Lec 20 Introduction to C++

1. C++简介

C++由Bjarne Stroustrup于1979年在贝尔实验室开发，是C语言的扩展，增加了面向对象编程（OOP）特性。它广泛应用于系统软件、游戏开发和高性能应用。

关键特性：

- **面向对象编程**：通过类和对象组织代码。
- **高性能**：支持低级内存操作。
- **标准库**：提供丰富的数据结构和算法。

2. 面向对象编程（OOP）

OOP是一种基于“对象”的编程范式，对象包含数据和操作数据的函数。核心概念包括：

- **封装**：将数据和方法封装在一个类中，限制外部访问以保护数据完整性。
- **继承**：允许新类从现有类继承属性和行为，促进代码重用。
- **多态**：允许不同类的对象通过统一接口操作，提供灵活性。

3. C++中的类

类是C++中定义对象的蓝图，包含数据成员和成员函数。与C中的结构体相比，C++的类具有以下特点：

- **访问控制**：支持`private`（私有）、`public`（公有）和`protected`（受保护）访问说明符。
- **成员函数**：类可以定义操作数据的函数。
- **默认访问**：类成员默认是`private`，而C++中的结构体默认是`public`。

与C结构体的区别：

- C中的结构体仅用于数据分组，无访问控制或成员函数。
- C++中的类和结构体功能相似，但类更常用于OOP，结构体通常用于简单数据聚合。C++结构体也可包含成员函数，但这是高级主题。

示例：定义`Student`类

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

```
class Student {
private:
    char name[74];        // 学生姓名
    unsigned long UIN;    // 学生ID
    unsigned int year;    // 年级
    float GPA;            // 平均成绩
public:
    // 构造函数
    Student(char const *name, unsigned int UIN, unsigned int year, float GPA) {
        strcpy(this->name, name);  // 复制姓名
        this->UIN = UIN;          // 设置ID
        this->year = year;        // 设置年级
        this->GPA = GPA;          // 设置GPA
    }
    // 获取姓名
    char const *get_name() {
        return name;
    }
    // 获取GPA
    float get_GPA() {
        return GPA;
    }
};

int main() {
    Student s1("Garfield", 123456, 6, 3.5);
    cout << s1.get_name() << " is an excellent student!" << endl;
    cout << "Their GPA is: " << s1.get_GPA() << endl;
    return 0;
}
```

代码解析：

- **私有成员**：name、UIN、year、GPA只能通过类内函数访问。
- **构造函数**：初始化对象，使用this指针区分成员变量和参数。
- **获取函数**：get_name和get_GPA提供对私有成员的只读访问。
- **主函数**：创建Student对象并调用成员函数显示信息。

4. C++中的基本输入/输出（I/O）

C++使用iostream库处理输入输出，主要对象包括：

- std::cout：用于输出到控制台。

- `std::cin`：用于从用户读取输入。

示例：Hello, World!

```
#include <iostream>
int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

代码解析：

- `#include <iostream>`：引入I/O库。
- `std::cout`：标准输出流对象。
- `<<`：插入运算符，将数据发送到输出流。
- `std::endl`：插入换行符并刷新输出缓冲区。
- **命名空间**：`std`是标准库的命名空间，使用`std::`访问其成员，或使用`using namespace std;`（但在大型项目中不推荐）。

5. C++标识符和命名规范

标识符是程序员为变量、函数、类等命名的名称，需遵循以下规则：

- 由字母、数字和下划线组成。
- 必须以字母或下划线开头。
- 不能是保留关键字（如`int`、`class`）。
- 在其作用域内唯一。

命名规范：

- **变量和函数**：使用驼峰命名法（`camelCase`），如`frequencyCount`、`getName()`。
- **类**：使用帕斯卡命名法（`PascalCase`），如`Student`、`Car`。
- **常量**：全大写加下划线，如`MAX_SIZE`。

6. C与C++的区别

C++在C的基础上增加了许多功能，以下是主要区别：

- **文件扩展**：C使用`.c`，C++使用`.cpp`。
- **编译器**：C使用`gcc`，C++使用`g++`。
- **I/O处理**：C使用`stdio.h`（如`printf`），C++使用`iostream`（如`cout`）。
- **高级特性**：C++支持类、函数重载、默认参数、运算符重载等。

7. 函数重载

函数重载允许定义多个同名函数，但参数列表（数量或类型）不同。编译器根据调用时的参数选择合适的函数。

示例：计算不同形状的体积

```
double volume(float r) {
    return (4.0/3.0) * 3.14159 * r * r * r; // 球体
}
double volume(float r, float h) {
    return 3.14159 * r * r * h; // 圆柱体
}
```

代码解析：

- 两个`volume`函数分别计算球体和圆柱体的体积。
- 编译器根据参数数量（一个或两个）选择正确的函数。
- **注意**：仅靠返回类型不同无法实现重载。

8. 默认参数

函数可以为参数指定默认值，使某些参数可选。

示例：计算BMI（体重指数）

```
float bmi(float ht, float wt, bool si = false) {
    float val = wt / (ht * ht);
    if (si) return val * 10000; // 公制单位
    else return val * 703;      // 英制单位
}
```

代码解析：

- `si`参数默认为`false`，表示英制单位。
- 调用`bmi(1.8, 70, true)`使用公制，`bmi(70, 150)`使用默认英制。
- 默认参数必须从右到左定义。

9. 类、对象和成员访问

- **类**：定义对象的模板。
- **对象**：类的实例，拥有自己的数据副本，但共享成员函数。
- 访问说明符：
  - `private`：仅类内可访问。

- public：外部可访问。
- protected：类及其派生类可访问。

10. 构造函数和析构函数

- 构造函数：对象创建时自动调用，用于初始化成员。
  - 与类同名，无返回类型，可重载。
- 析构函数：对象销毁时自动调用，用于释放资源。
  - 与类同名，前缀~，无参数，无返回值。

示例：Student构造函数

代码解析：

- 构造函数使用this指针初始化成员。
- 析构函数通常用于释放动态分配的内存。

11. this指针

this是一个指向当前对象的指针，在非静态成员函数中可用，用于访问对象的成员。

示例：在Student构造函数中，this->name区分成员变量和参数。

12. 获取器和设置器

- **获取器**：读取私有成员值的函数。
- **设置器**：修改私有成员值的函数，提供控制访问的能力。

示例：Student类的get_name和get_GPA。

13. 运算符重载

运算符重载允许为用户定义类型（如类）重新定义运算符行为。

示例：复数加法

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

class Complex {
    double real, imag;
public:
    Complex(double r, double i) : real(r), imag(i) {}
```

```
    Complex operator+(const Complex& c) {
        return Complex(real + c.real, imag + c.imag);
    }
    void print() { cout << "(" << real << " + " << imag << "i)"; }
};

int main() {
    Complex c1(2, 4);
    Complex c2(3, -5);
    Complex c3 = c1 + c2;
    c3.print(); // 输出：(5 + -1i)
    return 0;
}
```

代码解析：

- Complex类表示复数，real和imag分别存储实部和虚部。
- operator+重载+运算符，返回两个复数相加的结果。
- main函数创建复数对象并使用+运算符计算和。

练习

1. 运算符重载：
    - 为Complex类重载乘法运算符*，实现复数乘法：对于(a + bi)和(c + di)，乘积为((ac - bd) + (ad + bc)i)。
2. 链表实现：
    - 使用类实现单向链表，包含Node类（数据和下一节点指针）和LinkedList类（插入、删除、遍历方法）。