02 - 函数 II

C++ 程序设计进阶

SOJ 信息学竞赛教练组

2024 年 7 月 21 日

目录

函数的定义

- 函数的定义形式如下：

```
1 void 函数名(参数列表) {
2     函数体
3 }
```

函数的定义

- 函数的定义形式如下：

```
1 void 函数名(参数列表) {
2     函数体
3 }
```

- 需要写在 `main` **函数外面**

函数的定义

- 函数的定义形式如下：

```
1 void 函数名(参数列表) {
2     函数体
3 }
```

- 需要写在 `main` **函数外面**
- 确定函数的**功能**，把功能的实现写在函数体中

函数的调用

- 形式参数、实际参数

```cpp
#include <iostream>

using namespace std;

void print(int n) {
  for (int i = 1; i <= n; i++) {
    cout << "hello world" << endl;
  }
}

int main() {
  print(5);
  return 0;
}
```

函数的调用

- 形式参数、实际参数

```cpp
#include <iostream>

using namespace std;

void print(int n) {
  for (int i = 1; i <= n; i++) {
    cout << "hello world" << endl;
  }
}

int main() {
  print(5);
  return 0;
}
```

形式参数

函数的调用

- 形式参数、实际参数

```cpp
#include <iostream>

using namespace std;

void print(int n) {
  for (int i = 1; i <= n; i++) {
    cout << "hello world" << endl;
  }
}

int main() {
  print(5);
  return 0;
}
```

形式参数

实际参数

函数的调用

- 形式参数、实际参数
- 调用函数

函数的调用

- 形式参数、实际参数
- 调用函数
  - 函数名(实参1，实参2，...);

函数的调用

- 形式参数、实际参数
- 调用函数
  - 函数名(实参1，实参2，...);
  - 无参数时也要保留小括号

函数的调用

- 形式参数、实际参数
- 调用函数
  - 函数名(实参1，实参2，...);
  - 无参数时也要保留小括号
  - 注意参数顺序要对应

函数的调用

- 形式参数、实际参数
- 调用函数
  - 函数名(实参1，实参2，...);
  - 无参数时也要保留小括号
  - 注意参数顺序要对应
  - 求 2 的 10 次方：pow(10,2)

函数的调用

- 形式参数、实际参数
- 调用函数
  - 函数名(实参1，实参2，...);
  - 无参数时也要保留小括号
  - 注意参数顺序要对应
  - 求 2 的 10 次方： ~~pow(10,2)~~ pow(2,10)

函数的调用

- 调用无返回值的函数
  - 直接调用
- 调用有返回值的函数
  - 处理返回的结果，直接使用或存储后使用

目录

函数的使用场景

什么时候使用函数?

函数的使用场景

- 多次使用相同功能

函数的使用场景

- 多次使用相同功能
  - 五边形面积

函数的使用场景

- 多次使用相同功能
  - 五边形面积
- 复杂的逻辑嵌套

函数的使用场景

- 多次使用相同功能
  - 五边形面积
- 复杂的逻辑嵌套
  - 质数判断

函数的使用场景

- 多次使用相同功能
  - 五边形面积
- 复杂的逻辑嵌套
  - 质数判断

模块化编程

如何设计一个函数?

模块化编程

- 将一个大程序按照功能划分为若干小程序模块

模块化编程

- 将一个大程序按照功能划分为若干小程序模块
- 每个小程序模块完成一个确定的功能

模块化编程

- 将一个大程序按照功能划分为若干小程序模块
- 每个小程序模块完成一个确定的功能
- 模块之间建立必要的联系，通过模块的互相协作完成整个程序

例 2.1：五边形面积

编程题

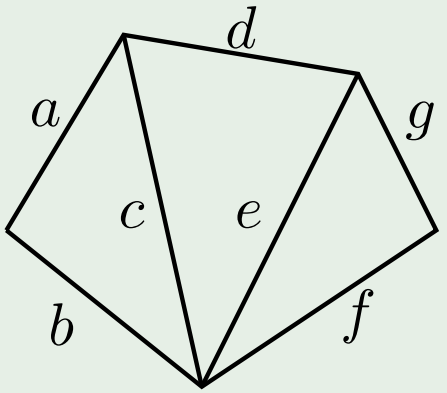
- 如图所示的五边形，输入其 5 条边和 2 条对角线的边长，求这个五边形的面积。
  根据海伦公式，边长分别为 a, b, c 的三角形面积为 $\sqrt{p(p-a)(p-b)(p-c)}$，其中 $p = \frac{a+b+c}{2}$。
  输出结果保留至小数点后两位。

