



# 《计算概论A》课程 程序设计部分

## 指针 (3)

李 戈

北京大学 信息科学技术学院 软件研究所

[lige@sei.pku.edu.cn](mailto:lige@sei.pku.edu.cn)



北京大学



# 指针与函数



北京大学



# 内容提要

## ■ 指针与函数

- ◆ 指针用做函数参数
- ◆ 指针用做函数返回值
- ◆ 指向函数的指针



北京大学



# 指针变量作为函数参数

- 输入2个整数，按由大到小顺序输出。

```
int main()
{
    int a,b,c,*p1,*p2;
    cin>>a>>b;
    p1=&a; p2=&b;
    Rank(p1, p2);
    cout<<a<<b<<endl;
    return 0;
}
```

```
void Rank(int *q1, int *q2)
{
    int temp;
    if(*q1<*q2)
    {
        temp = *q1;
        *q1 = *q2;
        *q2 = temp;
    }
}
```





# 指针变量作为函数参数

- 输入A、B、C 3个整数，按由大到小顺序输出。

```
#include<stdio.h>
void main()
{
    int a,b,c,*p1,*p2,*p3;
    cin>>a>>b>>c;
    p1=&a; p2=&b; p3=&c;
    Rank(p1, p2, p3);
    cout<<a<<b<<c<<endl;
}
```

```
void Swap(int *pt1, int *pt2)
{
    int temp;
    temp = *pt1;
    *pt1 = *pt2;
    *pt2 = temp;
}

void Rank(int *q1, int *q2, int *q3)
{
    if(*q1<*q2) Swap(q1,q2);
    if(*q1<*q3) Swap(q1,q3);
    if(*q2<*q3) Swap(q2,q3);
}
```





## 多维数组名做函数参数

**例：**有一个 $3 \times 4$ 的矩阵，求所有元素中的最大值。

```
max-value(_____)
{
    int max = array[0][0];
    for(int i=0; i<3; i++)
        for(int j=0; j<4; j++)
            if(array[i][j]>max)
                max = array[i][j];
    return max;
}
main( )
{
    int a[3][4] = {{1,3,5,7},{9,11,13,15},{2,4,6,8}};
    cout<<"The Max value is "<<max-value(a);
}
```



## 多维数组名做函数参数

**例：**有一个 $3 \times 4$ 的矩阵，求所有元素中的最大值。

```
max-value(int (*array)[4])
{
    int max = array[0][0];
    for(int i=0; i<3; i++)
        for(int j=0; j<4; j++)
            if(array[i][j]>max)
                max = array[i][j];
    return max;
}
main( )
{
    int a[3][4] = {{1,3,5,7},{9,11,13,15},{2,4,6,8}};
    cout<<"The Max value is "<<max-value(a);
}
```



## 数组名实参 vs. 指针形参

- 可否将数组名作为实参附给指针型形参？ 可以！

```
#include<iostream.h>
int main()
{
    int a[10] =
        {1,2,3,4,5,6,7,8,9,10};
    sum(a,10);
    return 0;
}
```

```
void sum(int *p, int n)
{
    int total = 0;
    for(int i=0;i<n;i++)
    {
        total += *p++;
    }
    cout<<total<<endl;
}
```







## 数组名实参 vs. 数组名形参

- 数组名可否用作形参以接收指针实参？ 可以！

```
#include<iostream.h>
int main()
{
    int a [10] =
        {1,2,3,4,5,6,7,8,9,10};
    int *p = a;
    sum(p,10);
    return 0;
}
```

```
void sum(int array[], int n)
```

等价于 `sum(int *array, int n)`

```
{ int total = 0;
  for(int i=0;i<n;i++)
  {
      total += *array++;
  }
  cout<<total<<endl; }
```

C++编译都将形参数组名作为指针变量来处理！



# 值传递 vs. 地址传递

## ■ 可否实现由小到大的排序？

```
#include<iostream>
using namespace std;
void main()
{
    int a[2] = {12, 5};
    rank(a[0], a[1]);
    cout<<"min to max: "
    <<a[0]<<" ", <<a[1]<<endl;
}
```

```
void rank(int a, int b)
{
    int temp;
    if(a > b)
    {
        temp = a;
        a = b;
        b = temp;
    }
}
```



