函数：基础与参数传递

COMP250205：计算机程序设计

李昊

hao.li@xjtu.edu.cn

西安交通大学计算机学院

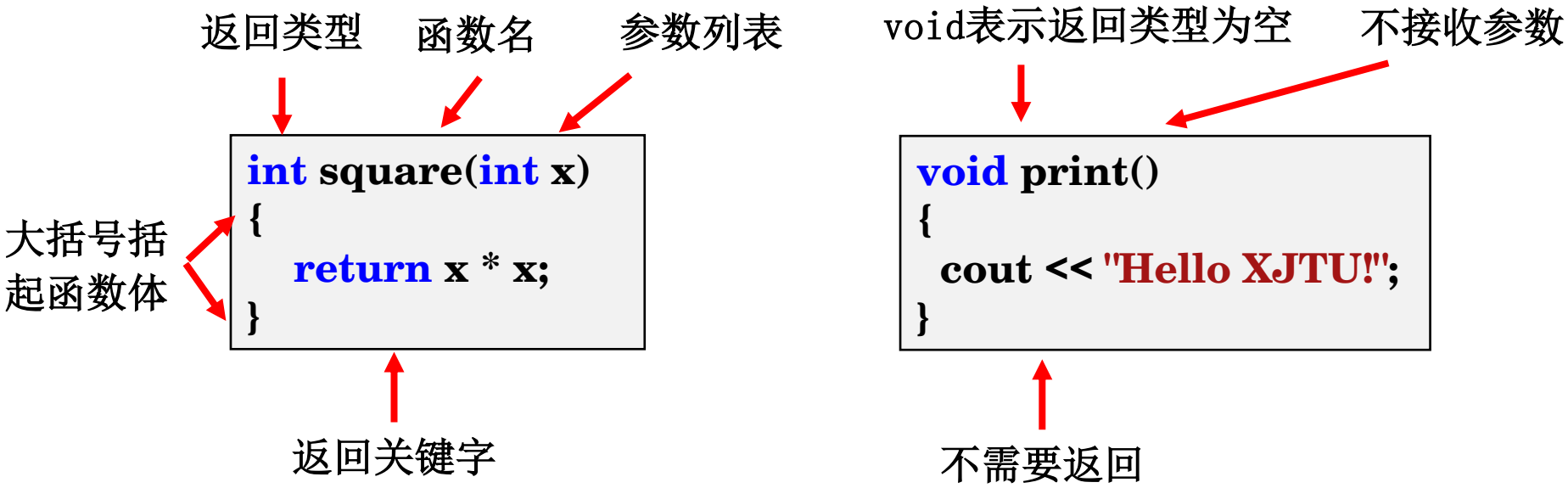
函数基础

函数：C++的基础编程模块

命名的代码块，通过调用函数执行代码

接收0个或多个参数（输入）

通常产生一个结果（输出）



函数的定义、调用和原型

定义：包含函数名、形式参数（形参）和实现

调用：包含函数名和实际参数（实参）

原型：包含函数名和形参

函数原型；可以仅定义形参类型

函数完整定义

