CS 1502H 程序设计思想与方法 (C++)

Chapter 7. 指针-2

陈东尧
上海交通大学
2025年秋季

上节课回顾

- **指针概念及指针变量的定义**
- **指针运算**
- **指针与数组**
- **动态内存分配**
- **字符串与指针**

```cpp
// chapter7-dynamic-memory.cpp
#include <iostream>
#include <cstring>
using namespace std;
int main(){
    int *p;
    char *q;
    p = new int(99);    //动态分配内存，并将 99 作为初始化值赋给它
    q = new char[10];

    strcpy(q, "abcde");
    cout<< *p << q << endl;
    cout << p << '\t' << (void *) q << endl;
    cout << &p << '\t'<< &q << endl;

    //int a[3] = {1, 2, 3};
    //char b[4] = {'a','b','c'};
    //cout << a << '\t' << b << endl;
    delete  p;
    delete  q;
    return 0;
}
```

```
99abcde
0x7fa24e705bd0  0x7fa24e705be0
0x7ff7b3c1cd50  0x7ff7b3c1cd48
```

```cpp
// chapter7-dynamic-memory.cpp
#include <iostream>
#include <cstring>
using namespace std;
int main(){
    int *p;
    char *q;
    p = new int(99);    //动态分配内存，并将 99
    q = new char[10];

    strcpy(q, "abcde");
    cout<< *p << q << endl;
    cout << p << '\t' << (void *) q << endl;
    cout << &p << '\t'<< &q << endl;

    //int a[3] = {1, 2, 3};
    //char b[4] = {'a','b','c'};
    //cout << a << '\t' << b << endl;
    delete  p;
    delete  q;
    return 0;
}
```

```
99abcde
0x7fa24e705bd0   0x7fa24e705be0
0x7ff7b3c1cd50   0x7ff7b3c1cd48
```

q为存放一个 char 类型一维数组的空间的首地址的字符型指针

问题：

1. 为什么输出q，它不是字符型指针吗？为什么输出的是字符串本身？
2. 输出地址为何需要(void *) q？

```cpp
// chapter7-dynamic-memory.cpp
#include <iostream>
#include <cstring>
using namespace std;
int main(){
    int *p;
    char *q;
    p = new int(99);    //动态分配内存，并将 99
    q = new char[10];

    strcpy(q, "abcde");
    cout<< *p << q << endl;
    cout << p << '\t' << (void *) q << endl;
    cout << &p << '\t'<< &q << endl;

    //int a[3] = {1, 2, 3};
    //char b[4] = {'a','b','c'};
    //cout << a << '\t' << b << endl;
    delete  p;
    delete  q;
    return 0;
}
```

```
99abcde
0x7fa24e705bd0  0x7fa24e705be0
0x7ff7b3c1cd50  0x7ff7b3c1cd48
```

q为存放一个 char 类型一维数组的空间的首地址的字符型指针

问题：

1. 为什么输出q，它不是字符型指针吗？为什么输出的是字符串本身？
2. 输出地址为何需要(void *) q？

解答：

C++标准库中 I/O 类对<<操作符进行了重载，在遇到字符型指针时会将其当作字符串名来处理，输出指针所指的字符串。

拓展：试试看分析这段代码的输出

现象1: 字符类型数组q直接输出字符串本身。
解析：
- cout 对 char* 有**特殊处理** —— 会从首地址开始输出字符，直到遇到 '\0'（字符串结束符）

现象2: p和&p是不一样的。
解析：
- p 是指针变量，它**存储的内容是动态内存**的首地址
- &p 是指针变量 p 自己的内存地址。

现象3: (void *)q和&q是不一样的。
解析：
- (void *)q将char*强制转换为void*，cout 对 void* 会输出**指针存储的地址**（即字符数组的首地址）
- &q输出指针变量 q **自身的内存地址**（与&p同理）

```
p = new int(99);
q = new char[10];
strcpy(q, "abcde");
// int a[3] = {1, 2, 3};
// char b[4] = {'a','b','c'};

cout<< *p << '\t' << q << endl;
// cout << a << '\t' << b << endl;

cout << p << '\t' << (void *) q << endl;
cout << &p << '\t' << &q << endl;
cout << *(&p) << '\t' << *(&q) << endl;
delete p;
delete q;
```

```
99      abcde
0x7fded1f05f70  0x7fded1f060c0
0x7ff7b491cd30  0x7ff7b491cd28
0x7fded1f05f70  abcde
```

拓展：进一步解析下面的例子

```
//你会看到下面代码中，(void *)q和&q是一样的：
char q[3] = {'a', 'b'};  // 字符数组，存储'a','b'，第三个元素默认'\0'（因为初始化列表长度不足）
cout << q << endl;       // 输出从q[0]开始的字符串（直到遇到'\0'），即"ab"
cout << (void *)q << endl;  // 强制转换为void*，输出数组q的首地址（q[0]的地址）
cout << &q << endl;      // 输出数组q整体的地址（数组作为一个整体的地址）
```

```
ab
0x16b36e76c
0x16b36e76c
```