# 值传递 vs. 地址传递

## ■ 如何实现由小到大的排序？

```
#include<iostream>
using namespace std;
int a[2] = {12, 5};
void main( )
{
    rank( );
    cout<<"min to max: "
    <<a[0]<<" ", <<a[1]<<endl;
}
```

```
void rank( )
{
    int temp;
    if(a[0] > a[1])
    {
        temp = a[0];
        a[0] = a[1];
        a[1] = temp;
    }
}
```



# 值传递 vs. 地址传递

## ■ 如何实现由小到大的排序？

```
#include<iostream>
using namespace std;
void main( )
{
    int a[2] = {12,5};
    rank(&a[0], &a[1]);
    cout<<"min to max:
    "<<a[0]<<" , "<<a[1]<<endl;
}
```

```
void rank(int *a, int *b)
{
    int temp = 0;
    if(*a > *b)
    {
        temp = *a;
        *a = *b;
        *b = temp;
    }
}
```



# 值传递 vs. 地址传递

## ■ 还能实现由小到大的排序？

```
#include<iostream.h>
int main()
{
    int a[2] = {12,5};
    rank(&a[0], &a[1]);
    cout<<"min to max:
    "<<a[0]<<"", "<<a[1]<<endl;
    return 0;
}
```

```
void rank(int *a, int *b)
{
    int *temp = NULL;
    if(*a > *b)
    {
        temp = a;
        a = b;
        b = temp;
    }
}
```



# 如何“限制”指针的功能



北京大学



# 指针做函数参数

- 如下程序完成什么功能？

```
#include<iostream>
using namespace std;
int main( )
{
    int a[2] = {12, 5};
    cout<<"Max: "
        <<max(a, 2)
        <<endl;
    return 0;
}
```

```
int max(int *p, int n)
{
    int i = 0, temp = 0;
    for(i = 0; i < n; i++)
    {
        if(*(p+i) > temp)
        {
            temp = *(p+i);
        }
    }
    return temp;
}
```



# 指针做函数参数

- 如下程序完成什么功能？

```
#include<iostream>
using namespace std;
void main( )
{
    int a[2] = {12, 5};
    cout<<"Max: "
        <<max(a, 2)
        <<endl;
}
```

```
int max(int *p, int n)
{
    int i = 0, temp = 0;
    for(i = 0; i < n; i++)
    {
        if(*(p+i) > temp)
        {
            temp = *(p+i);
        }
        else
        {
            *(p+i) = 0;
        }
    }
    return temp;
}
```





# 指针做函数参数

- 如下程序完成什么功能？

```
#include<iostream>
using namespace std;
void main( )
{
    int a[2] = {12, 5};
    cout<<"Max: "
        <<max(a, 2)
        <<endl;
}
```

```
int max(const int *p, int n)
{
    int i = 0, temp = 0;
    for(i = 0; i < n; i++)
    {
        if(*(p+i) > temp)
        {
            temp = *(p+i);
        }
        else
        {
            *(p+i) = 0; //错误
        }
    }
    return temp;
}
```



# 符号常量

## ■ 符号常量声明语句:

◆ 方式一: **const** 数据类型 常量名=常量值;

◆ 方式二: 数据类型**const** 常量名=常量值;

```
#include <iostream.h>
```

```
void main()
```

```
{
```

```
    const float PI=3.14159f; // float const PI=3.14159f;
```

```
    float r;
```

```
    cout<<"请输入半径r: ";
```

```
    cin>>r;
```

```
    cout<<"圆面积为:"<<PI*r*r<<endl;
```

```
}
```



北京大学



## 指向符号常量的指针

### ■ 定义语句: `const int *p ;`

```
#include <iostream>
using namespace std;
void main()
{
    int a = 256;
    int *p = &a;
    *p = 257;
    cout<<*p<<endl;
}
```

```
#include <iostream>
using namespace std;
void main()
{
    int a = 256;
    const int *p = &a;
    *p = 257; //错误
    cout<<*p<<endl;
}
```





# 指向符号常量的指针

## ■ 用途

```
void mystrcpy(char *dest, const char *src)
```

```
{ ..... }
```

```
int main()
```

```
{
```

```
    char a[20] = "How are you!";
```

```
    char b[20];
```

```
    mystrcpy(b,a);
```

```
    cout<<b<<endl;
```

```
    return 0;
```

```
}
```

保证字符串src不被修改!



