08 Adventures in Functions

引用

引用（Reference）是一种创建别名的方式，允许采用不同的名字访问同一个内存地址。

引用在C++中有很多用途，如函数参数传递、返回值、别名等

引用的主要优点是提高了程序的效率，因为使用引用可以避免不必要的值拷贝

```
int main() {
    int x = 10;
    int& ref_x = x; // 定义一个引用，引用变量x

    std::cout << "x = " << x << std::endl; // 输出 x = 10
    std::cout << "ref_x = " << ref_x << std::endl; // 输出 ref_x = 10

    ref_x = 20; // 通过引用修改x的值
    std::cout << "x = " << x << std::endl; // 输出 x = 20
    std::cout << "ref_x = " << ref_x << std::endl; // 输出 ref_x = 20

    return 0;
}

//////////////////////////////////////
void swap(int& a, int& b);
int main() {
    int x = 10;
    int y = 20;

    swap(x, y);

```

```
    std::cout << "x = " << x << std::endl; // 输出 x = 20
    std::cout << "y = " << y << std::endl; // 输出 y = 10

    return 0;
}

void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}
```

注意

1. 引用不是一个新的变量，而只是一个已存在变量的别名。因此，引用本身不占用额外的内存空间。
2. 常量引用（const reference）是一种特殊类型的引用，它不允许通过引用修改所绑定对象的值。常量引用在函数参数传递中可以避免值拷贝，同时保证实参的值不被修改。
3. C++11及以后的版本引入了右值引用（rvalue reference），用于支持移动语义和完美转发。右值引用用两个`&`符号表示，例如：`int&&`

默认参数

函数声明时为某个参数提供一个默认值。当调用这个函数时，如果没有为该参数提供实参，那么函数会使用默认值作为实参。

简化函数调用，同时提高代码的可读性【最常用的参数值写成了默认值，简化了函数调用】。

```
// 声明一个带有默认参数的函数
void print_message(const std::string& message, int repeat = 1);

int main() {
    // 使用默认参数调用函数
    print_message("Hello, world!");
```

```cpp

    // 提供所有参数调用函数
    print_message("Hello, world!", 3);

    return 0;
}

void print_message(const std::string& message, int repeat) {
    for (int i = 0; i < repeat; ++i) {
        std::cout << message << std::endl;
    }
}
```

函数重载

允许多个具有相同函数名但参数列表不同的函数共存的特性。

函数重载用一种统一的方式处理不同类型或不同数量的参数，从而提高代码的可读性。【代码一致性】

```cpp
// 函数重载：相同的函数名，不同的参数列表
int add(int a, int b);
double add(double a, double b);

int main() {
    int int_result = add(1, 2);
    double double_result = add(1.5, 2.5);

    std::cout << "整数相加结果：" << int_result << std::endl;
    std::cout << "浮点数相加结果：" << double_result << std::endl;

    return 0;
}

// 整数相加的实现
int add(int a, int b) {

```

```
    return a + b;
}

// 浮点数相加的实现
double add(double a, double b) {
    return a + b;
}
```

函数模板

函数模板（Function Template）是C++中一种用于实现泛型编程的特性。

通过函数模板，可以编写通用的代码来处理不同数据类型的情况，而无需为每种数据类型编写单独的函数。降低代码重复，提高代码的可读性。

```
// 定义一个函数模板
template <typename T>
T add(T a, T b);

int main() {
    int int_result = add(1, 2);
    double double_result = add(1.5, 2.5);

    std::cout << "整数相加结果：" << int_result << std::endl;
    std::cout << "浮点数相加结果：" << double_result << std::endl;

    return 0;
}

// 函数模板的实现
template <typename T>
T add(T a, T b) {
    return a + b;
}
```