现象：(void *)q和&q是一样的。
解析：

- 输出q则输出从q[0]开始的字符串（直到遇到'\0'），即"ab"
- 输出(void *)q则强制转换为void*，输出数组q的首地址（q[0]的地址）
- 输出&q则输出数组q整体的地址（数组作为一个整体的地址）

本讲大纲

- **指针概念及指针变量的定义**
- **指针运算**
- **指针与数组**
- **动态内存分配**
- **字符串与指针**

字符串与指针

- 字符串的另一种表示是定义一个指向字符的指针，然后直接将一个字符串常量赋给它
- 如：

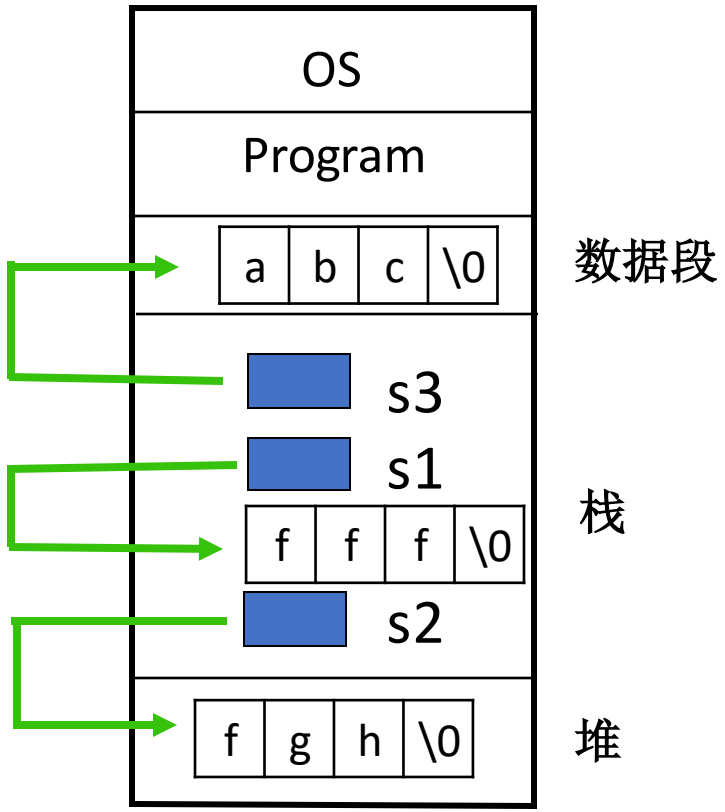
```
char *string;
string = “abcde”;
```

- 将存储字符串“abcde”的内存的首地址赋给指针变量string
- 字符串常量存储在一个称为数据段的内存区域里

用指针表示字符串的其他用法

- 将表示字符串的字符数组名赋给指针
- 用动态内存分配的方法申请一块空间存储字符串，让指针指向这块空间

```
int main() {
    char  *s1, *s2, *s3 = “abc”;
    char  ch[] = “fff”;
    s1 = ch; s2 = new char[4];
    strcpy(s2, “fgh”);
    return 0;
}
```



字符串作为函数的参数

- 字符串作为参数和数组名作为参数的传递一样，可以有两种方法：
  - 作为**字符数组**传递
  - 作为**指向字符的指针**传递
- 两种传递方式的本质是一样的，都是传递了字符串的首地址

const与指针

const int *p;

不能通过p修改它指向的单元的内容，但p可以指向不同的变量

int *const p = &x;

可以通过p修改指向的对象，但p的值不能变

即p永远只能指向x。p定义时必须给出初值

const int *const p = &x;

p的值不能修改，*p的值也不能修改

提高了安全性

超长加法程序

- 试设计一个函数，计算两个128位的整数（十进制）的和，结果作为函数值返回。
  - 设计思想：用一个由数字组成的字符串来表示数据。如允许的最大长度为128位，则字符数组的长度为128。
  - 如数字123，可表示为：

0	1	2	3	4	5	…	127
‘3’	‘2’	‘1’	‘0’	‘0’	‘0’	‘0’	‘0’

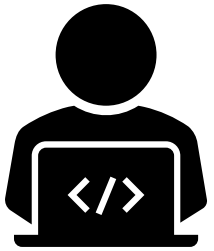
思路：set(“123”, num1); //num1为字符串，即把用户定义的任意长短的数据用字符串来表示

超长加法程序：生成函数设计

```
int set(const char *s1, char *s) { //从字符串常量到数组/指针
    int i= strlen(s1) - 1, j = 0;

    while (i>= 0) {
        if (s1[i] > '9' || s1[i] < '0')
        { cout << "set error\n";  return -1; }
        else { s[j] = s1[i]; --i; ++j; }
    }
    for ( ; j < 128; ++j ) s[j] =  ‘0’ ;   // 置空格

    return 0;
}
```