北京大学



# 指向符号常量的指针

## ■ 定义方式

```
int main()
```

```
{
```

```
    const int a = 78; const int b = 28; int c = 18;
```

```
    const int * pi = &a;
```

```
    *pi = 58;
```

```
    pi = &b; *pi = 68;
```

```
    pi = &c; *pi = 88;
```

```
    return 0;
```

```
}
```



北京大学



# 指向符号常量的指针

## ■ 定义方式

```
int main()
```

```
{
```

```
    const int a = 78; const int b = 28; int c = 18;
```

```
    const int * pi = &a;
```

```
    *pi = 58;                //(error, *p不能被赋值)
```

```
    pi = &b; *pi = 68;
```

```
        //(error, 可以给pi重新赋值, 但 *p不能被赋值)
```

```
    pi = &c; *pi = 88;        (error, *p不能被赋值)
```

```
    return 0;
```

```
}
```



北京大学



# 指向符号常量的指针

## ■ 定义方式

```
int main()
```

```
{
```

```
    const int a = 78; const int b = 28; int c = 18;
```

```
    int * pi = &a;
```

```
    *pi = 58;                //(error, *p不能被赋值)
```

```
    pi = &b; *pi = 68;
```

```
        //(error, 可以给pi重新赋值, 但 *p不能被赋值)
```

```
    pi = &c; *pi = 88;        (error, *p不能被赋值)
```

```
    return 0;
```

```
}
```



北京大学





# 内容提要

## ■ 指针与函数

- ◆ 指针用做函数参数
- ◆ 指针用做函数返回值
- ◆ 指向函数的指针



北京大学





## 返回指针值的函数

- 函数的返回值可以是多种类型

- ◆ 返回整型数据的函数：

- ```
int max( int x, int y );
```

- ◆ 返回指针类型数据的函数

- ```
int *function( int x, int y );
```

- 函数名字前面表示函数的类型 “\*”



北京大学



## 返回指针值的函数

- 打印出第二行第三列的值

```
#include<iostream>
using namespace std;
void main(){
    int a[4][4]=
    {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
    12, 13, 14, 15, 16};
    int *p;
    p = get(a, 2, 3);
    cout<<*p<<endl;
}
```

```
int *get(int arr[ ][4],
          int n, int m)
{
    int *pt;
    pt = *(arr + n - 1) + m-1;
    return(pt);
}
```



北京大学



## 返回指针值的函数

- 注意：确保参数正确！

```
#include<iostream>
using namespace std;
void main(){
    int a[4][4]=
        {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
         11, 12, 13, 14, 15, 16};
    int *p;
    p = get( _____ , 2, 3);
    cout<<*p<<endl;
}
```

```
int *get(int *pt, int n,
         int m)
{
    pt = pt + (n-1) * 4 + m-1;
    return(pt);
}
```



北京大学



## 返回指针值的函数

- 判断程序的执行结果：

```
#include<iostream>
using namespace std;
int *getInt1()
{
    int value1 = 20;
    return &value1;
}
int main(){
    int *p;
    p = getInt1();
    cout << *p << endl;
    return 0;
}
```





# 返回指针值的函数

## ■ 判断程序的执行结果:

```
#include<iostream>
using namespace std;
int main(){
    int *p,*q;
    p = getInt1();
    q = getInt2();
    cout << *p << endl;
    return 0;
}
```

```
int *getInt1()
{
    int value1 = 20;
    return &value1;
}
int *getInt2()
{
    int value2 = 30;
    return &value2;
}
```



# 返回指针值的函数

## ■ 判断程序的执行结果:

```
#include<iostream>
using namespace std;
int main(){
    int *p,*q;
    p = getInt1();
    q = getInt2();
    cout << *p << endl;
    return 0;
}
```

```
int *getInt1()
{
    int value1 = 20;
    return &value1;
}
int *getInt2()
{
    int a[4][4]=
        {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
        11, 12, 13, 14, 15, 16};
    int value2 = 30;
    return &value2;
}
```





