计算机程序设计

刘龙军 副教授

2025年10月13日

目录

指针

天使与魔鬼的化身

—指针

天使与魔鬼的化身—指针

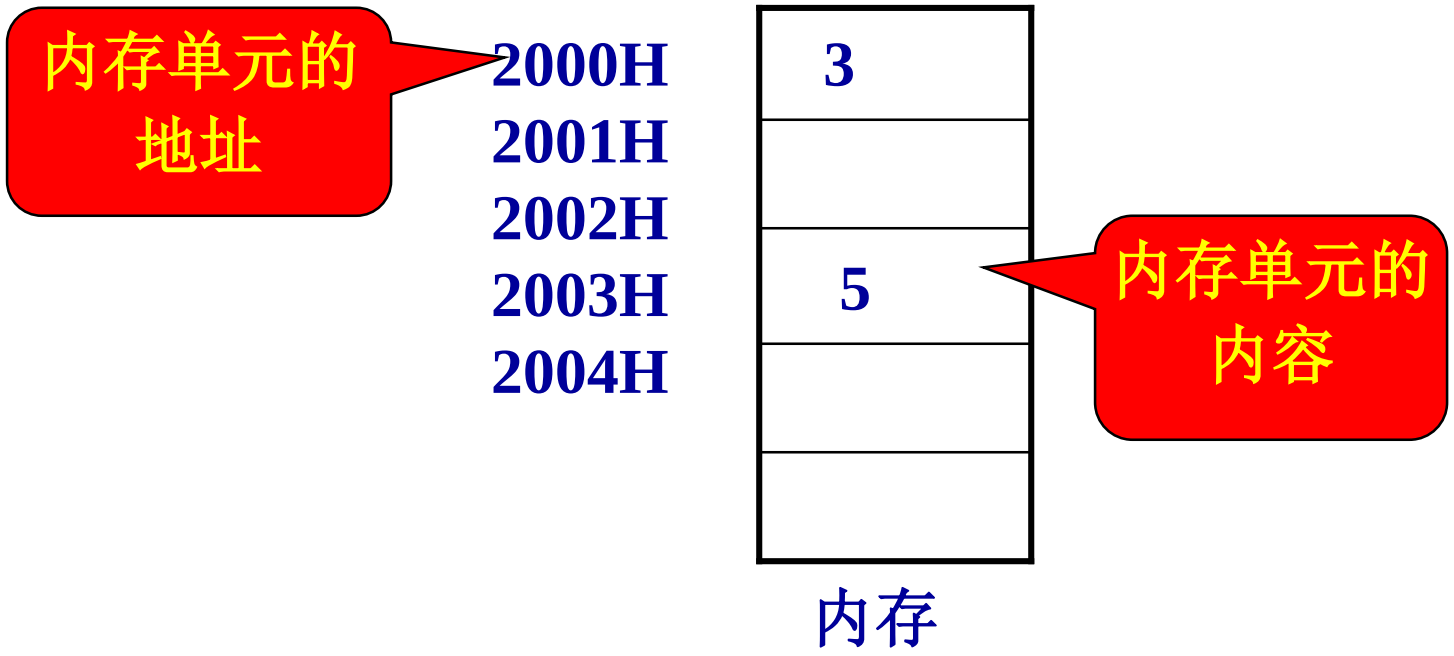
- 指针: 铁杆C/C++程序员最挚爱的武器
- 指针造就了C/C++的高效和强大，很多不可能的任务由指针完成（传递）
- 黑客攻击服务器利用的bug绝大部分都是指针和数组造成的
- “该程序执行了非法操作，即将关闭” 这种错误几乎全是由指针和数组导致的

内存地址

- **内存**：计算机内的存储部件，活动中程序指令和数据都保存在内存中
- 速度快、但是掉电即失
- 内存中的**每个字节都有唯一的一个地址，地址是一个无符号整数**（通常用16进制数）
- 只要指明要**访问的内存单元的地址，就可以立即访问到该单元**
- 可以认为：**地址和指针是同义词，变量的指针就是变量的地址**

