C++

程式語言（二）

Introduction to Programming (II)

Pure Virtual Functions
& Abstract Classes

Joseph Chuang-Chieh Lin

Dept. CSE, NTOU

Platform/IDE

- Dev-C++
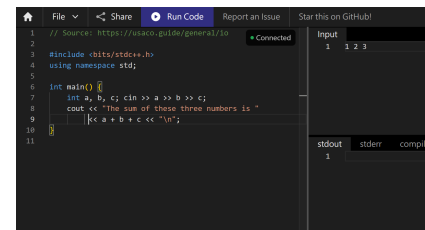

https://www.pngegg.com/en/search?q=Dev-C

- Codeblocks

https://icons8.com/icons/set/code-blocks

- OnlineGDB (https://www.onlinegdb.com/)

- Real-Time Collaborative Online IDE (https://ide.usaco.guide/)

Textbooks (We focusing on C++11)

- ***Learn C++ Programming by Refactoring*** (由重構學習 C++ 程式設計). **Pang-Feng Liu (劉邦鋒). NTU Press. 2023.**

- ***C++ Primer. 5th Edition.*** **Stanley B. Lippman, Josée Lajoie, Barbara E. Moo. 2019.**

- *Effective C++.* Scott Meyers. O'Reilly. 2016.

- *Thinking in C++. Vol. 1: Introducing to Standard C++.* 2nd Edition. Bruce Eckel. Prentice Hall PTR. 2000.

Useful Resources

- Tutorialspoint
  - https://www.tutorialspoint.com/cplusplus/index.htm
  - Online C++ Compiler
- Programiz
  - https://www.programiz.com/cpp-programming
- LEARN C++
  - https://www.learncpp.com/
- MIT OpenCourseWare - Introduction to C++
  - https://ocw.mit.edu/courses/6-096-introduction-to-c-january-iap-2011/pages/lecture-notes/
- Learning C++ Programming
  - https://www.programiz.com/cpp-programming
- GeeksforGeeks
  - https://www.geeksforgeeks.org/c-plus-plus/

Recall & Remark:
Virtual Functions

Virtual Function

- A capability known as polymorphism.
- It resolves the derived version of the function existing between the **base** and **derived** classes. [exists in inheritance]
  - Must have the same signature:
    - Name
    - Parameter
    - Return type
    - `const` or not

Virtual Function (contd.)

- Late binding:
  - The process in which the function call is resolved during **runtime**.
  - The type of object is determined by the compiler at the runtime and the function call is bound.

- Some important rules of virtual functions:
  - A member of some class and should be defined in the **base** class.
  - They are **NOT** allowed to be **static** members.
  - They can be a **friend** of other classes.
  - We can **NOT** have a virtual **constructor** but **we can have a virtual destructor**.

Example

```
#include <iostream>
using namespace std;

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

class Derived_A: public Base {
public:
    virtual void getName() const { cout <<  "Derived A" << endl; }
};

class Derived_B: public Derived_A {
public:
    virtual void getName() const { cout << "Derived B" << endl; }
};

int main() {
    Derived_B derived;
    Base& rBase{ derived };
    cout << "rBase is a ";
    rBase.getName();
    return 0;
}
```

Example

```cpp
class base {
public:
    void fun_1() { cout << "base1\n"; }
    virtual void fun_2() { cout << "base2\n"; }
    virtual void fun_3() { cout << "base3\n"; }
    virtual void fun_4() { cout << "base4\n"; }
};

class derived : public base {
public:
    void fun_1() { cout << "derived1\n"; }
    void fun_2() { cout << "derived2\n"; }
    void fun_4(int x) { cout << "derived4\n"; }
};
```

```cpp
int main() {
    base *p;
    derived obj1;
    p = &obj1;

    // Early binding; fun1() is non-virtual in base
    p->fun_1();

    // Late binding
    p->fun_2();

    // Late binding
    p->fun_3();

    // Late binding
    p->fun_4();

    // Early binding but this function call is
    // illegal because pointer is of base type
    // and function is of derived class
    // p->fun_4(5);

    return 0;
}
```

Pure Virtual Function

- **Implementation** of **all** functions sometimes cannot be provided all at once.

- We would like to **give a base class at first** and leave the actual implementation in the derived class.

- Such a base class is generally an **idea** or **concept**.

- Basically, we **must** **override** the pure virtual function in the derived class.

Pure Virtual Function

- **Pure Virtual Function:**
  - Assigning 0 in the declaration of a virtual function.

- **Abstract class**:
  - A class which has **at least one** pure virtual function.
  - An abstract class **cannot be instantiated**, but pointers (*) and references (&) of an abstract class can be created.
  - For derived classes to use its *interface*.
  - Any derived class of an abstract class **MUST implement ALL pure virtual functions**, otherwise it would become an abstract class, too.
  - We can **NOT** create an object of an abstract class.

