breathing as a reptile.\n");
    }
};

class Snake : public Animal, public Reptile {
};

int main() {
    Snake snake;
    snake.breathe();
    return 0;
}
```

Does it compile?

```
g++ test1.cpp -o main
```

- What has happened?

What about this?

```
#include <iostream>

class LivingThing {
public:
    void breathe() {
        printf("I'm breathing as a living thing.\n”);
    }
};

class Animal : public LivingThing {

};

class Reptile : public LivingThing {

};

class Snake : public Animal, public Reptile {
};

int main() {
    Snake snake;
    snake.breathe();
    return 0;
}
```

Does it compile?

```
g++ test2.cpp -o main
```

- What has happened?

How to solve the problem?

```
#include <iostream>

class LivingThing {
public:
    void breathe() {
        printf("I'm breathing as a living thing.\n");
    }
};

class Animal : public virtual LivingThing {

};

class Reptile : public virtual LivingThing {

};

class Snake : public Animal, public Reptile {
};

int main() {
    Snake snake;
    snake.breathe();
    return 0;
}
```