数据在内存中是如何存取的？

- 系统根据程序中定义变量的类型，给变量分配一定的长度空间。字符型占1个字节，整型数占4个字节……。内存区的每个字节都有编号，称之为地址。



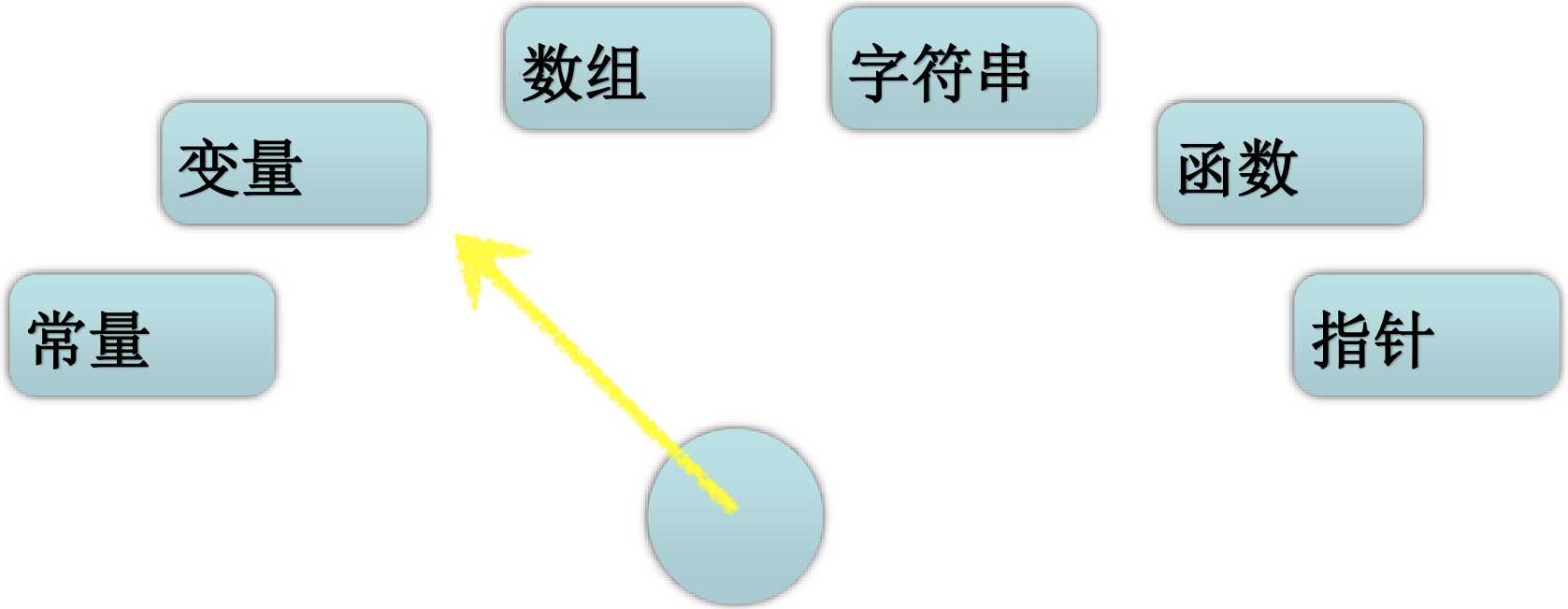
举例

- 某同学去找老师答疑
- 老师以前在课堂上留的**办公室地址**是主楼202房间
- 某学生到202房间后，发现开门的是陌生人，从他那里得知该代课老师**办公室换到了326房间**。
- 该学生上到三楼去326房间找到了代课老师。

- 该老师藏在一个内存地址单元中，地址是326。
- 地址326又由另一单元P所指认，P单元的地址为202。
- 该老师的直接地址是326，326的间接地址是202，因为202单元中存的是直接地址326。
- 我们称P为指针变量，326是指针变量的值，实际上是有用数据藏在地址为326的存储器中。
-

指针的概念

- **指针变量**：用来存放另一变量地址的变量（内存地址），简称指针
- 指针是对**一种特殊变量的称呼**，其特殊性表现在类型（运算）和值上。
-



指针定义

- 指针既然是变量，所以也具有变量的三个要素：
  - 变量的名称。命名规则与一般变量一样，是由英文字符或下划线开始的。
  - 变量的类型。“X型指针”、“指向X类型的指针”。
  - 变量的值。有特殊含义---是某个内存地址。