- 样例输入

```
3 4 5 6 7 8 9
```

- 样例输出

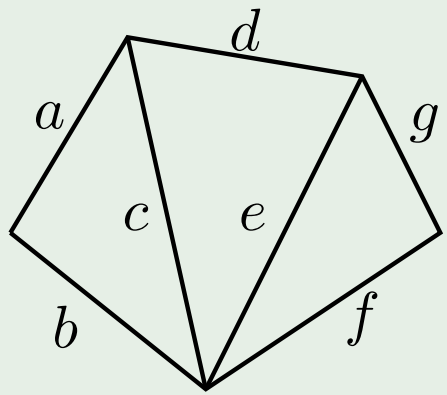
```
47.53
```



例 2.1：五边形面积

问题分析

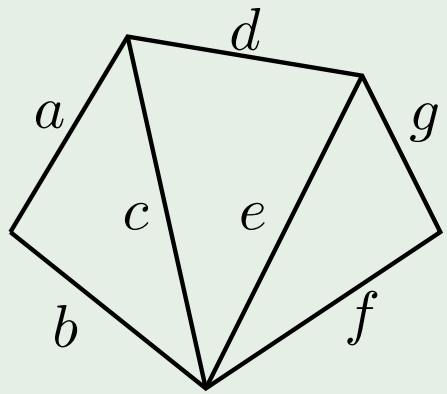
- 一个五边形可以分割为三个三角形，因此五边形的面积等于三个三角形的面积之和
- 三角形的面积可以套用海伦公式求解



例 2.1：五边形面积

细分任务

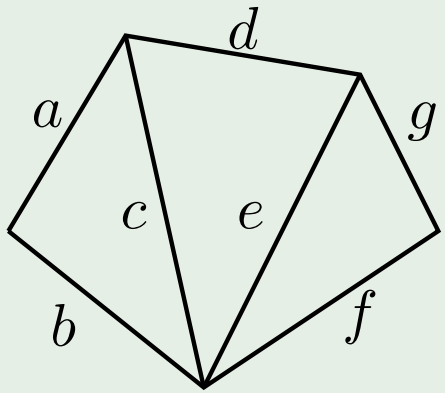
- 将求解五边形面积这个任务细分为多个子任务，每个子任务为一个模块
  - 输入数据
  - 计算边长为 a, b, c 三角形的面积
  - 计算边长为 c, d, e 三角形的面积
  - 计算边长为 e, f, g 三角形的面积
  - 计算五边形面积并输出



例 2.1：五边形面积

细分任务

- 将求解五边形面积这个任务细分为多个子任务，每个子任务为一个模块
  - 输入数据
  - 计算边长为 a, b, c 三角形的面积
  - 计算边长为 c, d, e 三角形的面积
  - 计算边长为 e, f, g 三角形的面积
  - 计算五边形面积并输出



功能相同的多个子任务，定义一个函数，重复调用即可

例 2.1：五边形面积

函数解决子任务

- 重复的子任务：计算三角形面积
- 函数的定义
  - 功能：返回边长分别为 a, b, c 的三角形面积
  - 参数：根据题目输入的数据类型和大小确定参数为 `int` 类型
  - 返回值：三角形面积可能是浮点数，返回值为 `double` 类型

例 2.1：五边形面积

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

double triArea(int a, int b, int c) {
  double p = (a + b + c) / 2.0;
  return sqrt(p * (p - a) * (p - b) * (p - c));
}

int main() {
  int a, b, c, d, e, f, g;
  cin >> a >> b >> c >> d >> e >> f >> g;
  double s1 = triArea(a, b, c);
  double s2 = triArea(c, d, e);
  double s3 = triArea(e, f, g);
  cout << fixed << setprecision(2) << s1 + s2 + s3 << endl;
  return 0;
}
```

例 2.2：质数判断

编程题

- 输入一个正整数 n ($1 \leq n \leq 2 \times 10^9$)，判断 n 是否为质数。若 x 是质数，输出 yes；否则输出 no。
- 样例输入

  3
- 样例输出

  yes

例 2.2：质数判断

质数基本概念

- 整除：指整数 a 除以整数 b ($b \neq 0$) 的余数为 0
- 因数（约数）：如果 a 除以 b 能够整除，则 b 是 a 的因数（约数）
- 倍数：如果 a 除以 b 能够整除，则 a 是 b 的倍数
- 质数（素数）：指在大于 1 的自然数中，除了 1 和它本身以外不再有其他因数的自然数

例 2.2：质数判断

- 判断 n 有没有 $2 \sim n-1$ 范围的因数

```
1 bool isPrime(int n) {
2   for (int i = 2; i < n; i++) {
3     if (n % i == 0) return false;
4   }
5   return true;
6 }
```

例 2.2：质数判断

- 判断 n 有没有 $2 \sim n-1$ 范围的因数

```
1 bool isPrime(int n) {
2   for (int i = 2; i < n; i++) {
3     if (n % i == 0) return false;
4   }
5   return true;
6 }
```

是否正确?

例 2.2：质数判断

- 判断 n 有没有 $2 \sim n-1$ 范围的因数

