C++程序设计基础 第五章

前言

本文档由 @ItsJiale 创作，作者博客：https://jiale.domcer.com/，作者依据数学与大数据学院 2024 级大数据教学班的授课重点倾向编写而成。所有内容均为手动逐字录入，其中加上了不少自己的理解与思考，耗费近一周时间精心完成。

此文档旨在助力复习 C++ 程序设计基础，为后续学习数据结构筑牢根基。信计专业的同学，也可参考本文档规划复习内容。需注意，若个人学习过程中存在不同侧重点或对重难点的理解有差异，应以教材内容为准。倘若文档内容存在任何不妥之处，恳请各位读者批评指正。

By：ItsJiale

2025.4.8

第5章 函数

5.1认识函数

函数式模块化程序设计的基础。函数是代码间数据传递和交互的基本方式，因此一个函数通常需要传入参数，并返回一个结果。不同的代码块还可以用同一个名字，即函数可以重载，也就是说同一个函数可以有不同的版本。

5.1.1 定义函数

一个函数的定义由4个要素组成：

1. 返回值类型
2. 函数名
3. 形参列表
4. 函数体

其中，返回值类型放在函数名前，形参列表放在圆括号内，每个形参必须含有一个类型说明符，形参列表可以为空。如果有多个形参，则形参间用逗号隔开。

下面编写一个名为maximun的函数，用来返回两个整数中较大的一个数：

```C++
int maximum(int a, int b){          //a和b为两个int类型形参
int c;                              //用来保存结果
c=a>b ? a : b;                      //三目运算符
return c;                           //返回结果
}

```

5.1.2 调用函数

使用函数名.( )；调用函数

如果函数有形参则在（ ）中填写对应实参的列表，用来初始化形参的值

函数调用的结果为函数的返回值

5.1.3 调用规则

1. 调用处的函数名要和被调函数的函数名一致
2. 实参和形参存在一一对应的关系，即顺序要对应
3. 实参可以是左值或者右值，但形参必须是左值（用于接受形参的值）

- **左值**：指的是可以取地址的表达式，通常是变量。比如 `int a = 5;` 这里的 `a` 就是左值。
- **右值**：指的是不能取地址的表达式，一般是临时值，像函数的返回值或者字面常量。例如 `5` 就是右值。

```cpp
#include <iostream>

// 定义一个名为 square 的函数，形参是左值
int square(int num) {
    return num * num;
}

using namespace std;
int main() {
    int x = 3;  // x 是左值
    int result1 = square(x);  // 实参是左值
    int result2 = square(4);  // 实参是右值
    cout << "Square of x: " << result1 << endl;
    cout << "Square of 4: " << result2 << endl;
    return 0;
}
```

5.1.4 无参列表和void返回类型

如果主调函数不需要向被调函数传递数据，则被调函数的形参列表可以显式或隐式地生命为一个空列表。

`void fun(  ){   ..... }   //隐式定义空形参列表，返回值为void`

`void fun(  ){   .......}   //显式定义空形参列表，返回值为void`

其中，虽然形参列表可以为空，但函数返回值类型说明符不能省略，即便是上面两条语句中的void返回值类型说明符也不能省略

5.1.5 函数声明

和对象的名字一样，函数的名字也必须在使用之前声明。和函数定义不同，函数声明只需要描述函数的返回值类型、名字和形参类型即可，函数体用一个分号代替，例如下面代码是maximum函数的声明：