-

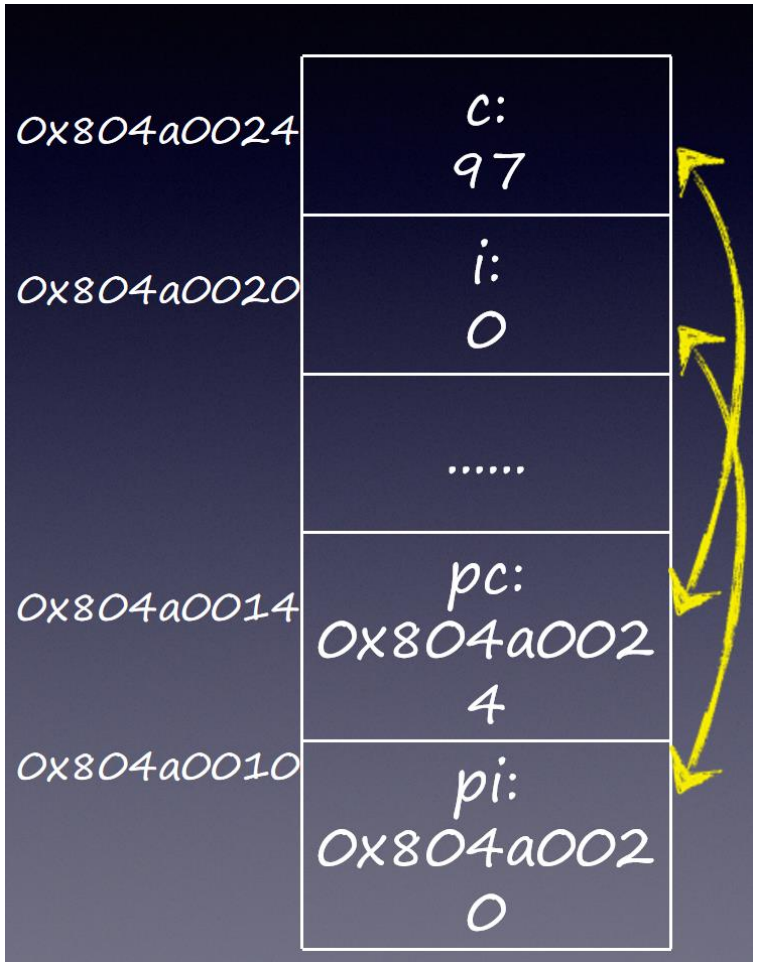
类型 *标识符

```
int *pi, *pc;
   int* pn;
```

指针所指变量

- 这里的&是取地址运算符，&i表示取变量i的地址
- int *pi = &i;表示定义一个指向int型的指针变量pi，并用i的地址来初始化pi
- pi是变量, int *是类型
- 变量占用内存空间，pi的大小是sizeof(int *)

```
int i = 0;
int *pi = &i;
char c =’a’;
char *pc = &c;
```



示例

- donuts value = 6 and donuts address = 0x7fff5fbff778
- cups value = 4.5 and cups address = 0x7fff5fbff770

```
// address.cpp -- using the & operator to find addresses
#include <iostream>
int main()
{
    using namespace std;
    int donuts = 6;
    double cups = 4.5;

    cout << "donuts value = " << donuts;
    cout << " and donuts address = " << &donuts << endl;
    // NOTE: you may need to use unsigned (&donuts)
    // and unsigned (&cups)
    cout << "cups value = " << cups;
    cout << " and cups address = " << &cups << endl;
    // cin.get();
    return 0;
}
```

空指针

- NULL是一个等于0的常量，在头文件stddef.h中定义，它是唯一的一个允许直接赋值给指针的整数值
- 表示指针不指向任何内存地址
- 防止其指向任何未知的内存区域

```
int *p = NULL;
```

