Язык C++

ООП. Полиморфизм

Полиморфизм

- *свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта*.

Динамический полиморфизм

- Позднее и раннее связывание
- Виртуальные функции

Виртуальные функции

```
class ILogger {
public:
   virtual void Log(const char* message) { }

   virtual ~ILogger() = default;
};

class CConsoleLogger : public ILogger {
public:
  void Log(const char* message) override {
      std::cout << message << std::endl;
  }
};
```

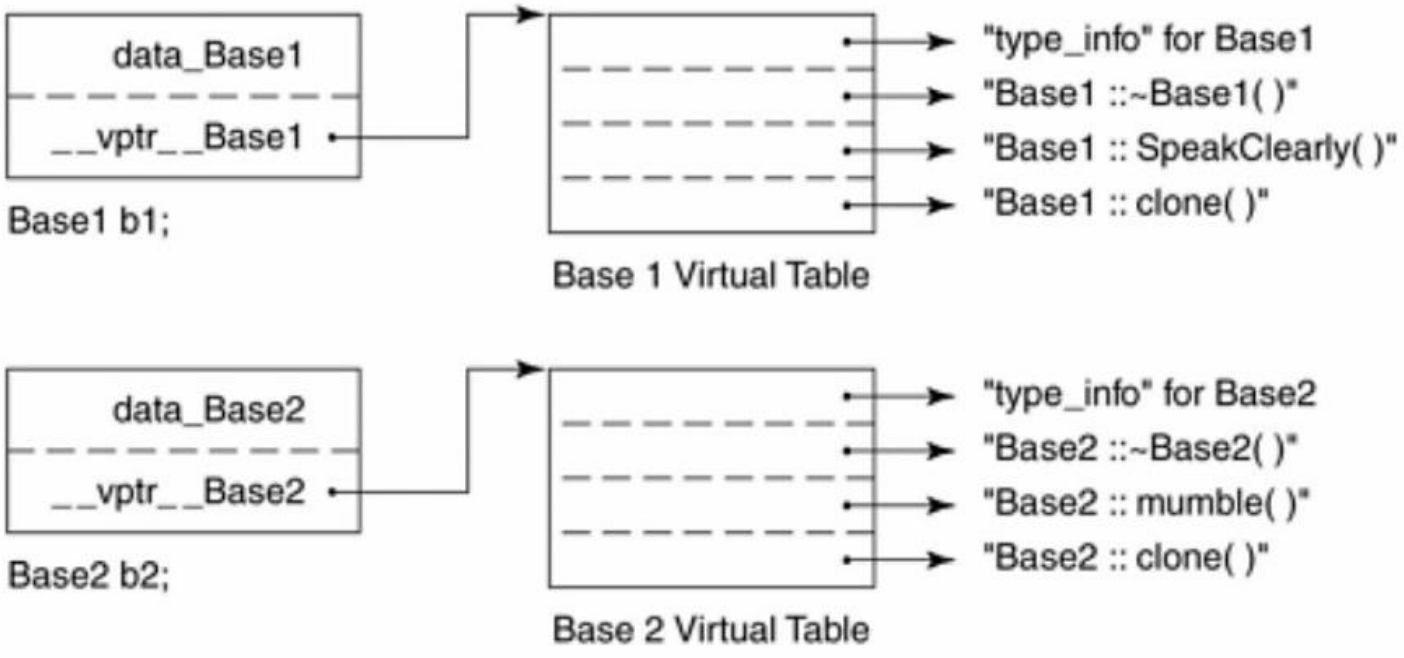
Виртуальные функции

```
class CFileLogger : public ILogger {
public:
   CFileLogger(const char* filename)
       : stream_(filename)
   {}

   ~CFileLogger() {
       stream_.close();
   }

   CFileLogger(const CFileLogger&) = delete;
   CFileLogger& operator=(const CFileLogger&) = delete;

   void Log(const char* message) override {
       stream_ << message << std::endl;
   }
private:
   std::ofstream stream_;
}
```

Таблица виртуальных функций

- Таблица заводится для любого класс с виртуальной функции
- Вызов виртуального метода — это вызов метода по адресу из таблицы

- Стандарт не определяет механизм реализации виртуальных функций, однако большинство компиляторов реализуют именно таблицу вирутальных функций


data_Base1
__vptr__Base1
Base1 b1;
"type_info" for Base1
"Base1 ::~Base1( )"
"Base1 :: SpeakClearly( )"
"Base1 :: clone( )"
Base 1 Virtual Table
data_Base2
__vptr__Base2
Base2 b2;
"type_info" for Base2
"Base2 ::~Base2( )"
"Base2 :: mumble( )"
"Base2 :: clone( )"
Base 2 Virtual Table

override

```
class ILogger {
public:
   virtual void Log(const char* message) {
   }

   virtual ~ILogger() = default;
};

class CConsoleLogger : public ILogger {
public:
  void Log(const char* message) override {
      std::cout << message << std::endl;
  }
};
```

final

```
class CConsoleLogger : public ILogger {
public:
  void Log(const char* message) final {
       std::cout << message << std::endl;
  }
};


class CModernConsoleLogger : public CConsoleLogger {
public:
  void Log(const char* message) override {    // Compile-time error
        std::print("{0}", message);
  }
};
```

Virtual destructor

```
class Base {
public:
   Base() { std::cout << "Base\n"; }
   virtual ~Base() { std::cout << "~Base\n"; }
};

class Derrived : public Base {
public:
   Derrived() { std::cout << "Derrived\n"; }
   ~Derrived() { std::cout << "~Derrived\n";}
};

int main(int, char**) {
   Base * d = new Derrived;
   delete d;

  return 0;
}
```

Абстрактный класс

- класс экземпляр которого не может быть создан
- обычно используется в качестве базового класса
- содержит хотя бы 1 ***pure virtual function*** (чисто виртуальную функцию)

```
class ILogger {
public:
    virtual void Log(const std::string& msg) = 0; // pure virtual function
    virtual ~ILogger() = default;
};

int main(int, char**) {
   ILogger log;  // Error : variable type 'ILogger' is an abstract class
   return 0;
}
```

Коллекции полиморфных объектов

NVI Idiom

Virtual Friend Function Idiom

```
class Base {
public:
   virtual ~Base() = default;
   friend std::ostream& operator<<(std::ostream& stream, const Base& value);
protected:
   virtual void printImpl(std::ostream& stream) const {
       stream << "Base\n";
   }
};

std::ostream& operator<<(std::ostream& stream, const Base& value) {
   value.printImpl(stream);
   return stream;
}
```

Virtual Friend Function Idiom

```
class Derrived : public Base {
protected:
    void printImpl(std::ostream& stream) const override {
        stream << "Derrived\n";
    }
};
```

Стоимость виртуальных функций

- Лишнее обращение к таблице вместо явного адреса
- Не возможно сделать inline optimization
- Для коллекций объектов - они всегда в куче
- Порядок объектов также может влиять на скорость

ООП

- Абстракция
- Инкапсуляция
- Наследование
- Полиформизм