601.220 Intermediate Programming

Function hiding and abstract classes

Function hiding

```
#include<iostream>
using std::cout; using std::endl;

class Base {
public:
   void fun(int i) const { cout << "Base " << i << endl; };
};

class Derived: public Base {
public:
   void fun(char c) const { cout << "Derived " << c << endl; };
};

int main() {
   Derived d;
   d.fun(76);
   d.fun('a');
   return 0;
}
```

```
$ g++ -c hiding.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o hiding hiding.o
$ ./hiding
Derived L
Derived a
```

Function hiding

```
#include<iostream>
using std::cout; using std::endl;

class Base {
public:
   virtual void fun(int i) const { cout << "Base " << i << endl; };
};

class Derived: public Base {
public:
   void fun(char c) const { cout << "Derived " << c << endl; };
};

int main() {
   Derived d;
   d.fun(76);
   d.fun('a');
   return 0;
}
```

```
$ g++ -c hiding2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o hiding2 hiding2.o
$ ./hiding2
Derived L
Derived a
```

Function hiding

```
#include<iostream>
using std::cout; using std::endl;

class Base {
public:
   void fun(int i, int j) const { cout << "Base " << i << j << endl; };
};

class Derived: public Base {
public:
   void fun(char c) const { cout << "Derived " << c << endl; };
};

int main() {
   Derived d;
   d.fun(76, 77);
   return 0;
}
```

```
$ g++ -c hiding3.cpp -std=c++11 -pedantic -Wall -Wextra
hiding3.cpp: In function ‘int main()’:
hiding3.cpp:16:16: error: no matching function for call to ‘Derived::fun(int, int)’
   16 |    d.fun(76, 77);
      |                ^
hiding3.cpp:11:9: note: candidate: ‘void Derived::fun(char) const’
   11 |    void fun(char c) const { cout << "Derived " << c << endl; };
      |         ^~~
hiding3.cpp:11:9: note:   candidate expects 1 argument, 2 provided
```

Function hiding

```
#include<iostream>
using std::cout; using std::endl;

class Base {
public:
   virtual void fun(int i, int j) const { cout << "Base " << i << j << endl; };
};

class Derived: public Base {
public:
   void fun(char c) const { cout << "Derived " << c << endl; };
};

int main() {
   Derived d;
   d.fun(76, 77);
   return 0;
}
```

```
$ g++ -c hiding4.cpp -std=c++11 -pedantic -Wall -Wextra
hiding4.cpp: In function ‘int main()’:
hiding4.cpp:16:16: error: no matching function for call to ‘Derived::fun(int, int)’
   16 |    d.fun(76, 77);
      |                ^
hiding4.cpp:11:9: note: candidate: ‘void Derived::fun(char) const’
   11 |    void fun(char c) const { cout << "Derived " << c << endl; };
      |         ^~~
hiding4.cpp:11:9: note:   candidate expects 1 argument, 2 provided
```

Function hiding

```
#include<iostream>
using std::cout; using std::endl;

class Base {
public:
   void fun(int i, int j) const { cout << "Base " << i << " " <<  j << endl; };
};

class Derived: public Base {
public:
   void fun(char c) const { cout << "Derived " << c << endl; };
};

int main() {
   Derived d;
   d.Base::fun(76, 77);
   return 0;
}
```

```
$ g++ -c hiding5.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o hiding5 hiding5.o
$ ./hiding5
Base 76 77
```

Pure virtual functions

What does the $= 0$ mean here?

```
class Shape {
public:
    virtual double size() const = 0;
    ...
};
```

Pure virtual functions

- Declaring a virtual member function and adding $= 0$ makes it a pure virtual function
- In the memory layout, it means set the address of the virtual function to `nullptr`
- When we declare a pure virtual function:
  * `We do not give it an implementation`
  * `Makes the class it's declared in an abstract class`
  * `We cannot create a new object with the type, though we might be able to create an object from a derived type`

Pure virtual functions

```
class Shape {
public:
   virtual double size() const = 0;
   ...
};

class Shape2D : public Shape {
   ...
};

class Circle : public Shape2D {
public:
   double size() const {
      return 3.14 * r * r;
   }
   ...
private:
   double r;
   ...
};
```