## 确保返回地址的意义

- 返回一个处于生命周期中的变量的地址
  - ◆ 返回全局变量的地址，而非局部变量的地址

```
#include<iostream.h>
int value1 = 20;
int value2 = 30;
int main()
{  int *p,*q;
   p = getInt1();
   q = getInt2();
   cout << *p << endl;
   return 0; }
```

```
int *getInt1()
{
    return &value1;
}

int *getInt2()
{
    return &value2;
}
```





## 确保返回的地址意义

### ■ 返回一个处于生命周期中的变量的地址

- ◆ 返回静态局部变量的地址，而非动态局部变量的地址

```
#include<iostream>
using namespace std;
int main(){
    int *p,*q;
    p = getInt1();
    q = getInt2();
    cout << *p << endl;
    return 0;
}
```

```
int *getInt1()
{
    static int value1 = 20;
    return &value1;
}
int *getInt2()
{
    auto int value2 = 30;
    return &value2;
}
```





# 什么是静态局部变量

## ■ 静态局部变量

- ◆ 有时希望函数中的局部变量的值在函数调用结束后不消失而保留原值，即其占用的存储单元不释放，在下一次该函数调用时，仍可以继续使用该变量；
- ◆ 用关键字static进行声明，可将变量指定为“静态局部变量”。

**static** int value1 = 20;



北京大学

```

#include<iostream.h>
void function( )
{
    auto int a = 0;
    static int b = 0;
    a = a+1;
    b = b+1;
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
}
void main()
{
    for(int i = 0; i < 5; i++)
    {
        function( );
        cout<<"Call Again!"<<endl;
    }
}

```

```

a = 1
b = 1
Call Again!
a = 1
b = 2
Call Again!
a = 1
b = 3
Call Again!
a = 1
b = 4
Call Again!
a = 1
b = 5
Call Again!
Press any key to continue

```



# 什么是静态局部变量

## ■ 动态局部变量

- ◆ 如果没有特别说明，一般定义的内部变量都是“动态局部变量”；
- ◆ 在调用该函数时系统会给它们分配存储空间，在函数调用结束时就自动释放这些存储空间。
- ◆ 可选用关键字“**auto**”

**auto** int value2 = 30;

**auto**不写则隐含确定为“自动存储类别”，它属于动态存储方式。



北京大学



# auto vs. static

```
#include<iostream.h>
```

```
int *getInt1()
```

```
{    static int value1 = 20;  
    return &value1; }
```

```
int *getInt2()
```

```
{    auto int value2 = 30;  
    return &value2; }
```

```
int main()
```

```
{  int *p,*q;  
    p = getInt1();  
    q = getInt2();  
    cout << *p << endl;  
    return 0; }
```

■ **动态变量**  
value2的存  
在范围

■ **静态变量**  
value1存在  
的范围



北京大学



## 确保返回的地址意义

- 判断程序返回的结果:

```
#include<iostream>
using namespace std;
int main(){
    int *p,*q;
    p = getInt1();
    q = getInt2();
    cout << *p << endl;
    return 0;
}
```

```
int *getInt1()
{
    static int value1 = 20;
    return &value1;
}
int *getInt2()
{
    auto int value2 = 30;
    return &value2;
}
```



北京大学



# 内容提要

## ■ 指针与函数

- ◆ 指针用做函数参数
- ◆ 指针用做函数返回值
- ◆ 指向函数的指针

## ■ 指针与结构体



北京大学



# 指向函数的指针

## ■ 已知

- ◆ C++程序中的一个函数，编译器进行编译时会给函数分配一个入口地址。
- ◆ 该地址被称为“函数的地址”或“函数的指针”。
- ◆ C++语言规定，函数的名字代表“函数的地址”。

## ■ 可以

- ◆ 定义一个指针变量**pointer**，并让该变量指向某个特定的函数；
- ◆ 指针变量的类型应与函数的类型相同；
  - 函数的类型 由它的返回值和参数表决定