```
1 bool isPrime(int n) {
2   if (n <= 1) return false;
3   for (int i = 2; i < n; i++) {
4     if (n % i == 0) return false;
5   }
6   return true;
7 }
```

例 2.2：质数判断

质数判定的优化

- 观察一个数字除了 1 和它本身之外的因数，例如 36
  - $2, 3, 4, 6, 9, 12, 18$
  - 因数是成对存在的：$2 \times 18 = 36$, $3 \times 12 = 36$, ...
- 成对的两个因数，其中一个 $\leq \sqrt{n}$，一个 $\geq \sqrt{n}$
- 如果 n 不存在 $\leq \sqrt{n}$ 的因数，肯定也不存在 $\geq \sqrt{n}$ 的因数
- 所以判定质数代码中的 i 只需要枚举到 $\sqrt{n}$，不用枚举到 $n-1$

例 2.2：质数判断

```cpp
#include <iostream>

using namespace std;

bool isPrime(int n) {
  if (n <= 1) return false;

  // i * i <= n 相当于 i <= sqrt(n)
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) return false;
  }
  return true;
}

int main() {
  int n;
  cin >> n;
  if (isPrime(n)) cout << "yes" << endl;
  else cout << "no" << endl;
  return 0;
}
```

例 2.2：质数判断

```cpp
#include <iostream>

using namespace std;

bool isPrime(int n) {
  if (n <= 1) return false;

  // i * i <= n 相当于 i <= sqrt(n)
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) return false;
  }
  return true;
}

int main() {
  int n;
  cin >> n;
  if (isPrime(n)) cout << "yes" << endl;
  else cout << "no" << endl;
  return 0;
}
```

- 时间复杂度 $O(\sqrt{n})$

例 2.2：质数判断

```cpp
#include <iostream>

using namespace std;

bool isPrime(int n) {
  if (n <= 1) return false;

  // i * i <= n 相当于 i <= sqrt(n)
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) return false;
  }
  return true;
}

int main() {
  int n;
  cin >> n;
  if (isPrime(n)) cout << "yes" << endl;
  else cout << "no" << endl;
  return 0;
}
```

- 时间复杂度 $O(\sqrt{n})$

如果 n 是 long long 类型，怎么修改?

使用函数的优点

- 将功能封装到函数中，提高代码的可读性
- 提高代码的复用性，使代码简洁
- 将代码分解成更小的块（模块化），使代码逻辑简单明了，易于查错

目录

变量的作用域

- 代码中的变量并不是在任何位置都可以使用
- 一个变量可用的代码范围就是这个变量的作用域
- 根据作用域的不同，可以把变量分为局部变量和全局变量

局部变量

- 声明在函数内部的变量
- 作用域：从变量的声明开始到包含它的块结束
  - 块是用一对大括号括起来的代码区域
  - 循环、分支语句（包含条件语句）视作一个块
  - 函数（包含参数列表）也视作一个块

局部变量

```
1  // #include ...
2
3  int main() {
4    int n;
5    if (...) {
6      int a;
7      ...
8    }
9    for (int i = ...) {
10     ...
11   }
12   cout << i << endl;
13   return 0;
14 }
```

- 这是一个“块”，变量 n 只能在这个“块”中使用

局部变量

```
1  // #include ...
2
3  int main() {
4    int n;
5    if (...) {
6      int a;
7      ...
8    }
9    for (int i = ...) {
10     ...
11   }
12   cout << i << endl;
13   return 0;
14 }
```

- 这是一个“块”，变量 a 只能在这个“块”中使用

局部变量

```
1  // #include ...
2
3  int main() {
4    int n;
5    if (...) {
6      int a;
7      ...
8    }
9    for (int i = ...) {
10     ...
11   }
12   cout << i << endl;
13   return 0;
14 }
```

- 这是一个“块”，变量 i 只能在这个“块”中使用

局部变量

```
1  // #include ...
2
3  int main() {
4    int n;
5    if (...) {
6      int a;
7      ...
8    }
9    for (int i = ...) {
10     ...
11   }
12   cout << i << endl;
13   return 0;
14 }
```

- 编译错误：此处的 i 未声明

局部变量

- 函数（包含参数列表）也视作一个块

```
1 void sayHello(int n) {
2   for (int i = 1; i <= n; i++) {
3     cout << "hello world" << endl;
4   }
5 }
```

局部变量

- 函数（包含参数列表）也视作一个块

```
1 void sayHello(int n) {
2   for (int i = 1; i <= n; i++) {
3     cout << "hello world" << endl;
4   }
5 }
```

- 这是一个“块”，变量 n 只能在这个“块”中使用

全局变量

- 定义在所有函数外部、没有被任何大括号包含起来的变量
- 作用域：从变量的声明开始一直到代码结束，作用域中任意的函数都可以使用该变量

全局变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  int a = 1;
6
7  void f() {
8    cout << "f(): " << a << endl;
9  }
10
11 int main() {
12   cout << "main(): " << a << endl;
13   f();
14
15
16   return 0;
17 }
```

