C语言---高级语言：效率非常高

C语言 + 类的概念--->c++前身 带类的C语言 + 面向对象的思想--->C++

C语言：struct

写工具：

1.版本一

2.版本二

一、C++关键字

二、命名空间

1.普通的命名空间

```
namespace N1{
    int a = 10;
    int Add(int left, int right){
        return left + right;
    }
}
```

2.命名空间可以嵌套

```
namespace N2{
    int a = 20;
    int Sub(int left, int right){
        return left - right;
    }
    namespace N3 {
        int a = 30;
```

```
        int Mul(int left, int right) {
            return left * right;
        }
    }
}
```

3.在C++ 工程中，允许定义相同名称的命名空间

```
namespace N1 {
    int b = 40;
    int Div(int left, int right) {
        if(0 == right) {
            exit(1);
        }
        return left / right;
    }
}

int a = 50;

int main() {
    int a = 60;

    printf("%d\n", a);
    // :: 作用域运算符，::a 表示访问的是全局作用域中的变量 a
    printf("%d\n", ::a);
    printf("%d\n", N1::a);
    printf("%d\n", N2::N3::Mul(2, 3));
    return 0;
```

```
}
```

1.普通的命名空间

```
namespace N1{
    int a = 10;
    int Add(int left, int right){
        return left + right;
    }
}

// 相当于是把当前N1这个命名空间中的成员变成了一个全局变量
using N1::a;
// 适用场景：当前命名空间中的部分变量在工程中适用比较频繁
int main() {
    printf("%d\n", N1::a);
    printf("%d\n", a);
}



namespace N1{
    int a = 10;
    int Add(int left, int right){
        return left + right;
    }
}

// N1 这个命名空间中所有的成员在当前这个工程中全部为可见的
```

```
using namespace N1;

int main() {
    printf("%d\n", a);
    printf("%d\n", Add(10, 20));
    return 0;
}
```

三、C++输入输出

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

int main() {
    std::cout << "change world" << std::endl;
    cout << "change world\n";
    // 优点
    cout << 10 << " " << 12.34 << endl;

    int a;
    double d;

    cin >> a;
    cin >> d;
    cout << a << " " << d << endl;

    cin >> a >> d;
    cout << a << " " << d << endl;
    return 0;
```

```
}
```

四、缺省参数

1. 全缺省参数

每一个参数都有缺省值

```
void TestFunc(int a = 1, int b = 2, int c = 3) {
    cout << a << " " << b << " " << c << endl;
}

int main() {
    TestFunc(10, 20, 30);
    TestFunc(10, 20);
    TestFunc(10);
    TestFunc();
    return 0;
}
```

2. 半缺省参数

部分参数带有缺省值

缺省参数只能从右往左依次给出，不能隔着给。

```
void TestFunc(int a, int b = 2, int c = 3) {
    cout << a << " " << b << " " << c << endl;
}

int main() {
    TestFunc(10, 20, 30);
    TestFunc(10, 20);
    TestFunc(10);
    // TestFunc();  没办法通过编译
```

```
    return 0;
}
```

缺省参数一般给在函数声明的位置

五、函数重载

函数重载:是函数的一种特殊情况，C++允许在同一作用域中声明几个功能类似的同名函数，这些同名函数的 形参列表(参数个数 或 类型或 顺序)必须不同，常用来处理实现功能类似数据类型不同的问题

```
// 函数重载了：
```

大前提形成重载的几个函数的作用域一定要相同

函数名字相同，参数列表必须不同

参数列表不同：

1. 参数个数不同
2. 参数类型不同
3. 参数类型的次序不同

是否构成重载与返回值类型无关

```
int Add(int left, int right) {
    return left + right;
}
double Add(double left, double right) {
    return left + right;
}
int main() {
    // 编译器在编译阶段会对实参的类型进行推演，选择合适的函数调用
    Add(1, 2);
    Add(1.0, 2.0);
```

```
    return 0;
}
```

调研C语言中是否支持函数重载？

C语言不支持函数重载：在C语言中，编译器对函数名字的修饰规则：只是在函数名前面加一个下划线。

因为C语言的命名规则非常简单，只是在函数名的前边加一个下划线“_”

验证为什么C++支持函数重载

```
int Add(int left, int right);             // ?Add@@YAHHH@Z
```

HHH=>返回值

类型以及参数列表的类型

```
double Add(double left, double right);  // ?Add@@YAHHH@Z
char Add(char left, char right);        // ?Add@@YADDD@Z
int Add(int left, char right);          // ?Add@@YAHHD@Z
//
int main() {
    Add(1, 2);
    Add(1.0, 3.0);
    Add('1', '4');
    Add(1, '2');
    return 0;
}
```

在函数前边加上一个 extern "C" 编译器就会把C++工程中的某个函数按照C语言的风格进行编译。

extern "C"的功能：在C++的工程中把一个函数按照C语言的风格进行编译

```
// C语言中：函数传参有几种方式？
// 1. 传值
void Swap(int left, int right){
   int tmp = left;
   left = right;
   right = tmp;
}
// 2. 传地址
void Swap(int *pLeft, int *pRight) {
   int tmp = *pLeft;
   *pLeft = *pRight;
   *pRight = tmp;
}
int main() {
   int a = 10;
   int b = 20;
   Swap(a, b);
   Swap(&a, &b);
   return 0;
}
```

六、引用

验证：用引用作为返回值

注意：不能返回函数栈上的空间，因为函数调用结束后，栈上的空间已经还给

系统。

```
int ret = 0;
int& Add(int left, int right){
   // int ret = left + right;
   ret = left + right;
   return ret;
}
int main() {
   int a = 10;
   int b = 20;
   int& c = Add(a, b);
   Add(20, 30);
   cout << c << endl;
   return 0;
}
```

实际上在底层没有任何区别，引用最终被转换成了指针

引用类型实际上是有空间的

```
int& ra = a;  b          const int& cra = a;
int* const pa = &a;      const int *const
```

指针与引用的不同、引用