指针的一些知识点概况

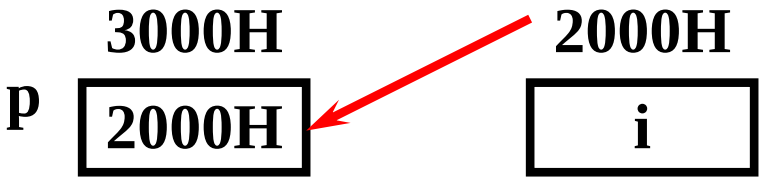
关于指针的原则

- 学习原则
  - 一定要学会
  - 其实通常的应用很简单，就是一个变量
  - 复杂的应用结合后面应用再深入理解
- 使用原则
- 永远要清楚每个指针指向了哪里

访问

- 直接访问
  - 按变量地址存取变量的值。cin>>i;　实际上放到定义 i 单元的地址中

- 间接访问
  - 将变量的地址存放在另一个单元p中，通过 p 取出变量的地址，再针对变量操作。



一个变量的地址称为该变量的指针。

例子

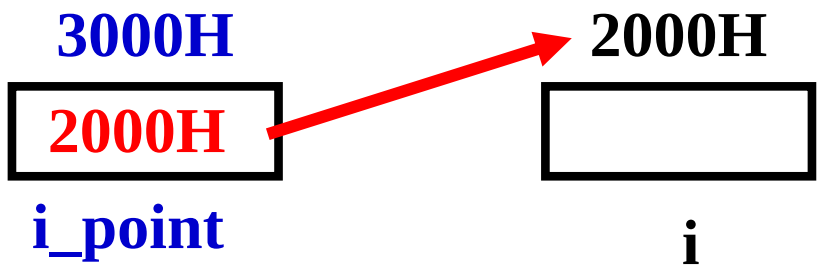
- 变量的指针和指向变量的指针变量
- 变量的指针就是变量的地址，当变量定义后，其指针（地址）是一常量。

int i;　　&i ： 2000H

- 可以定义一个变量专门用来存放另一变量的地址，这种变量我们称之为**指针变量**。在编译时同样分配一定字节的存储单元，未赋初值时，该存储单元内的值**是随机的**。

指针的赋值：

```
int  i,  *i_point;
i_point=&i;
```



也可以在定义指针变量时赋初值：

```
int  i;
int   *i_point=&i;
```

一个指针变量只能指向同一类型的变量。即整型指针变量只能放整型数据的地址，而不能放其它类型数据的地址。

* 在定义语句中只表示变量的类型是指针，没有任何计算意义。

* 在语句中表示“指向”。&表示“地址”。

取地址运算符&

- &运算符用来取被操纵对象的地址
- 例如：
  - &x表示变量x在内存的存放地址，若x的地址是0xbfff0010, 则&x的值就是0xbfff0010。

取值运算符 *取值运算符 *

- * 运算访问指针所指变量。
- 例如：
  - x=*p是将指针p所指变量的值赋给x
  - *p=5是将5送入指针p所指变量中

& 与 * 的关系

- &和*互为逆运算
- &是取地址运算符，操作对象是变量，&运算的结果指向该变量的指针
- *是取值运算符，操作对象是地址，*运算访问地址表达式所指变量

```
// pointer.cpp -- our first pointer variable
#include <iostream>
int main()
{
    using namespace std;
    int updates = 6;          // declare a variable
    int * p_updates;          // declare pointer to
an int
    p_updates = &updates;     // assign address of
int to pointer
// express values two ways
    cout << "Values: updates = " << updates;
    cout << ", *p_updates = " << *p_updates <<
endl;
// express address two ways
    cout << "Addresses: &updates = " << &updates;
    cout << ", p_updates = " << p_updates <<
endl;
// use pointer to change value
    *p_updates = *p_updates + 1;
    cout << "Now updates = " << updates << endl;
    return 0;
}
```

```
Values: updates = 6, *p_updates = 6
Addresses: &updates = 0x7fff5fbff768, p_updates = 0x7fff5fbff768
Now updates = 7
```

指针赋值

```
//pointer_init.cpp
//演示指针的赋值
#include <iostream>
int main( )
{
    using namespace std;
    int a = 66;
    int *p=NULL;
    int *q=NULL;

    p = &a;  //将变量 a 的地址赋给 p ---- 画内存图
    q = p;       // 将 p 的值赋给 q
    cout<<a<<endl;
    cout<<*p<<endl;
    cout<<*q<<endl;

    return 0;
}
```

```
p = a;
q = 66;
```



```
66
66
66
```