- 输出

全局变量

```cpp
#include <iostream>

using namespace std;

int a = 1;

void f() {
  cout << "f(): " << a << endl;
}

int main() {
  cout << "main(): " << a << endl;
  f();


  return 0;
}
```

- 输出

  ```
  main(): 1
  ```

全局变量

```cpp
#include <iostream>

using namespace std;

int a = 1;

void f() {
  cout << "f(): " << a << endl;
}

int main() {
  cout << "main(): " << a << endl;
  f();


  return 0;
}
```

- 输出

```
main(): 1
f(): 1
```

全局变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  int a = 1;
6
7  void f() {
8    cout << "f(): " << a << endl;
9  }
10
11 int main() {
12   cout << "main(): " << a << endl;
13   f();
14   int a = 2;
15   cout << "main(): " << a << endl;
16   return 0;
17 }
```

全局变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  int a = 1;
6
7  void f() {
8    cout << "f(): " << a << endl;
9  }
10
11 int main() {
12   cout << "main(): " << a << endl;
13   f();
14   int a = 2;
15   cout << "main(): " << a << endl;
16   return 0;
17 }
```

- 能成功编译吗?

目录

同名变量

- 在同一个“块”里面声明的同名变量冲突
- 在不同“块”里面声明的同名变量不冲突
  - 完全覆盖
  - 完全不覆盖

同名变量

- 输出 3 行 4 列的星号 *

```
1 for (int i = 1; i <= 3; i++) {
2   for (int i = 1; i <= 4; i++) {
3     cout << "*";
4   }
5   cout << endl;
6 }
```

同名变量

- 输出 3 行 4 列的星号 *

```
1 for (int i = 1; i <= 3; i++) {
2   for (int i = 1; i <= 4; i++) {
3     cout << "*";
4   }
5   cout << endl;
6 }
```

同名变量

- 输出 3 行 4 列的星号 *

```
1 for (int i = 1; i <= 3; i++) {
2   for (int i = 1; i <= 4; i++) {
3     cout << "*";
4   }
5   cout << endl;
6 }
```

- 第一层 for 循环完全覆盖第二层 for 循环

同名变量

- 输出 3 行 4 列的星号 *

```
1 for (int i = 1; i <= 3; i++) {
2   for (int i = 1; i <= 4; i++) {
3     cout << "*";
4   }
5   cout << endl;
6 }
```

- 第一层 for 循环完全覆盖第二层 for 循环
- i 的声明屏蔽了外部的 i, 若此时使用 i, 则使用的是内部的 i
  容易混淆两个 i 的使用

同名变量

- 查询第一个 x 所在位置

```
1  #include <iostream>
2
3  using namespace std;
4
5  int a[110];
6
7  int main() {
8    int n;
9    cin >> n;
10   for (int i = 1; i <= n; i++) cin >> a[i];
11   int x;
12   cin >> x;
13   for (int i = 1; i <= n; i++) {
14     if (a[i] == x) {
15       cout << i << endl;
16       break;
17     }
18   }
19   return 0;
20 }
```

同名变量

- 查询第一个 x 所在位置

```
1  #include <iostream>
2
3  using namespace std;
4
5  int a[110];
6
7  int main() {
8    int n;
9    cin >> n;
10   for (int i = 1; i <= n; i++) cin >> a[i];
11   int x;
12   cin >> x;
13   for (int i = 1; i <= n; i++) {
14     if (a[i] == x) {
15       cout << i << endl;
16       break;
17     }
18   }
19   return 0;
20 }
```

两个“块”完全不重叠, 变量 i 互不影响

同名变量

- 函数：输出 n 行 hello world

```
1  #include <iostream>
2
3  using namespace std;
4
5  void sayHello(int n) {
6    for (int i = 1; i <= n; i++) {
7      cout << "hello, world" << endl;
8    }
9  }
10
11 int main() {
12   int n;
13   cin >> n;
14   sayHello(n);
15   return 0;
16 }
```

同名变量

- 函数：输出 $1 \sim n$ 的和

```cpp
// #include ...

int getSum(int n) {
  int res = 0;
  for (int i = 1; i <= n; i++) {
    res += i;
  }
  return res;
}

int main() {
  int n;
  cin >> n;
  cout << getSum(n) << endl;
  return 0;
}
```

引用变量

两个变量可以用相同名称，
那么同一个变量可不可以有不同名称?