Pure virtual functions

```cpp
#include <iostream>
#include <cmath>

using std::cout;
using std::endl;

class Shape {
public:
   virtual double size() const = 0;
};

class Shape2D : public Shape { };

class Circle : public Shape2D {
public:
   Circle(double radius) : Shape2D(), r(radius) { }
   double size() const {
      return 3.14 * r * r;
   }
private:
   double r;
};

int main() {
   Circle c(1.0 / sqrt(3.14));
   cout << c.size() << endl;
   return 0;
}
```

```
$ g++ -c shape_virt.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o shape_virt shape_virt.o
$ ./shape_virt
1
```

Pure virtual functions

```
#include <iostream>
#include <cmath>

using std::cout;
using std::endl;

class Shape {
public:
   virtual double size() const = 0;
};

class Shape2D : public Shape { };

class Circle : public Shape2D {
public:
   Circle(double radius) : Shape2D(), r(radius) { }
   double size() const {
      return 3.14 * r * r;
   }
private:
   double r;
};

int main() {
   Shape s;
   return 0;
}
```

```
$ g++ -c shape2_virt.cpp -std=c++11 -pedantic -Wall -Wextra
shape2_virt.cpp: In function ‘int main()’:
shape2_virt.cpp:25:10: error: cannot declare variable ‘s’ to be of abstract type ‘Shape’
   25 |    Shape s;
      |          ^
shape2_virt.cpp:7:7: note:   because the following virtual functions are pure within ‘Shape’:
    7 | class Shape {
      |       ^~~~~
shape2_virt.cpp:9:19: note:     ‘virtual double Shape::size() const’
    9 |    virtual double size() const = 0;
      |                   ^~~~
```

Pure virtual functions

```
#include <iostream>
#include <cmath>

using std::cout;
using std::endl;

class Shape {
public:
   virtual double size() const = 0;
};

class Shape2D : public Shape { };

class Circle : public Shape2D {
public:
   Circle(double radius) : Shape2D(), r(radius) { }
   double size() const {
      return 3.14 * r * r;
   }
private:
   double r;
};

int main() {
   Shape2D s2d;
   return 0;
}
```

```
$ g++ -c shape2_virt.cpp -std=c++11 -pedantic -Wall -Wextra
shape2_virt.cpp: In function ‘int main()’:
shape2_virt.cpp:25:12: error: cannot declare variable ‘s2d’ to be of abstract type ‘Shape2D’
   25 |    Shape2D s2d;
      |            ^~~
shape2_virt.cpp:12:7: note:   because the following virtual functions are pure within ‘Shape2D’:
   12 | class Shape2D : public Shape { };
      |       ^~~~~~~
shape2_virt.cpp:9:19: note:     ‘virtual double Shape::size() const’
    9 |    virtual double size() const = 0;
      |                   ^~~~
```

Abstract classes

- "Cannot declare variable `s` to be of abstract type Shape"
- When a class has one or more pure virtual functions, it cannot be instantiated; it is *abstract*

  ```
  * Similar to abstract class and interface in Java
  ```

- The derived Circle class can be instantiated because it provides an implementation for the (only) pure virtual, `size()`
- Another way to make a class abstract is give it only non-public constructors
  - Can't instantiate the abstract class because the constructor can't be called from the outside
  - Derived class can still use protected constructor in the base class

Abstract classes

```
class Piece
{
public:
   ...
protected:
   // This is the only constructor
   Piece(bool is_white): is_white(is_white){ }
   ...
private:
   bool is_white;
};

class Queen: public Piece {
   ...
   // Queen constructor calls Piece constructor
   // OK because it's protected
   Queen( bool is_white ) : Piece( is_white ) { }
   ...
};
```

Facts about abstract class

- A class is abstract if it has at least one pure virtual function or has only non-public constructor
- We can have pointers and references of abstract class type but not concrete objects
- If we do not override the pure virtual function in derived class, then derived class also becomes abstract class
- An abstract class can have constructors

Quiz!

Consider the following classes and partial `main` function:

```
#include <iostream>
using std::cout;

class Op {
public:
  virtual int apply(int a, int b) = 0;
};
class Add {
public:
  virtual int apply(int a, int b)
    { return a + b; }
};
class Mul {
public:
  virtual int apply(int a, int b)
    { return a * b; }
};

int main() {
  Op *a, *b;
  HERE
  cout << a->apply(b->apply(2, 3), 4);
}
```

What code could be added in place of `HERE` to cause the program to print the output 20?

A. `a = new Add(); b = new Add();`

B. `a = new Add(); b = new Mul();`

C. `a = new Mul(); b = new Add();`

D. `a = new Mul(); b = new Mul();`

E. None of the above