指针new和delete

```
//pointer_init2.cpp
//演示指针的赋值
#include <iostream>
int main( )
{
   using namespace std;

   int *p= new int;

    *p = 66;  //将变量 a 的地址赋给 p

    cout<<*p<<endl;

    delete p;

    return 0;
}
```

```
int *p;
 *p = 66;WHY
```

- 指针变量没有赋上全值之前，不要用*改变取值， *其实在改变另一个实体数据类型的值
- 此语句会在堆上为指定类型的变量分配足够的内存，并返回指向该内存的指针。如果内存分配成功，这个指针可以用于访问新分配的内存。如果内存分配失败（例如，因为堆空间不足）
- 三种安全赋值方法
  - NULL
  - 赋初始化过的变量地址
  - new-delete

动态编程的思想 （与数组联系）

指针运算法则

- 只能进行加减和关系运算
  - +、-
  - > < >= <= == !=
- 只能同类型指针之间或指针与整数之间运算

指针的加法（+）运算

- p++表示p=p+1

- 如果指针指向数组变量，允许对指针值加上一个整数表达式。
- 例如

  p = &a[5],则q=p+3将使q指向a[8]

指针的减法（-）运算

- p--表示p=p-1
- 如果指针指向数组变量，允许对指针值减去一个整数表达式。
- 允许两个指针值相减，得到的结果是两个指针之间的距离
- 指针的加减运算是以其指向的类型的字长为单位的

关系运算

- == 和 !=
  - 判断两个指针的值是否相等和不相等
  - 例如：
    - pi == NULL; pi != NULL
    - px == py;
- > , >= 和<=
  - 判断两个指针值的大小关系
  - 例如：
    - px < py 判断px所指向的地址是否小于py所指向的地址

指针与数组

- 指针与数组有着密切的关系
  - 数组名是数组的首地址，指针值也是一个地址
  - 如果指针指向数组的首地址，例如p指向a[0]，则p与a表示的是同一个对象（内容）
- 事实上，在C/C++中把数组名可以看作常量指针
- 指针也可当作数组名使用

指针与数组等价之处

- 注意：
  - a[i],
  - p[i],
  - *(a+i),
  - *(p+i)

  等价

```
//array_ptr.cpp
#include <iostream>

using namespace std;

int main( )
{
    int i=0;
    int *p = NULL;
    int a[5]={0,1,2,3,4};

    p = a;

    for(i=0; i<5; i++)
    {

cout<<i<<"\t"<<a[i]<<"\t"<<p[i]<<"\t"<<*(a+i)<<"\t"<<*(p+i)<<endl;
    }

    return 0;
}
```

指针与数组不同之处

- 指针是一个变量
- 数组名不是一个变量，是一个常量指针
- 定义数组分配空间由类型和下标决定，定义指针仅仅分配指针类型空间

```
int a[]={1,2,3};
int *p=a;

p++;
```

```
int a[]={1,2,3};
int *p;

a = p;
a++;
```

例子

- 例8.6 有一个整型数组a，有10个元素，要求输出数组中的全部元素。
- 三种方法：引用数组中各元素的值有3种方法：
  - (1)下标法；
  - (2)通过数组名计算数组元素地址，找出元素的值；
  - (3) 用指针变量指向数组元素
- 分别写出程序，以资比较分析。

(1) 下标法

```
#include <stdio.h>
int main()
{ int a[10];  int i;
   printf( “enter 10 integer numbers:\n");
   for(i=0;i<10;i++) scanf("%d",&a[i]);
   for(i=0;i<10;i++)  printf( “%d ” ,a[i]);
   printf("%\n");
   return 0;
 }
```

```
enter 10 integer numbers:
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

(2) 通过数组名计算数组元素地址，找出元素的值

```
#include <stdio.h>
int main()
{ int a[10];  int i;
   printf( “enter 10 integer numbers:\n");
   for(i=0;i<10;i++)  scanf("%d",&a[i]);
   for(i=0;i<10;i++)
         printf( “%d ” ,*(a+i));
   printf("\n");
   return 0;
 }
```

scanf("%d",a+i);

(3) 用指针变量指向数组元素