代码演示

chapter7-long-add.cpp

超长加法程序：加法函数设计

```
void add(const char  *p1, const char  *p2, char *s)  {
    int i, j = 0; /* j为进位 */

    for (i=0; i < 128; ++i)  {
        s[i] = p1[i] - '0' + p2[i] - '0' + j;
        if ( s[i] >= 10)  j = s[i] / 10; else j = 0;
        s[i] = s[i] % 10 + '0';
    }
}
```

超长加法程序：显示函数设计

```
void show(const char *s) {
    int i=127;

    while (s[i] == '0')  --i; // 跳过前置的0
    while (i>=0) {
        cout << s[i];
        --i;
    }
    cout << endl;
}
```

超长加法程序：示例

```
int main() {
    char num1[128], num2[128], sum[128];

    set("123", num1);
    set("345", num2);

    show(num1);
    show(num2);
    add(num1, num2, sum);
    show(sum);

    return 0;
}
```

执行结果：

123

345

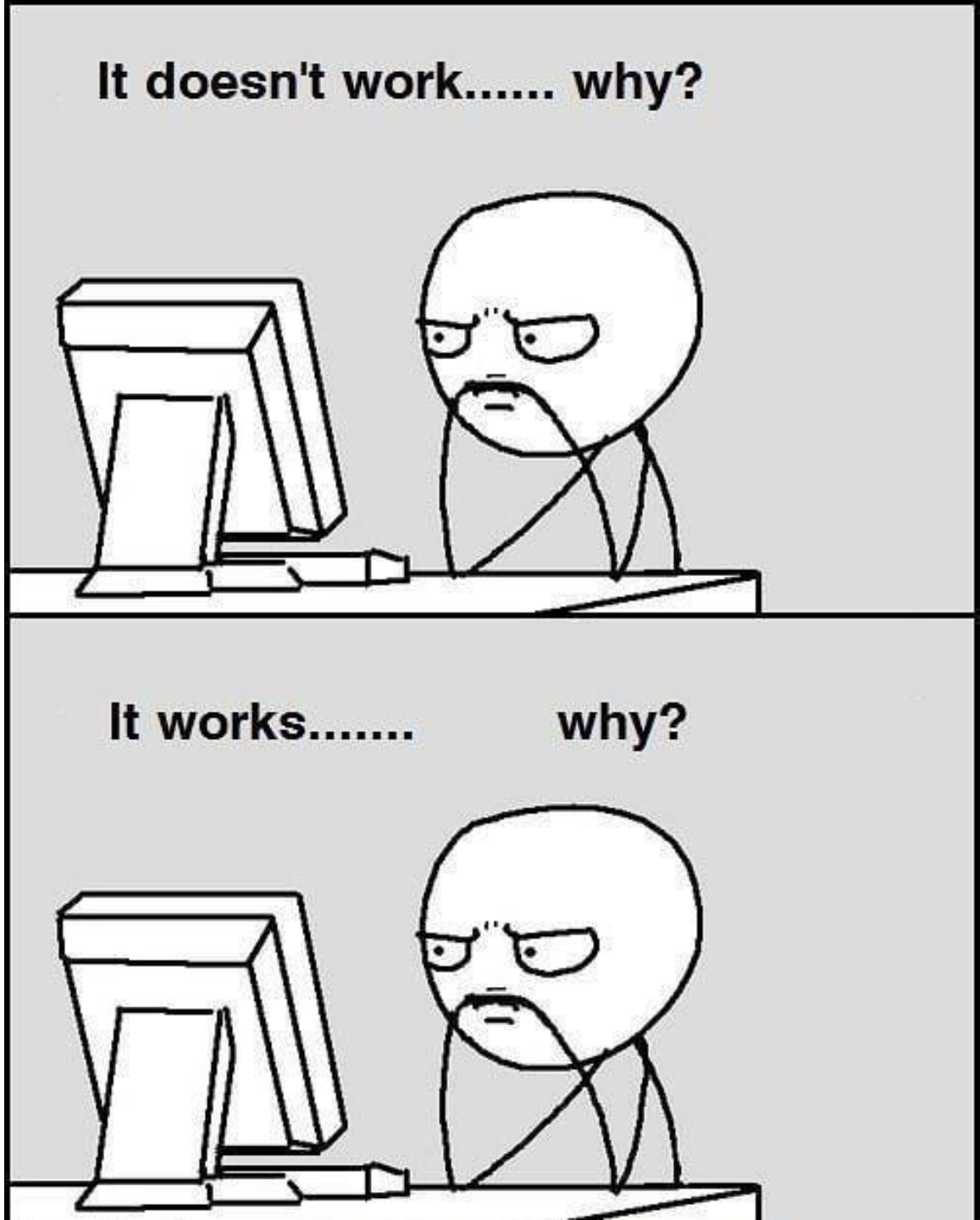
468

本章内容

- 指针概念及指针变量的定义