Example

```cpp
#include<iostream>
using namespace std;

class B {
   public:
      virtual void s() = 0;
      // Pure Virtual Function
};
```

```cpp
class D : public B {
   public:
      void s() {
         cout << "Virtual Func. in D" << endl;
      }
};

int main() {
   B *b; // What if we use "B b;" ?
   D dobj;
   b = &dobj;
   b->s();
   return 0;
}
```

What if we do not override the pure function in the derived class?

```cpp
#include<iostream>
using namespace std;

class Base {
public:
    virtual void show() = 0;
};

class Derived : public Base { };

int main() {
  Derived d;
  return 0;
}
```

https://onlinegdb.com/6UtgjdG81T

What if we do not override the pure function in the derived class?

```cpp
#include<iostream>
using namespace std;

class Base {
public:
    virtual void show() = 0;
};


class Derived : public Base { };


int main() {
    Derived d;
    return 0;
}
```

```
main.cpp:17:11: error: cannot declare variable 'd' to be of abstract type
'Derived'
   17 |    Derived d;
      |            ^
main.cpp:14:7: note:   because the following virtual functions are pure within
'Derived':
   14 | class Derived : public Base { };
      |       ^~~~~~~
main.cpp:11:18: note:   'virtual void Base::show()'
   11 |     virtual void show() = 0;
      |                  ^~~~
```

What if we do not override the pure function in the derived class?

```cpp
#include<iostream>
using namespace std;

class Base {
public:
    virtual void show();
};

class Derived : public Base { };

int main() {
  Derived d;
  return 0;
}
```

What's the output?

Example

```cpp
#include<iostream>
using namespace std;

class B {
   public:
      virtual void s() = 0;
      // Pure Virtual Function
};
```

```cpp
class D : public B {
   public:
      void s() {
         cout << "Virtual Func. in D" << endl;
      }
};

int main() {
   B *b = new D;
   b->s();
   return 0;
}
```

Example

```cpp
#include<iostream>
using namespace std;

class B {
   public:
      virtual void s() = 0;
      // Pure Virtual Function
};
```

```cpp
class D : public B {
   public:
      void s() {
         cout << "Virtual Func. in D" << endl;
      }
};

int main() {
   B *b = new D;
   B &r = (*b); // what's the outcome?
   r.s();
   return 0;
}
```

Can an abstract class have a constructor?

https://onlinegdb.com/bmaRE5eOM

```cpp
#include<iostream>
using namespace std;

class Base {
protected:
    int x;
public:
    virtual void fun() = 0;
    Base(int i) {
        x = i;
        cout << "Constructor of base called"
             << endl;
    }
    ~Base() = default;
};
```

```cpp
class Derived: public Base {
    int y;
public:
    Derived(int i, int j): Base(i) {
        y = j;
    }
    void fun() { cout << "x = " << x
                      << ", y = " << y
                      << endl; }
};
```

```cpp
int main() {
    Derived d(4, 5);
    d.fun();

    Base *ptr = new Derived(6,7);
    ptr->fun();
    return 0;
}
```

C++ Programming Languages, CSE, NTOU, Taiwan
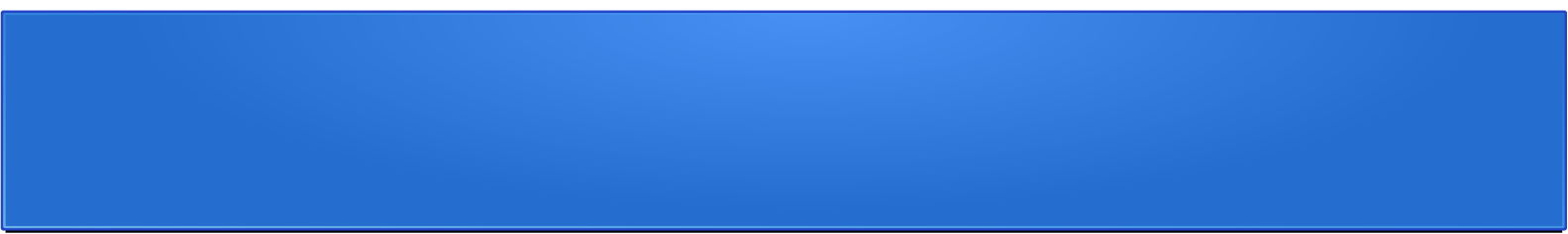
18

Exercise

```cpp
class GuessGame {
public:
    virtual void go() = 0; // play game
    virtual void welcome(string text) = 0;
    // print the welcome message
    virtual void exitGame(string text) = 0;
    // print the ending message
    virtual ~GuessGame() = default;
};
```

```cpp
int main() {
    GuessGame &game = ConsoleGame();
    game.welcome();
    game.go();
    game.exitGame();
    return 0;
}
```

```cpp
class ConsoleGame : public GuessGame {
public:
    void go() {
        /* please implement the number
           guessing game*/
    }
    void welcome(string text) {
    /* please implement this
       welcoming function */
    void exitGame(string text) {
    /* please implement this
       ending function */
    }
};
```

Exercise (contd.)

- The number guessing game in C: **source_code**

  - Please rewrite it in C++ style.

  - Implement the member function `go()` in the derived class `ConsoleGame`.
  - Design your own `void welcome(string text)` and `exitGame` by printing some words or sentences (up to you).

Discussions & Questions