引用变量

- 一般变量
  - 一般变量在声明时，会被分配一个独立的内存空间

引用变量

- 一般变量
  - 一般变量在声明时，会被分配一个独立的内存空间
- 引用变量
  - 引用变量是其他变量的别名，与其他变量共用同一块内存空间
  - 在变量类型和变量名之间加一个 &，表示该变量是一个引用
    - **数据类型 & 引用变量名 = 被引用的变量名;**
    - 引用需要在声明的时候进行初始化，与一个固定的变量绑定

引用变量

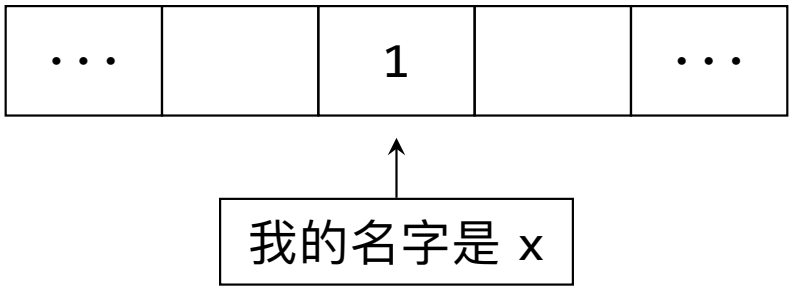
```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    int x = 1;
7    int &y = x;
8    y = 2;
9    cout << x << endl;
10   cout << y << endl;
11   return 0;
12 }
```

- 内存

引用变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    int x = 1;
7    int &y = x;
8    y = 2;
9    cout << x << endl;
10   cout << y << endl;
11   return 0;
12 }
```

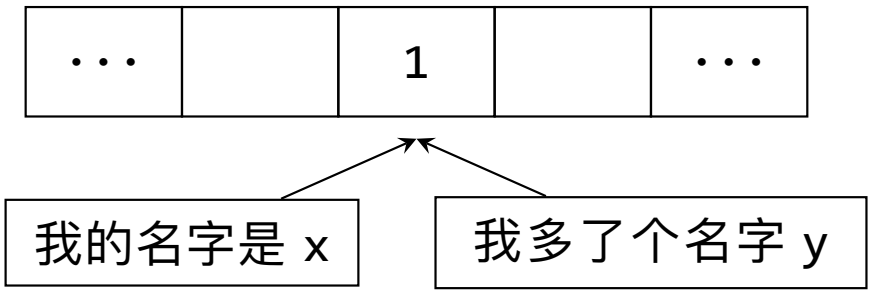
- 内存



引用变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    int x = 1;
7    int &y = x;
8    y = 2;
9    cout << x << endl;
10   cout << y << endl;
11   return 0;
12 }
```

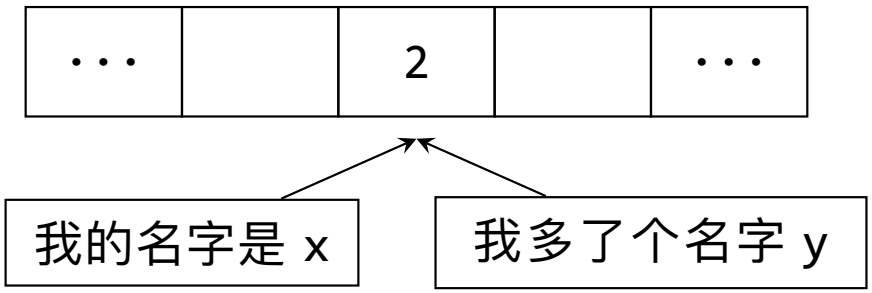
- 内存



引用变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    int x = 1;
7    int &y = x;
8    y = 2;
9    cout << x << endl;
10   cout << y << endl;
11   return 0;
12 }
```

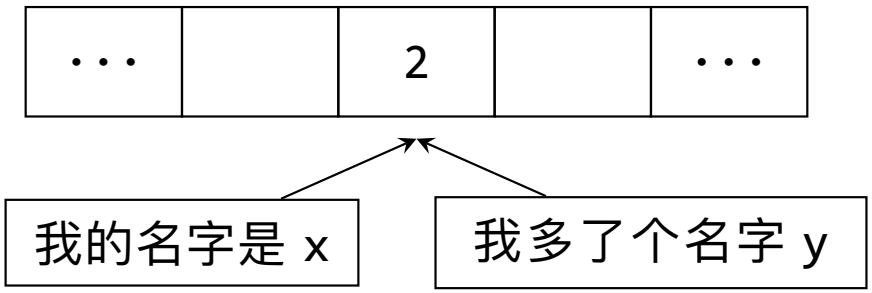
- 内存



引用变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    int x = 1;
7    int &y = x;
8    y = 2;
9    cout << x << endl;
10   cout << y << endl;
11   return 0;
12 }
```