- 指针运算
- 指针与数组
- 动态内存分配
- 字符串与指针
- 指针与函数
- 引用类型与函数
- 指针数组与多级指针

本讲内容

指针与函数



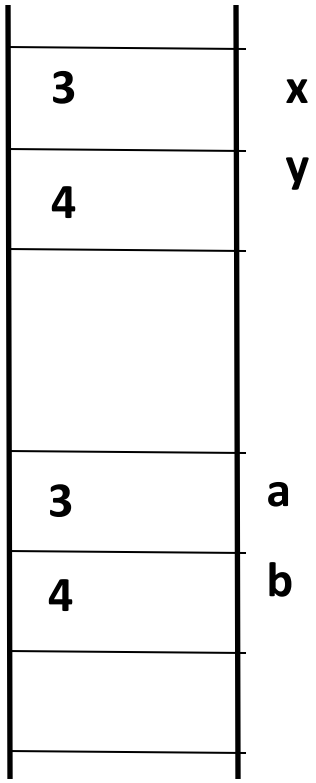
指针作为函数参数

例：编一函数，交换二个参数值 （再次回顾）

新手可能会编出如下的函数：

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

打算通过调用swap(x, y)交换变量 x 和 y 的值

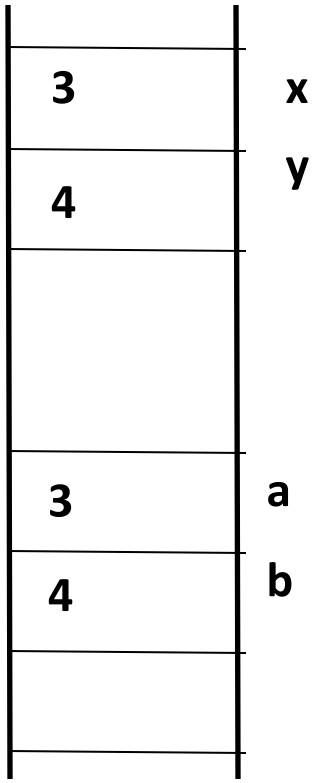


指针作为函数参数

例：编一函数，交换二个参数值

新手可能会编出如下的函数：

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

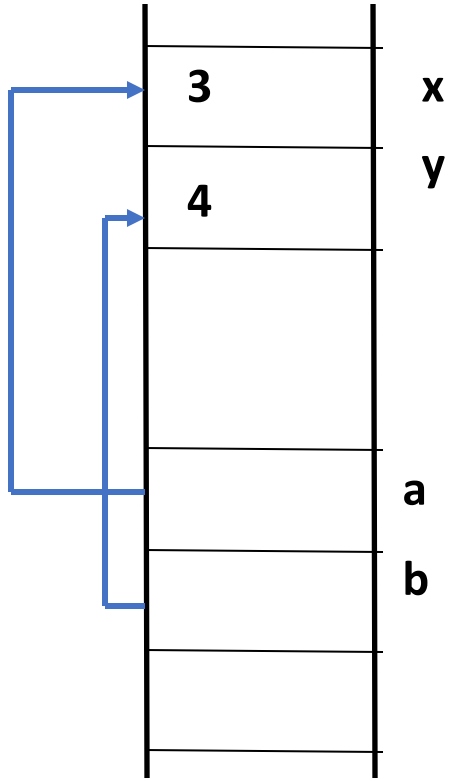


打算通过调用swap(x, y)交换变量 x 和 y 的值

这里采用的是**值传递机制（pass by value）**，函数中a、b值的交换不会影响实际参数x和y的值。

可用的方法

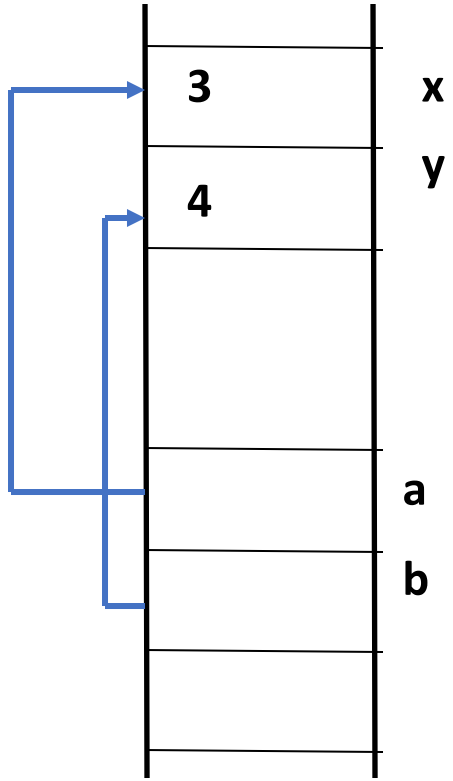
```
void swap(int  *a, int *b)
{
      int c;
      c=*a;   *a= *b;      *b=c;
 }
```



用指针作为参数可以在函数中修改主调程序的变量值，即实现变量传递

可用的方法

```
void swap(int  *a, int *b)
{
      int c;
      c=*a;   *a= *b;      *b=c;
 }
```