北京大学





## 指向函数的指针

```
void main( )  
{  int max(int, int);    //用于声明的函数原型  
    int (*p)(int, int);  
        // 定义一个指向整型函数的指针变量;  
    int a, b, c;  
    p = max; //给“整型函数型”指针变量p赋值;  
    cin>>a>>b;  
    c = p(a, b); //利用指针变量p调用函数  
    cout<<"a= "<<a<<", b="<<b<<", max="<<c;  
}
```







# 指向函数的指针

```
void main( )  
{  int max(int, int);    //用于声明的函数原型  
    int (*p)(int, int);  
        // 定义一个指向整型函数的指针变量;  
    int a, b, c;  
    p = max; //给“整型函数型”指针变量p赋值;  
    cin>>a>>b;  
    c = (*p)(a, b);    //利用指针变量p调用函数  
    cout<<"a= "<<a<<", b="<<b<<", max="<<c;  
}
```





# 指向函数的指针

## ■ `int (*p)( int, int)` 的意义是什么？

### ◆ 对比

● `int max ( int, int )`

● `int ( *p )( int, int )`

◆ 指针变量`p`的值是函数`max`的入口地址

## ■ 无意义的操作

◆ `p = max( a, b );`

◆ `p++, p--; ++p, --p;`



北京大学



## 指向函数的指针 的赋值

```
#include<iostream>
using namespace std;
int fn1(char x, char y) {return x+y;}
int fn2(int a)    {return a;}
int (*fp1)(char x, char y);
int (*fp2)(int s);
void main()
{ fp1 = fn1;      //ok
  fp2 = fn2;      //ok
  fp1 = fn2;      //error类型不一致
  fp2 = fp1;      //error类型不一致
  fp2 = fn2(5);   //error, fn2(5)不是函数的地址
  cout<<(*fp2)(5)<<endl;
  cout<<fp2(5)<<endl;
}
```





## 用指向函数的指针做函数参数

- **例:** 设一个函数`process`, 在调用它的时候, 可以根据输入参数的不同, 完成下列三个函数的功能。输入`max`时找出`a`和`b`中大者, 输入`min`时找出其中小者, 输入`add`时求`a`与`b`之和。

```
max(int x, int y)
{
    int Z;
    if(x > y) Z = x;
    else Z = y;
    return(Z);
}
```

```
min(int x, int y)
{
    int Z;
    if(x < y) Z = x;
    else Z = y;
    return(Z);
}
```

```
add(int x, int y)
{
    int Z;
    Z = x + y;
    return(Z);
}
```





# 用指向函数的指针做函数参数

```
process(int x, int y, int (*fun)(int, int))
```

/\* **int (\*fun)(int,int)** 表示**fun是指向函数的指针**，该函数是一个整型函数，有两个整型形参\*/

```
{  
    int reSult;  
    reSult = (*fun)(x, y);  
    cout<<reSult<<endl;  
}
```

```
void main()  
{   int a, b;  
    cin>>a>>b;  
    process(a, b, max);  
    process(a, b, min);  
    process(a, b, add);  
}
```



# 几个问题



北京大学



# 关于函数库的使用

## ■ 使用早期的函数库

**C传统方法**

```
#include <stdio.h>
```

```
#include <math.h>
```

```
#include <string.h>
```

**C++新方法**

```
#include <cstdio>
```

```
#include <cmath>
```

```
#include <cstring>
```

```
using namespace std;
```

## ■ 使用C++新的函数库

```
#include<iostream>
```

```
#include<math>
```

```
#include<string>
```

```
using namespace std;
```



北京大学





## 关于\*\*p与(\*p)[4]

■ 输入 i, j; 输出 a[i][j];

main( )

{

int a[3][4]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23};

int \*\*p, i, j; //错误

p = a ;

cin>>i>>j;

cout<<setw(4)<<p[i][j];

}



北京大学





## 关于\*\*p与(\*p)[4]

■ 输入 i, j; 输出 a[i][j];

main( )

{

int a[3][4]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23};

int (\*p)[4], i, j;

p = a ;

cin>>i>>j;

cout<<setw(4)<<p[i][j];

}



北京大学



好好想想，有没有问题？

谢谢！



清华大学