- 内存



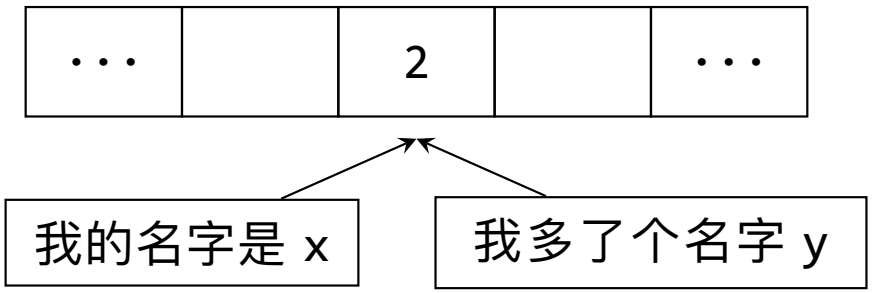
- 输出

  2

引用变量

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    int x = 1;
7    int &y = x;
8    y = 2;
9    cout << x << endl;
10   cout << y << endl;
11   return 0;
12 }
```

- 内存



- 输出

  2

  2

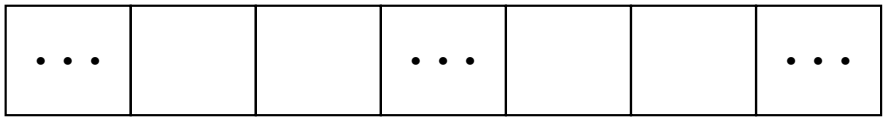
交换变量的值

交换两个变量的值，如何用函数实现?

交换变量的值

```
1  // #include ...
2
3  void mySwap(int a, int b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

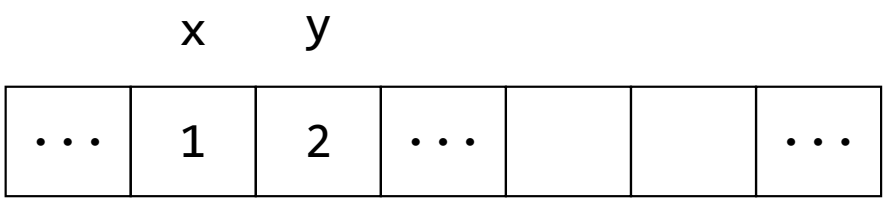
- 内存



交换变量的值

```
1  // #include ...
2
3  void mySwap(int a, int b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

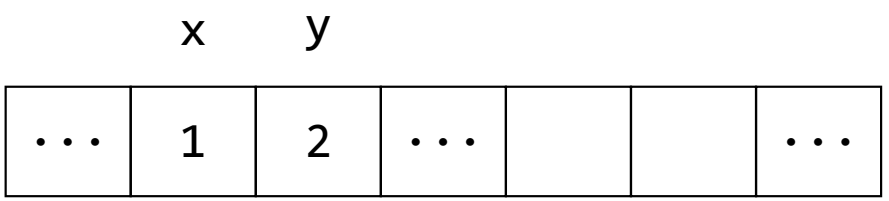
- 内存



交换变量的值

```
1  // #include ...
2
3  void mySwap(int a, int b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

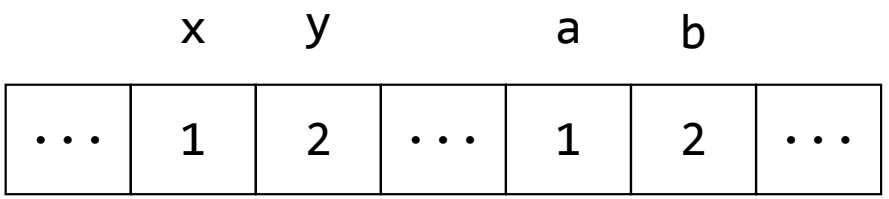
- 内存



交换变量的值

```
1  // #include ...
2
3  void mySwap(int a, int b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

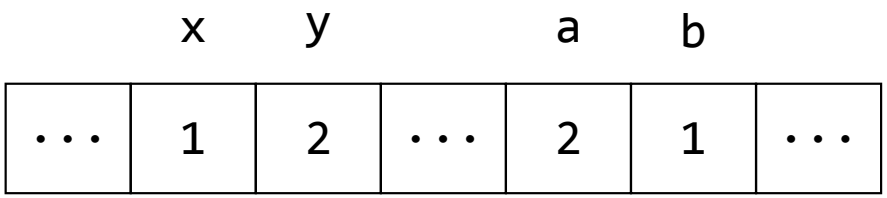
- 内存



交换变量的值

```
1  // #include ...
2
3  void mySwap(int a, int b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

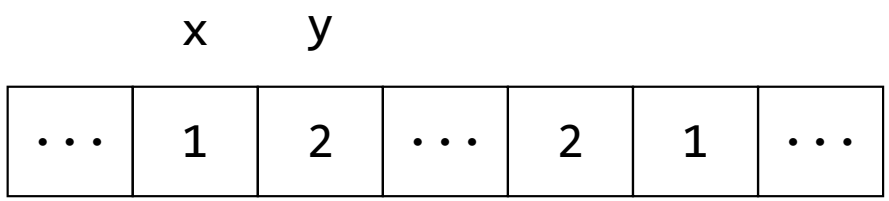
- 内存



交换变量的值

```
1  // #include ...
2
3  void mySwap(int a, int b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

- 内存



- 输出

  1 2

参数传递

- 按值传递
  - 若参数声明为一般形式，参数传递时只是将实参的数值赋值给形参，函数不能访问实参本身
- 按引用传递
  - 引用变量是其他变量的别名，与其他变量共用同一块内存空间
  - 将形式参数声明为引用类型，可以访问和修改实参本身

参数传递

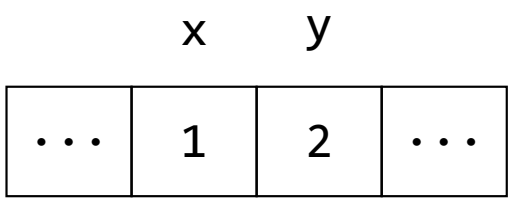
```cpp
// #include ...