函数调用

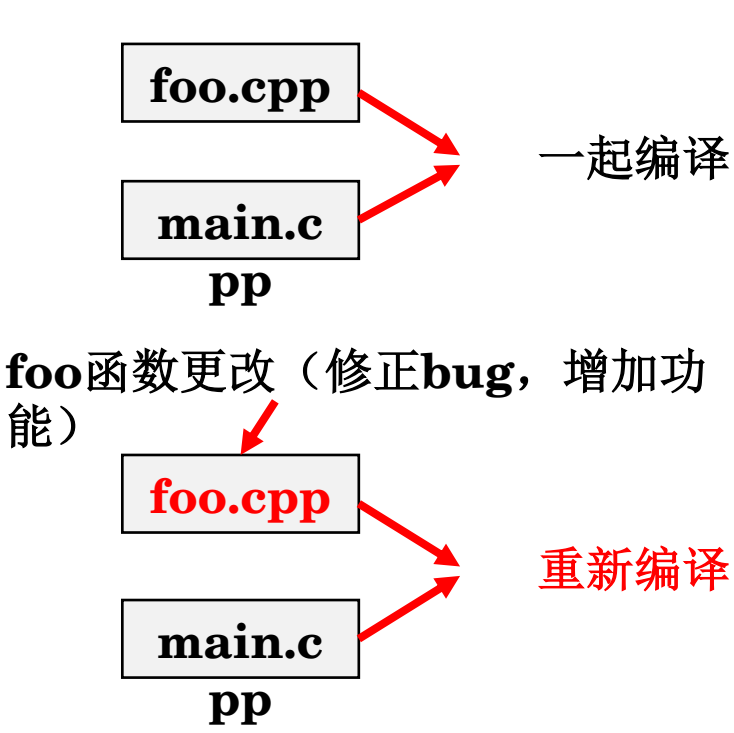
对应函数原型的函数定义

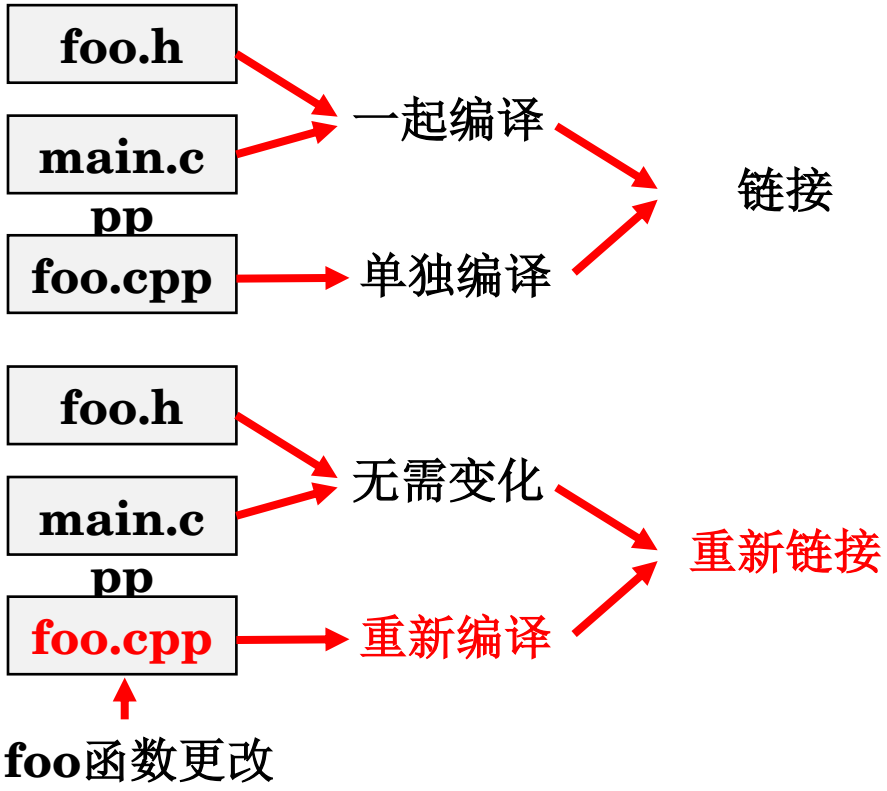
```
void foo(int);
int bar(int a) {return a+1;}

int main()
{
    int i = bar(5);
    foo(1);
    cout << i << j;
    return 0;
}
void foo(int i) {cout << i;}
```