```
#include <stdio.h>
int main()
{ int a[10];  int *p,i;
   printf( “enter 10 integer numbers:\n");
   for(i=0;i<10;i++)  scanf("%d",&a[i]);
   for(p=a;p<(a+10);p++)
       printf( “%d ” ,*p);
   printf("\n");
   return 0;
}
```

```
for(p=a;p<(a+10);p++)
```

```
for(p=a;p<(a+10);a++)
    printf(“%d ”,*a); 错！
```

3种方法的比较：

- 第(1)和第(2)种方法执行效率相同
  - C编译系统是将a[i]转换为*(a+i)处理的，即先计算元素地址。
  - 因此用第(1)和第(2)种方法找数组元素费时较多。
- 第(3)种方法比第(1)、第(2)种方法快
  - 用指针变量直接指向元素，不必每次都重新计算地址，像p++这样的自加操作是比较快的
  - 这种有规律地改变地址值(p++)能大大提高执行效率

3种方法的比较：

- 用下标法比较直观，能直接知道是第几个元素。
- 用地址法或指针变量的方法不直观，难以很快地判断出当前处理的是哪一个元素。

例子

- 通过指针变量输出整型数组a的10个元素。
- 解题思路：

用指针变量p指向数组元素，通过改变指针变量的值，使p先后指向a[0]到a[9]各元素。

例子代码

```
#include <stdio.h>
int  main()
{ int *p,i,a[10];
   p=a;
   printf( “enter 10 integer numbers:\n");
   for(i=0;i<10;i++) scanf( “%d” ,p++);
   for(i=0;i<10;i++,p++)
        printf( “%d ” ,*p);
   printf("\n");
   return 0;
}
```

重新执行
p=a;

退出循环时p指向a[9]
后面的存储单元

因此执行此
循环出问题

指针与动态数组 new[ ] 和delete[ ]

- Type* pointer = new Type [number];
- int* p = new int(10); // 在堆上创建一个 int，并初始化为 10
- int* arr = new int[10]; // 在堆上创建一个包含 10 个 int 的数组
- delete p; // 释放 p 指向的内存
- delete[] arr; // 释放动态数组
- 如果你使用 new 在堆上分配了内存，你必须记住使用 delete 来释放这个内存。否则，你的程序可能会出现内存泄漏

```
//pointer_init2.cpp
//演示指针的赋值
#include <iostream>
int main( )
{
   using namespace std;

   int *p= new int;

    *p = 66;  //将变量 a 的地址赋给 p

    cout<<*p<<endl;

    delete p;

    return 0;
}
```

动态编程的思想 （与数组联系）

指针作为函数参数

- 指针既然是数据类型，自然可以做函数的参数和返回值的类型值的类型
- 指针作为参数的经典例子：
- 这里的函数调用过程还是“实参”的内容复制到“形参”

```
//swap_prt.cpp
#include <iostream>
using namespace std;\

void Swap(int*, int*);

int main()
{
    int a=123;
    int b=456;

    Swap(&a,&b);

    cout<<"a\t"<<a<<endl;
    cout<<"b\t"<<b<<endl;