void mySwap(int &a, int &b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main() {
  int x = 1, y = 2;
  mySwap(x, y);
  cout << x << " " << y << endl;
  return 0;
}
```

- 内存

参数传递

```
1  // #include ...
2
3  void mySwap(int &a, int &b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

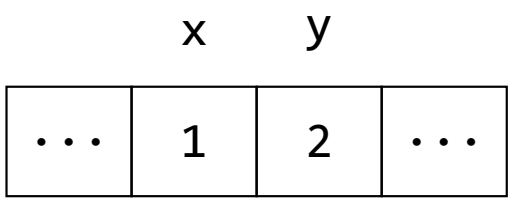
- 内存



参数传递

```
1  // #include ...
2
3  void mySwap(int &a, int &b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

- 内存



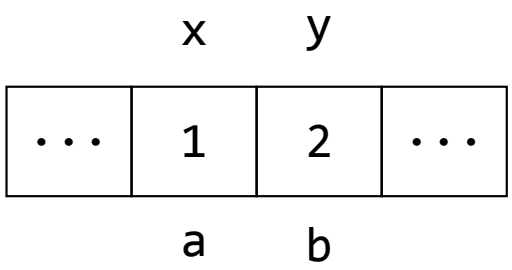
参数传递

```
1  // #include ...
2
3  void mySwap(int &a, int &b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

int &a = x;

int &b = y;

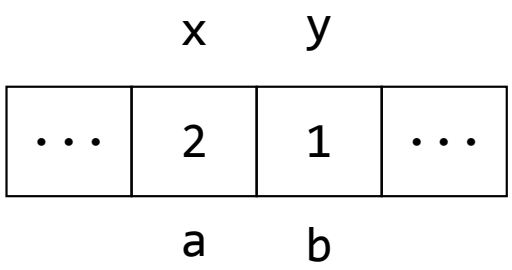
- 内存



参数传递

```
1  // #include ...
2
3  void mySwap(int &a, int &b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

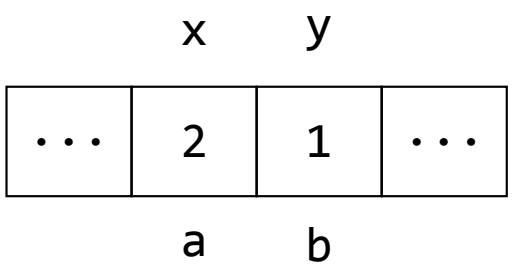
- 内存



参数传递