交换x和y的值，可以调用swap(&x, &y)

用指针作为参数可以在函数中修改主调程序的变量值，即实现变量传递

解一元二次方程的函数

问题

如何让此函数**返回2个根**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

解一元二次方程的函数

问题

如何让此函数**返回2个根**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

解决方案

采用指针作为函数的参数

由调用程序准备好存放两个根的变量，将变量地址传给函数，函数将两个根的值分别放入这两个地址

解一元二次方程的函数

问题

如何让此函数**返回2个根**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

解决方案

采用指针作为函数的参数

由调用程序准备好存放两个根的变量，将变量地址传给函数，函数将两个根的值分别放入这两个地址

难点1

如何传递2个根？

解一元二次方程的函数

问题

如何让此函数**返回2个根**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

解决方案

采用指针作为函数的参数

由调用程序准备好存放两个根的变量，将变量地址传给函数，函数将两个根的值分别放入这两个地址

难点1
如何传递2个根？

难点2
如何判断根的数量？

函数原型

函数原型

void SolveQuadratic(double a, double b, double c, double *px1, double *px2)

函数调用

函数原型

函数原型

void SolveQuadratic(double a, double b, double c, double *px1, double *px2)

函数调用

SolveQuadratic(1.3, 4.5, 2.1, &x1, &x2)

SolveQuadratic(a, b, c, &x1, &x2)

两类函数参数

输入参数：用值传递

输出参数：用指针传递

在参数表中，输入参数放在前面，输出参数放在后面

原型的改进

如何获知方程有根、没根?

并不是每个一元二次方程都有两个不同根，有的可能有两个等根，有的可能没有根。函数的调用者如何知道x1和x2中包含的是否是有效的解?

解决方案

原型的改进

如何获知方程有根、没根?

并不是每个一元二次方程都有两个不同根，有的可能有两个等根，有的可能没有根。函数的调用者如何知道x1和x2中包含的是否是有效的解?

解决方案

让函数返回一个整型数。该整型数表示解的情况

0：两个解

1：一个解

2：无解

3：不是一元二次方程

完整的函数

```
int SolveQuadratic (double a,   double b,   double c,   double *px1,  double *px2)
{
        double disc, sqrtDisc;

        if(a == 0) return 3;          //不是一元二次方程

        disc = b * b - 4 * a * c;
        if( disc < 0 ) return 2;       //无根

        if ( disc == 0 ) { *px1 = -b /(2 * a); return 1;}  //等根

       //两个不等根
        sqrtDisc = sqrt(disc);
        *px1 = (-b + sqrtDisc) / (2 * a);
        *px2 = (-b - sqrtDisc) / (2 * a);

         return 0;
}
```

函数的调用

```
int main()
{
    double a,b,c,x1,x2;
    int  result;

    cout << "请输入a,b,c: ";
    cin >> a >> b >> c;

    result = SolveQuadratic(a, b, c, &x1, &x2);
    switch (result)     {
        case 0: cout << "方程有两个不同的根：  x1 = " << x1 << "  x2 = " << x2; break;
        case 1: cout << "方程有两个等根：  " << x1; break;
        case 2: cout << "方程无根"; break;
        case 3: cout << "不是一元二次方程";
    }

    return 0;
}
```

数组传递的进一步讨论

数组传递的本质是地址传递

形参和实参可以使用数组名，也可以使用指针

数组传递是函数原型可写为：

```
type func(type a[], int size);
```

也可写为

```
type func(type *p, int size);
```

但在函数内部，a 和 p 都能当作数组使用

调用时，对这两种形式都可用数组名或指针作为实参

建议

如果传递的是数组，用第一种形式；如果传递的是普通的指针，用第二种形式

数组传递本质是指针传递

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

void f(int arr[ ], int k)
{
    cout << sizeof(arr) << "    " << sizeof(k) << endl;
 }
void main()
{
     int a[10]={1,2,3,4,5,6,7,8,9,0};

     cout << sizeof(a) << endl;
     f(a,10);
  }
```

数组传递本质是指针传递

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

void f(int arr[ ], int k)
{
    cout << sizeof(arr) << "   " << sizeof(k) << endl;
 }
void main()
{
    int a[10]={1,2,3,4,5,6,7,8,9,0};

    cout << sizeof(a) << endl;
    f(a,10);
  }
```

输出：

40

8　　4

注：

64位机器会判定pointer大小为8 bytes

32位机器可能会判定sizeof(arr)为4 bytes

数组传递的灵活性

```
void  sort(int p[ ] , int n)
{  …   }

int main()
{
    int a[100];
     …
    sort(a, 100);  //排序整个数组
    sort(a, 50);  //排序数组的前50个元素
    sort(a+50, 50);  //排序数组的后50个元素
    …
}
```