`int maximum(int a ,int b);`

函数声明包含了描述一个函数所需要的三个要素：返回值类型、函数名和形参类型

5.2 局部对象和全局对象

程序中每一个都有作用域和生命期

5.2.1 存储周期

1. 自动存储周期

定义在函数体或语句块内部的对象（包括函数的形参）具有自动存储周期，即在程序执行到其定义的位置时在内存中创建，离开其作用域时被释放

2. 静态存储周期

定义在函数外面或使用static关键字声明的对象具有静态存储周期，即在程序运行期间，它们始终存在，直到程序结束

3. 动态存储周期

利用运算符new生成的对象具有动态存储周期，可利用运算符delete释放其内存空间，也就是说，它们的存储周期从new操作开始，到delete操作结束

4. 线程存储周期

为了支持并行程序设计，C++11引入了thread_local关键字，利用thread_local创建的对象的存储周期在其所在的线程创建时开始，在线程结束时结束

一个对象的生命期是指其存储周期内可以访问该对象的时间。通常情况下，该时间是指一个对象从产生到消亡的时间

5.2.2 局部对象

在语句块内部定义的对象称为局部对象，包括函数的形参。局部对象仅仅在相应的语句块内部可见，而且还会屏蔽外层作用域中的同名对象。局部对象的生命周期取决于定义的方式。

1. 自动对象

自动对象的生命周期起始于定义语句的执行，结束于作用域的结尾处，因此具有自动存储周期。

自动对象创建时，如果提供了初始值，则利用初始值来初始化对象，比如函数调用时，实参作为初始值来初始化形参；否则，执行默认初始化。对于局部自动内置类型，将不会执行初始化，因此其值是未定义的。

2. 局部静态变量

延长了变量的生存周期，从运行开始到结束仍然存在不会被释放。

一个函数被多次调用时，有时候希望函数体内部定义的某些局部对象具有静态存储周期，此类局部对象的值在每一次函数执行完之后都可以被保存下来，也就意味着，***在下一次调用该函数时，此类对象保留上一次函数调用结束时的值。***关键字static可以用来定义此类型对象，这样的对象称为局部静态对象，例如：

C++

```
int fun () {
int a =0;              //a为局部自动对象
static int b = 0;      //b为局部静态对象   第二次调用时a = 0, b=1
return ++b + ++a;
}
int main( ){
for(int i =0 ; i<3 ; ++i){
cout<<fun () <<endl;
}
}
```

5.2.3 全局对象

定义全局对象

在函数外面定义的对象称为全局对象。全局对象具有静态存储周期，也就是说在程序开始执行时创建，在程序运行结束时消亡。全局对象具有全局作用域，即从定义处到其所在的文件末尾都是可见的，因此也称文件域。

```C++
#include<iostream>
using namespace std;
int sum = 10;              //定义全局对象
int main(  ){
int sum =1 ;               //定义局部对象
cout<<sum<<endl;           //访问局部对象sum，打印输出1
cout<<: : sum<<endl;       //访问全局对象sum，打印输出10
}
```

如果想要在局部对象sum的作用域内访问同名的全局对象，则可以使用全局作用域符

：：访问全局对象

5.3 参数传递

5.3.1 值传递

1. 传值调用

函数接收的是实参值的副本，而非实参本身。

```cpp
#include <iostream>
using namespace std;
// 定义一个函数，使用传值调用
void changeValue(int num) {
    num = 50;
    cout << "函数内部形参 num 的值: " << num << endl;
}

int main() {
    int realNum = 10;
    cout << "调用函数前实参 realNum 的值: " << realNum << endl;

    changeValue(realNum);

    cout << "调用函数后实参 realNum 的值: " << realNum << endl;
    return 0;
}
```

1. **`changeValue` 函数**：它接收一个 `int` 类型的参数 `num`，这是传值调用。在函数内部，把 `num` 的值改成了 `50` 并输出。
2. **`main` 函数**：
   - 定义了一个 `int` 类型的变量 `realNum` 并初始化为 `10`。
   - 输出调用函数前 `realNum` 的值。
   - 调用 `changeValue` 函数，将 `realNum` 作为实参传递给它。
   - 输出调用函数后 `realNum` 的值，你会发现其值仍为 `10` 没有改变。