```
1  // #include ...
2
3  void mySwap(int &a, int &b) {
4    int tmp = a;
5    a = b;
6    b = tmp;
7  }
8
9  int main() {
10   int x = 1, y = 2;
11   mySwap(x, y);
12   cout << x << " " << y << endl;
13   return 0;
14 }
```

- 内存



- 输出

  2 1

参数传递

- 按值传递
  - 传递的是数值，形参与实参是不同变量，修改形参不影响实参
  - 不需要在函数中修改实参时使用
- 按引用传递
  - 传递变量本身，形参与实参是同一个变量
  - 需要在函数中修改实参时使用

目录

数组参数传递

参数是数组，应该按什么方式传递?

数组参数传递

- 按值传递?

数组参数传递

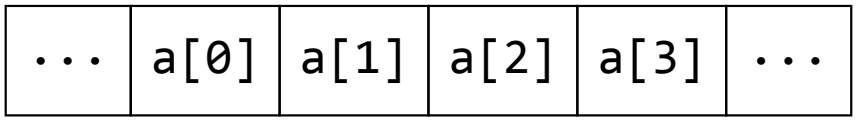
- 按值传递?
  - 把实参数组所有元素复制到形参数组中，需要付出的时间和储存空间代价过大
  - C++ 不支持按值传递数组

数组参数传递

- 按值传递?
  - 把实参数组所有元素复制到形参数组中，需要付出的时间和储存空间代价过大
  - C++ 不支持按值传递数组
- 函数传递数组默认用类似按引用传递的方式
  - 形参格式：**数据类型　数组名 []**
  - 不需要使用引用传递

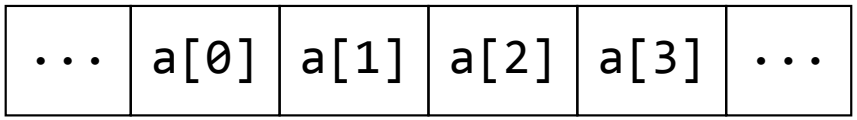
数组参数传递

- 数组储存在一块连续的内存中



数组参数传递

- 数组储存在一块连续的内存中
- 内存中每个位置有个“编号”，即“内存地址”
  - 通过数组名 a 可以访问到这个数组的首地址

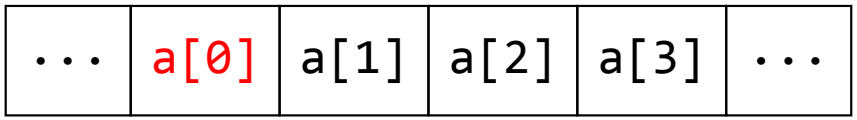


- 数组元素是通过首地址 + 偏移量来访问的
  - a[1] 储存在首地址往后偏移 1 个位置，a[2] 储存在首地址往后偏移 2 个位置……

数组参数传递

- 数组储存在一块连续的内存中
- 内存中每个位置有个“编号”，即“内存地址”
  - 通过数组名 a 可以访问到这个数组的首地址

输出数组名 a 可以看到这块内存的“编号”



- 数组元素是通过首地址 + 偏移量来访问的
  - a[1] 储存在首地址往后偏移 1 个位置，a[2] 储存在首地址往后偏移 2 个位置……

首地址与参数传递

- 只要知道数组的首地址，就能访问到数组的每个元素
- 参数传递首地址，被调用的函数就能访问原数组
  - 传递的是地址，效果类似按引用传递
  - 数组名的值表示数组首地址，实参填写数组名即可

例 2.3：一维数组的输入输出

编程题

- 实现一个 `input` 函数，功能：将 n 个整数存入一维数组中；
  实现一个 `output` 函数，功能：顺序输出数组中的 n 个整数，数字之间以空格间隔。
  请调用上述函数，输入 n ($n \leq 100$) 个整数，顺序输出这 n 个整数。

- 样例输入

```
5
3 1 5 2 8
```

- 样例输出

```
3 1 5 2 8
```

例 2.3：一维数组的输入输出

```cpp
// #include ...

// 函数功能：一维数组的输入
void input(int a[], int n) {
  for (int i = 1; i <= n; i++) cin >> a[i];
}

// 函数功能：一维数组的输出
void output(int a[], int n) {
  for (int i = 1; i <= n; i++) cout << a[i] << endl;
  cout << endl;
}

int A[110];

int main() {
  int n;
  cin >> n;
  input(A, n);
  output(A, n);
  return 0;
}
```

例 2.3：一维数组的输入输出

```
1  // #include ...
2
3  // 函数功能：一维数组的输入
4  void input(int a[], int n) {
5    for (int i = 1; i <= n; i++) cin >> a[i];
6  }
7
8  // 函数功能：一维数组的输出
9  void output(int a[], int n) {
10   for (int i = 1; i <= n; i++) cout << a[i] << endl;
11   cout << endl;
12 }
13
14 int A[110];
15
16 int main() {
17   int n;
18   cin >> n;
19   input(A, n);
20   output(A, n);
21   return 0;
22 }
```

形参加 []

例 2.3：一维数组的输入输出

```
1  // #include ...
2
3  // 函数功能：一维数组的输入
4  void input(int a[], int n) {
5    for (int i = 1; i <= n; i++) cin >> a[i];
6  }
7
8  // 函数功能：一维数组的输出
9  void output(int a[], int n) {
10   for (int i = 1; i <= n; i++) cout << a[i] << endl;
11   cout << endl;
12 }
13
14 int A[110];
15
16 int main() {
17   int n;
18   cin >> n;
19   input(A, n);
20   output(A, n);
21   return 0;
22 }
```

形参加 []

注意实参写数组名即可，不能写成 A[110]

小结：参数传递

- 按值传递（被调用的函数不会影响实参）
  - 形参的声明：和声明普通变量一样

小结：参数传递

- 按值传递（被调用的函数不会影响实参）
  - 形参的声明：和声明普通变量一样
- 按引用传递（被调用的函数能修改实参的值）
  - 形参的声明：变量名前面加 &

小结：参数传递

- 按值传递（被调用的函数不会影响实参）
  - 形参的声明：和声明普通变量一样
- 按引用传递（被调用的函数能修改实参的值）
  - 形参的声明：变量名前面加 &
- 数组传递（传递首地址，效果类似按引用传递）
  - 形参的声明：数组名后面加 []

目录

总结

- 函数的设计
  - 模块化编程
  - 函数的优点和适用场景

总结

- 函数的设计
  - 模块化编程
  - 函数的优点和适用场景
- 变量
  - 作用域
  - 同名变量
  - 引用变量

总结

- 函数的设计
  - 模块化编程
  - 函数的优点和适用场景
- 变量
  - 作用域
  - 同名变量
  - 引用变量
- 函数参数传递
  - 按值传递
  - 按引用传递
  - 数组传递