返回指针的函数

函数的返回值也可以是一个指针

返回指针的函数原型

类型　*函数名（形式参数表）；

实例

问题

设计一个函数从一个字符串中取出一个**子串**

原型设计

- 参数：从哪一个字符串中取子串、起点和终点
- 返回值：取出的子串
- 字符串可以用一个指向字符的指针表示，所以函数的返回值**是一个指向字符的指针**
- **返回值指针指向的空间必须在返回后还存在**
- 可以用动态字符数组

```
char *subString( const char  *s,  int  start,  int  end )
{
   int len = strlen( s );
   if ( start < 0 || start >= len || end < 0 ||  end >= len || start > end ) {
            cout << "起始或终止位置错" << endl;
            return NULL;
   }
   char *sub = new char[end - start + 2];
   strncpy(sub, s + start, end - start +1);

   return sub;
}
```

```
char *subString( const char  *s,  int  start,  int  end )
{
   int len = strlen( s );
   if ( start < 0 || start >= len || end < 0 ||  end >= len || start > end ) {
            cout << "起始或终止位置错" << endl;
            return NULL;
   }
   char *sub = new char[end - start + 2];
   strncpy(sub, s + start, end - start +1);

   return sub;
}
```

调用subString的函数后，须释放空间

```
如:    char *pc = subString(str, 3, 10);
       cout << pc << endl;
       delete [] pc;
```

实例2：交换最大最小值

- 例：交换a, b, c中的最大值和最小值
- 该问题的解决分成三个步骤：找出三个变量中的最大者；找出三个变量中的最小者；交换这两个变量的值。分别用三个函数来实现。
- 思考：max 和 min 函数的原型能否设计为
- int max(int, int, int) 和 int min(int, int, int)

//交换a, b, c中的最大值和最小值

```
int *max(int *a, int *b, int *c) { //返回最大的变量的地址
    if (*a>*b)
        if (*a>*c)  return(a);   else return(c);
    else if (*b>*c)   return(b);     else return(c);
}

int *min(int *a, int *b, int *c) { //返回最小的变量的地址
    if (*a<*b)
        if (*a<*c)  return(a);  else return(c);
    else if (*b<*c) return(b);   else return(c);
}

void swap(int *a, int *b) {
    int c;
    c = *a; *a = *b; *b = c;
}
```

//交换a, b, c中的最大值和最小值

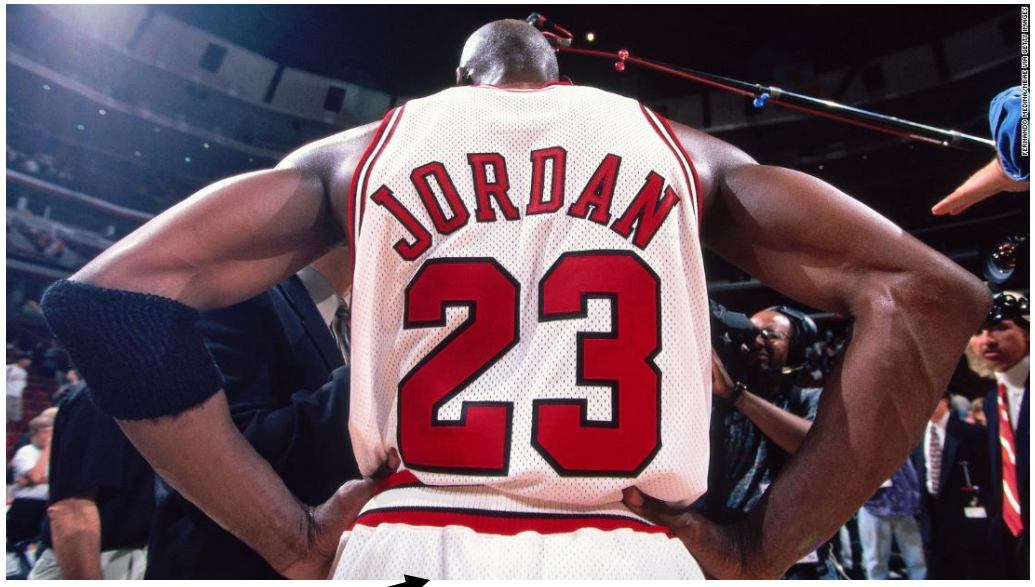
```
int main() {
    int x, y, z;
    cin >> x >> y >> z;
    cout << endl << x << '\t' << y << '\t' << z;
    swap(max(&x, &y, &z),  min(&x, &y, &z));
    cout << endl << x << '\t' << y << '\t' << z;
    return 0;
}
```