这就体现了传值调用时，形参的改变不会影响实参。

2. 引用做形参

对形参的改变就是对实参的改变

```cpp
#include <iostream>
using namespace std;
// 函数使用引用作为形参
void modifyValue(int& num) {
    num = 100; // 修改形参的值，实际上就是修改实参的值
    cout << "函数内部形参 num 的值: " << num << endl;
}

int main() {
    int value = 20;
    cout << "调用函数前实参 value 的值: " << value << endl;

    // 调用函数
    modifyValue(value);

    cout << "调用函数后实参 value 的值: " << value << endl;

    return 0;
}
```

- **引用形参函数**：`modifyValue` 函数的参数 `int& num` 表明 `num` 是一个引用。在函数内部修改 `num` 的值，由于它是实参的别名，所以会直接修改实参。
- **主函数部分**：在 `main` 函数里，定义了变量 `value` 并初始化为 `20`，调用 `modifyValue` 函数后，`value` 的值会变为 `100`。

与传值调用相比在函数的形参列表多了“&”

3. 地址传递

如果想通过形参改变实参的值，可以将实参的地址传递给形参，也就是说形参的类型为指针类型，例如：

```cpp
#include <iostream>
using namespace std;
// 定义交换函数，使用指针作为形参
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5;
    int y = 10;

    cout << "交换前: x = " << x << ", y = " << y << endl;

    // 调用交换函数，传递 x 和 y 的地址
    swap(&x, &y);

    cout << "交换后: x = " << x << ", y = " << y << endl;

    return 0;
}
```

1. **`swap` 函数**：参数为 `int` 指针 `a` 和 `b`，函数内利用临时变量和指针解引用操作，交换 `a`、`b` 所指变量的值。
2. **调用方式**：在 `main` 函数中，用取地址符 `&` 获取 `x` 和 `y` 的地址并传入 `swap` 函数，从而让函数能直接修改 `x` 和 `y` 的值。

5.4 返回值类型

5.4.1 无值返回

对于返回值为void类型的函数来说，函数体内部可以没有return语句，函数执行完之后，将自动返回

5.4.2 有值返回

1. 值返回

值返回方法是最简单最安全的返回方法，它通过复制返回值的方式将结果传递给主调函数。

```C++
 int maximum(int a , int b){
return a>b ? a : b ;
}
```

2. 引用或指针返回

对于比较大的对象，可以返回一个引用，该引用只是return语句返回的对象的一个别名，与返回对象指向同一个储存空间：

```C++
const int &maximum(const int &a , const int &b){
return a >b ? a : b;          //返回对象a或b的引用
}
```

5.5函数重载和特殊用途的函数

5.5.1函数重载

把同一作用域下具有相同名字但是不同形参列表的一组函数称为重载函数。

1. 形参类型不同：int、string、float
2. 形参个数不同
3. 形参顺序不同
4. 函数重载与返回值类型无关

5.5.2 默认参数

1. **默认形参值**： `add(int a, int b = 10)` 函数中的 `b` 有默认值 `10` 。调用 `add(5)` 时， `b` 会使用默认值，结果为 `5 + 10 = 15` 。
2. **默认形参位置**：默认形参值必须在形参变量的右边，例如 `add(int a, int b = 10)` 中 `b` 在右边。
3. **默认形参和函数重载**：代码中注释掉的 `add(int a)` 函数会导致错误，因为它和 `add(int a, int b = 10)` 在调用时会产生歧义，具有默认值的形参不能进行函数重载。

5.5.3 内联函数

特点：功能简单，语句较少，调用频繁

方法：在函数名前添加 inline

内联函数，不会发生函数调用，因而也不会产生函数调用的开销

5.6 递归调用

函数嵌套：在别的函数调用其他函数

C++

```cpp
#include<iostream>
using namespace std;
long fac(int n){
    long f;
    if(n<0)
        cout<<"n<0，数据错误"<<endl;
    else if(n==0){
        f==1;
    }
    else{
        f=n*fac(n-1);
    }
    return f;
}
int main(){
    int n;
    n=45;
    long f = fac(n);
    cout<<f<<endl;
    return 0;
}
```