函数原型的意义

调用函数之前必须确认函数名、参数和返回类型

为什么不直接把函数定义写在前面？





分离式编译

程序存储在不同文件中；分别编译，一起链接

头文件：经常以.h结尾，一般仅包含原型

使用#include方法将头文件包含进.cpp文件

分离编译：使用参数，编译生成不可直接执行的文件（对象文件）

链接：链接多个对象文件，生成可执行文件

形式参数与实际参数

定义函数时的参数为形参

调用函数时传递的参数为实参

每次调用函数，形参被自动创建（局部自动变量）

函数原型；仅定义了形参类型

形参为整型a

实参为立即数常量5

实参为整型变量i

形参为整型i

```
void foo(int);
int bar(int a) {return a+1;}

int main()
{
    int i = bar(5);
    foo(i);
    cout << i << j;
    return 0;
}
void foo(int i) {cout << i;}
```

参数传递：按值传递

按值传递参数（传值参数）

按值传递：使用实参来初始化形参变量（拷贝）

形参是一个非引用类型的变量时，均按值传递

形参n和形参m →

修改了局部变量n →

实参num和count →

局部变量n不会影响实参num →

```
void display(int n, int m)
{
    cout << n << " ";
    for (int i=0; i<m; ++i) {
        cout << --n << " ";
    }
}
int main()
{
    int num = 5, count = 5;
    display(num, count);
    cout << num;
    return 0;
}
```

指针形参 - 1

指针不是引用类型变量：指针形参也按值传递

ptr = p; *ptr为5

修改ptr为局部变量value的地

value为10；因此*ptr为10

ptr是p的拷贝,对ptr的任何修改不影响p，也不影响a

```
void modify_ptr(int *ptr, int value) {
    cout << "before: " << *ptr << endl;
    ptr = &value;
    cout << "after: " << *ptr << endl;
}

int main() {
    int a = 5, b = 10;
    int *p = &a;
    modify_ptr(p, 10);
    cout << a << " " << *p << endl;
    return 0;
}
```

指针形参 - 2

指针不是引用类型变量：指针形参也按值传递

ptr = p; *ptr为5

修改*ptr，实际修改了a的值

value为10；因此*ptr为10

a的值被修改了

*p的值也被修改了

```
void modify_value(int *ptr, int value)
{
    cout << "before: " << *ptr << endl;
    *ptr = value;
    cout << "after: " << *ptr << endl;
}

int main() {
    int a = 5, b = 10;
    int *p = &a;
    modify_value(p, 10);
    cout << a << " " << *p << endl;
    return 0;
}
```

指针形参 - QUIZ

编写一个函数，使用指针形参交换两个整数的值

数组参数 - 1

数组是一组数据，无法直接按值传递所有数据

经典组合：传递数组名和数组长度

使用首地址访问数组元素

返回累加值

在函数定义/原型中，
int a[]等价于int *a

传递数组名（首地址）和长度

```
int sum_arr(int arr[], int n) {
    int total = 0;
    for (int i=0; i<n; ++i) {
        total += arr[i];
    }
    return total;
}
int main()
{
    int cookies[] = {1, 2, 4, 8, 16};
    int sum = sum_arr(cookies, 5);
    cout << sum << endl;
    return 0;
}
```

数组参数 - 2

sizeof只能返回指针大小

可以使用范围for循环吗？

sizeof返回数组长度

不同的首地址“欺骗”函数

```
int sum_arr(int arr[], int n) {
    int total = 0;
    cout << “arr size: ” << sizeof(arr);
    for (int i=0; i<n; ++i) {
        total += arr[i];
    }
    return total;
}
int main()
{
    int cookies[] = {1, 2, 4, 8, 16};
    cout << "cookie size: " << sizeof(cookies);
    int sum1 = sum_arr(cookies, 2);
    int sum2 = sum_arr(cookies+2, 2);
    cout << sum << endl;
    return 0;
}
```