本章内容

- 指针概念及指针变量的定义
- 指针运算
- 指针与数组
- 动态内存分配
- 字符串与指针
- 指针与函数
- 引用类型与函数
- 指针数组与多级指针

什么是别名（alias）

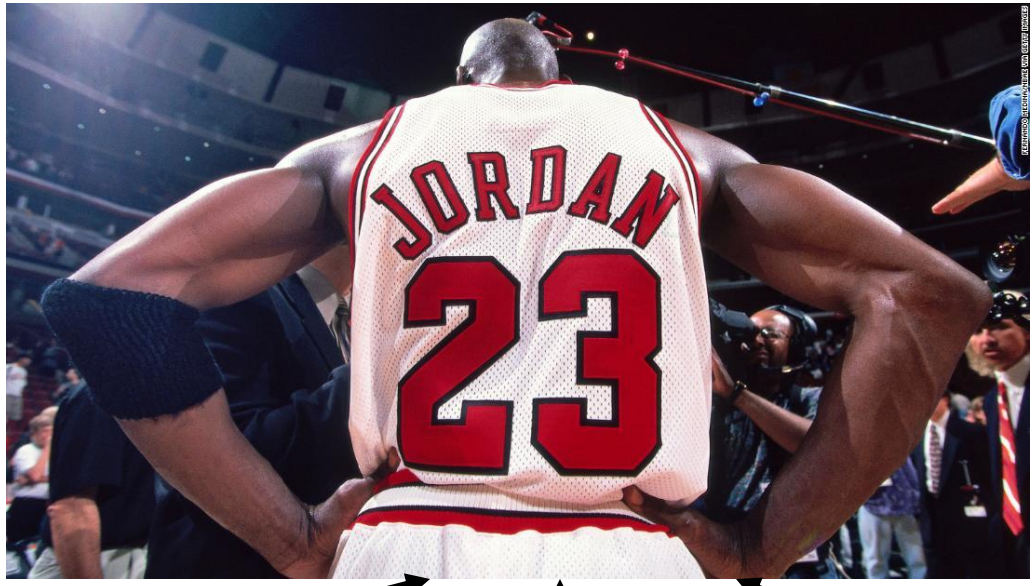
- 别名：as known as



迈克尔乔丹

什么是别名（alias）

- 别名：as known as



迈克尔乔丹　　Air Jordan　　MJ

引用概念

引用的意义：别名（alias)

给某个变量取一个别名,使一个内存单元可以**通过不同的变量名**来访问

定义格式

类型　&变量名 = 变量名;

例：int i;

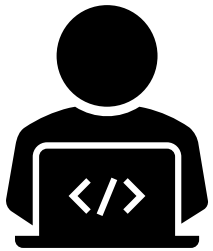
　　int &j = i;

　　j 是 i 的别名，i 与 j 是同一个内存单元。

引用的主要目的有两种：（1）将引用作为函数的参数；（2）将引用作为函数的返回值

【扩展阅读】C++中的引用https://www.tutorialspoint.com/cplusplus/cpp_references.htm

引用举例

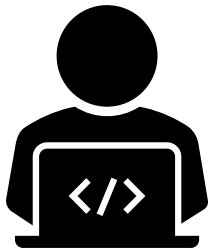


代码演示

chapter7-refernce-trivia.cpp

```
#include <iostream>
using namespace std;
int main () {
        // declare simple variables
        int i;
        double d;
        // declare reference variables
        int& r = i;
        double& s = d;
        i = 5;
        cout << "Value of i : " << i << endl;
        cout << "Value of i reference : " << r << endl;
        d = 11.7;
        cout << "Value of d : " << d << endl;
        cout << "Value of d reference : " << s << endl;
        return 0;
}
```

引用举例



代码演示

chapter7-refernce-trivia.cpp

```
#include <iostream>
using namespace std;
int main () {
        // declare simple variables
        int i;
        double d;
        // declare reference variables
        int& r = i;
        double& s = d;
        i = 5;
        cout << "Value of i : " << i << endl;
        cout << "Value of i reference : " << r << endl;
        d = 11.7;
        cout << "Value of d : " << d << endl;
        cout << "Value of d reference : " << s << endl;
        return 0;
}
```

```
Value of i : 5
Value of i reference : 5
Value of d : 11.7
Value of d reference : 11.7
```

C++中的引用

- 定义引用时必须立即对它初始化，不能定义完成后再赋值。如：

```
int i;
int &j; //错误
j = i;
```

为引用提供的初始值可以是一个变量或另一个引用。如：

```
int i = 5;
int &j1 = i;
int &j2 = j1;
```

引用不可重新赋值，不可使其作为另一变量的别名。

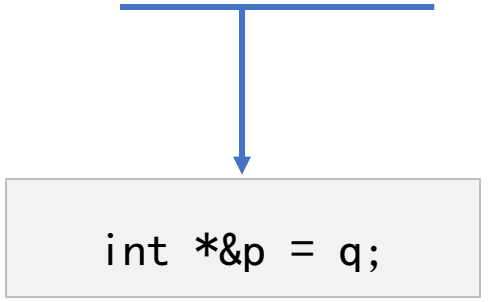
```
int i, k;
int &j = i;
j = &k;//错误 (注：指针可以指向另一个变量，有区别)
```

引用使用的注意事项

- 引用不同于普通变量，下面的类型说明都是非法的：
  int &a[3] = ……; //不能建立引用数组
  int &*p = ……; //不能建立指向引用的指针（但可以有指针的引用）
  int &&r= ……; //不能建立引用的引用

