物件導向程式設計

Virtual Functions

Joseph Chuang-Chieh Lin
Dept. CSIE, Tamkang University

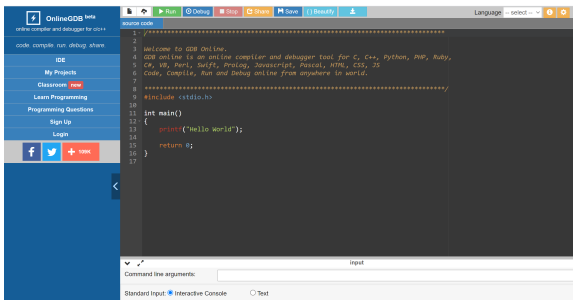
Platform

- Dev-C++

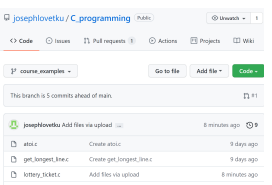
  Click here to download.

  Note: Please use this version otherwise you can't compile your programs/projects in Win10.

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



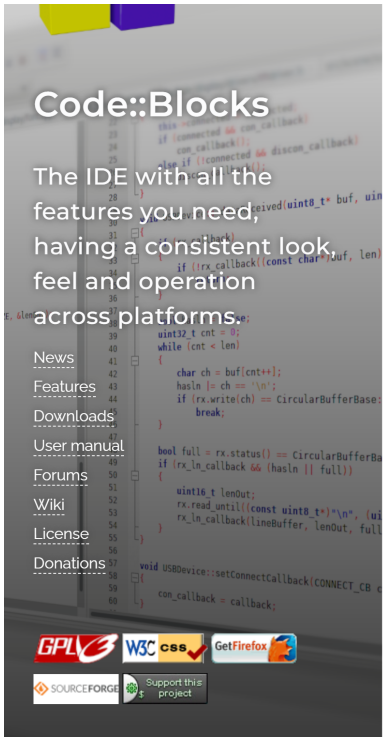
My GitHub page:
click the link here to visit.



- Other resources:
  - MIT OpenCourseWare - Introduction to C++ [link].
  - Learning C++ Programming [Programiz].
  - GeeksforGeeks [link]

Platform/IDE

- https://www.codeblocks.org/



Code::Blocks

Code::Blocks

The free C/C++ and Fortran IDE.

Code::Blocks is a free C/C++ and Fortran IDE built to meet the most demanding needs of its users. It is designed to be very extensible and fully configurable.

Built around a plugin framework, Code::Blocks can be extended with plugins. Any kind of functionality can be added by installing/coding a plugin. For instance, event compiling and debugging functionality is provided by plugins!

If you 're new here, you can read the **user manual** or visit the **Wiki** for documentation. And don't forget to visit and join our **forums** to find help or general discussion about Code::Blocks.

We hope you enjoy using Code::Blocks!

The Code::Blocks Team

Latest news

Migration successful

We are very happy to announce that the process of migrating to the new infrastructure has completed successfully!

Read more

Virtual Function

- A capability known as polymorphism.
- It resolves the derived version of the function existing between the **base** and **derived** classes.
  - 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

https://www.geeksforgeeks.org/virtual-function-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"; }
};
```

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

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

    // Late binding (RTP)
    p->fun_2();

    // Late binding (RTP)
    p->fun_3();

    // Late binding (RTP)
    p->fun_4();

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

    return 0;
}
```

Let’s have a review of “scope” and “Name lookup”.

Name Lookup

```
struct Base {
    int memfcn();
};
struct Derived : Base {
    int memfcn(int); // hides memfcn in the base
};

Derived d;
Base b;

b.memfcn(); // calls Base::memfcn
d.memfcn(10); // calls Derived::memfcn
d.memfcn(); // error: memfcn with no arguments is hidden
d.Base::memfcn(); // ok: calls Base::memfcn
```

Virtual Functions and Scope

```
class Base {
public:
    virtual int fcn();
};
class D1 : public Base {
public:
    // hides fcn in the base; this fcn is not virtual
    // D1 inherits the definition of Base::fcn()
    int fcn(int); // parameter list differs from fcn in Base
    virtual void f2();
                  // new virtual function that does not exist in Base
};
class D2 : public D1 {
public:
    int fcn(int); // nonvirtual function hides D1::fcn(int)
    int fcn(); // overrides virtual fcn from Base
    void f2(); // overrides virtual f2 from D1
};
```

Virtual Functions and Scope

```
Base bobj;
D1 d1obj;
D2 d2obj;
Base *bp1 = &bobj, *bp2 = &d1obj, *bp3 = &d2obj;

bp1->fcn(); // virtual call, will call Base::fcn at run time
bp2->fcn(); // virtual call, will call Base::fcn at run time
bp3->fcn(); // virtual call, will call D2::fcn at run time

D1 *d1p = &d1obj;
D2 *d2p = &d2obj;

bp2->f2(); // error: Base has no member named f2
d1p->f2(); // virtual call, will call D1::f2() at run time
d2p->f2(); // virtual call, will call D2::f2() at run time
```

Override Identifier (from C++ 11)

https://www.programiz.com/cpp-programming/virtual-functions

- **Purpose**: Avoid bugs while using virtual functions.

```
class Base {
   public:
    virtual void print() {
        // code
    }
};

class Derived : public Base {
   public:
    void print() override {
        // code
    }
};
```

```
class Derived : public Base {
   public:
    // function prototype
    void print() override;
};

// function definition
void Derived::print() {
    // code
}
```

Exercise

https://onlinegdb.com/7xszvlC4F_

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

class Animal {
private:
    string type;
public:
    Animal(): type("Animal") {}
    virtual string getType() { /* return the type of animal */ }

};

class Dog : public Animal {
private:
    string type;
public:
    Dog(): type("Dog") {}
    /* override getType() with identifier override */
};

class Cat : public Animal {
private:
    string type;
public:
    Cat(): type("Cat") {}
    /* override getType() with identifier override */
};

int main() {
    Animal* dog = new Dog();
    Animal* cat = new Cat();
    cout << dog->getType() << endl;
    cout << cat->getType() << endl;
    return 0;
}
```

- Please implement the required function `getType()` and its overridden versions.
- Please add destructors in the three classes.
- Please correct the main function using `delete`.

Sample output

```
Dog
Cat
```

Supplement (Static in Classes)

- https://onlinegdb.com/8LsFz0z8i

```
class Base {
private:
    static int count;
public:
    ...
    static int getCounter() {
        return count;
    }
};

int Base::count = 0;
```

Supplement (Static in Classes)

- https://onlinegdb.com/8LsFz0z8i

```
class Base {
private:
    static int count;
public:
    ...
    int getCounter() {
        return count;
    }
};

int Base::count = 0;
```

```
int main() {
    ...
    cout << obj4.getCounter()
         << endl;
    ...
}
```