数组参数 - 3

经典组合：传递数组名和数组长度

另一个经典组合：传递起始和结束位置

传递起始和结束指针 →

在起始和结束区间内循环 →

给出起始和结束区间 →

```
int sum_arr(int *begin, int *end) {
    int total = 0;
    for (int *i=begin; i!=end; ++i) {
        total += *i;
    }
    return total;
}
int main() {
    int cookies[] = {1, 2, 4, 8, 16};
    int sum = sum_arr(cookies,
cookies+5);
    cout << sum << endl;
    return 0;
}
```

数组参数 - 4

STL容器对象：不是一组数据的集合，而是一个独立的对象；因此可以直接按值传递

vec为vector<int>局部变量
由实参复制而来

对vec的操作不影响cookies

直接传递vector对象

```
int sum_vec(vector<int> vec) {
    int total = 0;
    for (int v : vec) {
        total += v;
    }
    vec.clear();
    return total;
}
int main()
{
    vector<int> cookies = {1, 2, 4, 8, 16};
    cout << sum_vec(cookies) << endl;
    return 0;
}
```

数组参数 - 5

通过迭代器传递一个区间

```
int sum_vec(vector<int>::iterator b, vector<int>::iterator e) {
    int total = 0;
    for (auto i=b; i!=e; ++i) {
        total += *i;
    }
    return total;
}
int main()
{
    vector<int> cookies = {1, 2, 4, 8, 16};
    cout << sum_vec(cookies.begin(), cookies.end()) << endl;
    return 0;
}
```

二维数组参数 - 1

二维数组参数：参数是指针的指针

难点：需要正确的声明指针类型

二维数组 →

sum函数的原型是什么？ →

```
int main()
{
    int data[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {2, 4, 6, 8}
    };
    int total = sum(data, 3);
    cout << total << endl;
    return 0;
}
```

二维数组参数 - 2

二维数组名是数组的指针

data是一个长度为3的数组的指针

或int (*data)[4]

只能接收长度为4的数组的指针

```
int sum(int data[][4], int count)
{
    int total = 0;
    for (int i=0; i<count; ++i) {
        for (int j=0; j<4; ++j) {
            total += data[i][j]
        }
    }
    return total;
}
int main() {
    int data[3][4] = {...};
    int total = sum(data, 3);
    cout << total << endl;
    return 0;
}
```

参数传递：按引用传递

传引用参数 - 1

将形参声明为引用类型；实参直接传递变量

形参将会被初始化为实参的引用

vec为vector<int>的引用
无需拷贝整个vector值

对vec的操作影响cookies

```
int sum_vec(vector<int>& vec) {
    int total = 0;
    for (int v : vec) {
        total += v;
    }
    vec.clear();
    return total;
}
int main()
{
    vector<int> cookies = {1, 2, 4, 8, 16};
    cout << sum_vec(cookies) << endl;
    return 0;
}
```

传引用参数 - 2

按引用传参无需拷贝实参

n被声明为引用类型
实际上执行了 int &n = num;

修改了引用n，实际上也修改了num

实参num和count

```
void display(int &n, int m)
{
   cout << n << " ";
   for (int i=0; i<m; ++i) {
      cout << --n << " ";
   }
}
int main()
{
   int num = 5, count = 5;
   display(num, count);
   cout << num;
   return 0;
}
```

使用引用/指针返回额外信息

一个函数只有一个返回值：返回结果或状态

额外信息可以使用指针或引用进行返回

使用occurs的引用来计数

实际上外部传进来的occurs实参也已经修改

返回最后一次出现的位置

```
unsigned find_char(string &s, char c,
                   unsigned &occurs)
{
    unsigned ret = -1, pos = 0;
    occurs = 0;
    for (char ch : s) {
        if (ch == c) {
            ret = pos;
            occurs++;
        }
        ++pos;
    }
    return ret;
}
```

内容总结

函数基础

- 接收参数，返回值
- 函数定义、函数原型和函数调用
- 形参与实参

函数参数传递

- 按值传递：使用实参初始化形参，局部自动变量
- 按引用传递：将形参初始化为实参的引用