引用使用的注意事项

- 引用不同于普通变量，下面的类型说明都是非法的：
  int &a[3] = ……; //不能建立引用数组
  int &*p = ……; //不能建立指向引用的指针（但可以有指针的引用）
  int &&r= ……; //不能建立引用的引用



第一种：引用作为参数传递 (Reference as Parameter)

- C++引用的主要目的是将引用作为函数的参数

指针参数

```
void swap(int *m,  int *n)
{
     int temp;
     temp=*m;
    *m=*n;
    *n=temp;
}
调用：swap(&x, &y)
```

引用参数

```
void swap(int &m,  int &n)
{
     int temp;
     temp=m;
     m=n;
     n=temp;
}
调用：swap( x, y)
```

注意：实参必须是变量，而不能是一个表达式或常量

验证引用传递

```
void f( int & r )
{
    cout <<   “r= “ << r <<endl;
    cout <<   “&r= “ << &r <<endl;
    r= 5;
    cout <<   “r= “ << r <<endl;
}

int main( )
{
    int x=47;
    cout <<   “x= “ << x <<endl;
    cout <<   “&x= “ << &x <<endl;
    f(x);
    cout <<   “x= “ << x <<endl;
}
```

验证引用传递的地址是否不变

```
void f( int & r )
{
    cout <<   “r= “ << r <<endl;
    cout <<   “&r= “ << &r <<endl;
    r= 5;
    cout <<   “r= “ << r <<endl;
}

int main( )
{
    int x=47;
    cout <<   “x= “ << x <<endl;
    cout <<   “&x= “ << &x <<endl;
    f(x);
    cout <<   “x= “ << x <<endl;
}
```

执行结果

x = 47

&x = **0x065FE00**

r = 47

&r = **0x065FE00**

r=5

x = 5

引用传递意义

- 在C++中，函数参数一般都采用引用传递
- 减少函数**调用时的开销**（在后续struct和类的内容展开后，这一点会更明显）
  - 不需要分配新的内存空间
  - 不需要复制实际参数的数值
- 作为输出参数
- 引用传递是地址传递的另一种更简单明了的实现方法。
- 如果在函数内不许改变参数的值，则形式参数用const限定。
- 对非const的参数采用引用传递时，实际参数不能是常量或临时量。

第二种：引用作为函数的返回值（Reference as Return Value）

- 函数的返回值可以是一个引用。它表示函数的返回值是函数内某一个变量（不能是局部变量）的引用。
  - 为什么不能是局部变量?
- 如：交换a, b, c中的最大值和最小值的函数也可以用引用实现。
- 返回引用的函数原型：

  类型 &函数名 (形式参数表);

第二种：引用作为函数的返回值（Reference as Return Value）

```
//交换a, b, c中的最大值和最小值
int &max(int &a, int &b, int &c) {
    if (a > b)
    if (a > c)  return(a);   else return(c);
    else if (b > c)   return(b);  else return(c);
}

int &min(int &a, int &b, int &c) {
    if (a < b)
    if (a < c)  return(a);  else return(c);
    else if (b < c) return(b);   else return(c);
}

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

第二种：引用作为函数的返回值（Reference as Return Value）

关键用途

将函数用于赋值运算符的左边，即作为**左值**

减少函数返回时的开销

实例

```
int a[] = {1, 3, 5, 7, 9};
int &index(int);    //声明返回引用的函数
void main()
{
    index(2) = 25;    //将a[2]重新赋值为25
    cout << index(2);
}
int &index(int j)
{   return a[j];    }    //返回第j个元素的引用
```

【扩展阅读】C++中的左值和右值：https://nettee.github.io/posts/2018/Understanding-lvalues-and-rvalues-in-C-and-C/

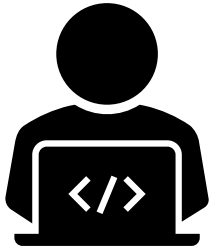
const与引用 （const reference）

定义格式

const 类型 &变量名 = 变量名;

例：int i;

const int &j = i;



代码演示

chapter7-const-reference-trivia.cpp

常量引用的意义

控制变量的修改，不管被引用对象是常量还是变量，用此名字时不能赋值

如 i= 5 是合法的， j = 5 是非法的

const引用的初值可以是常量或表达式。如

const int &y = 2+5;

常量的引用传递

void f(const double &);

实际参数可以是常量，也可以是变量或表达式

实际参数是变量时→形式参数是实际参数的引用

实际参数是常量时→函数中会创建一个临时变量

返回常量引用的函数

用途

仅减少函数返回时的开销，保证返回变量不被修改

实例

```
int a[] = {1, 3, 5, 7, 9};
const  int &index(int);    //声明返回引用的函数
void main()
{
      index(2) = 25;     //错
      cout << index(2); // 正确
}
const  int &index(int j)
{   return a[j];     }     //函数是a[j]的一个常量引用
```