    return 0;
}


void Swap(int* x, int* y)
{
    int tmp;
    tmp = *x;
    *x = *y;
    *y= tmp;
 }
```

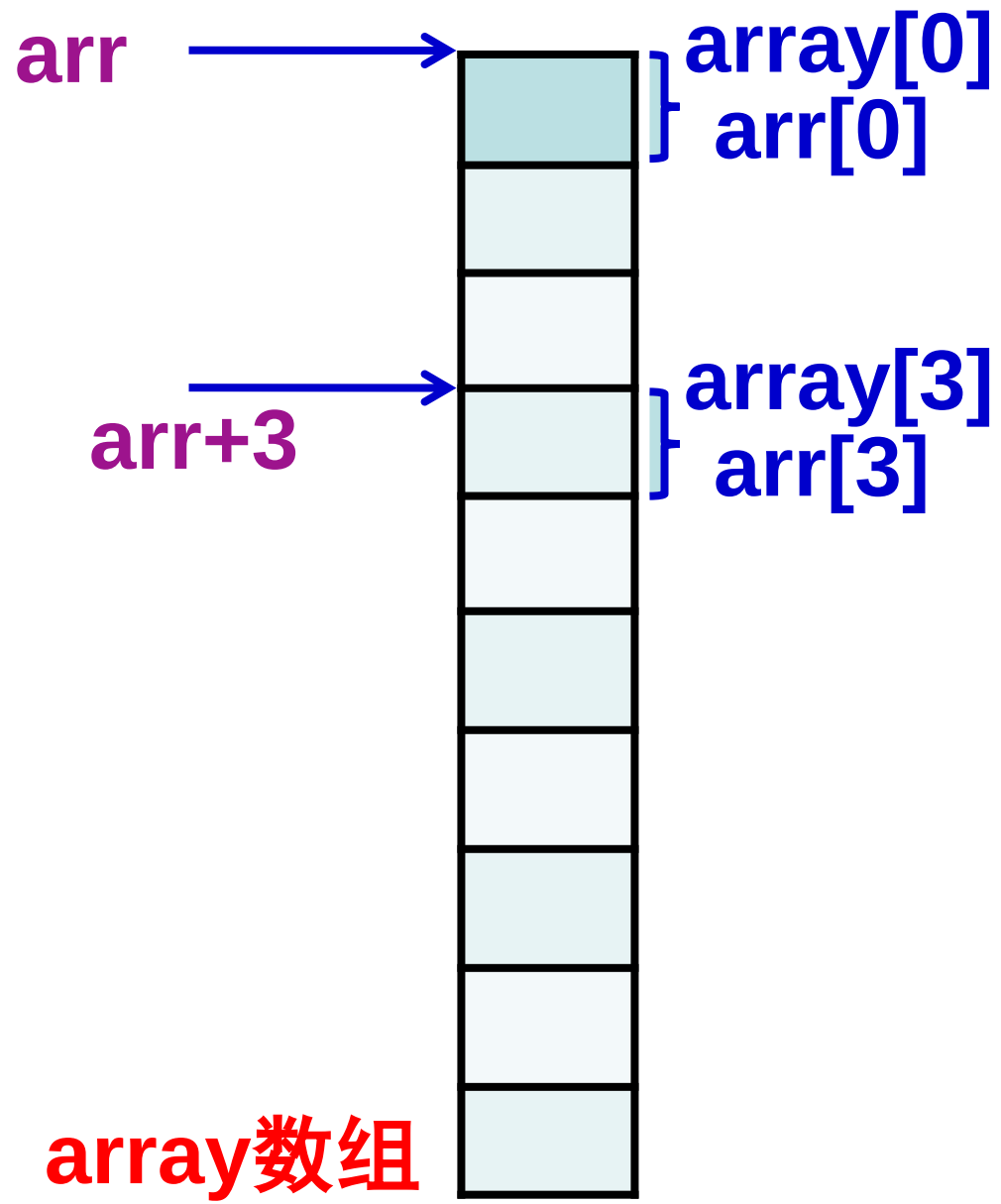
数组与指针例子

```
void fun(int arr[],int n];
int main()
{
   int array[10];
    ┇
   fun (array,10);
   return 0;
}
void fun(int arr[ ],int n)
{ ┇  }
```

fun(int *arr,int n)

例子

```
void fun(int *arr,int n);
int main()
{    int array[10];
     ┆
   fun (array,10);
   return 0;
}
void fun(int *arr,int n)
{ ┆  }
```



小结

- 实参数组名是指针常量，但形参数组名是按指针变量处理
- 在函数调用进行虚实结合后，它的值就是实参数组首元素的地址
- 函数执行期间，形参数组可以再被赋值

```
void fun (int arr[ ],int n)
{
  cout << *arr <<endl;
    arr = arr + 3;
    cout << *arr <<endl;
}
```

例子


```
void inv(int *x,int n);
int main()
{
   int i, arr[10],*p=arr;
   for(i=0;i<10;i++,p++)
       scanf(“%d”,p);
   inv(p,10);
   for(p=arr;p<arr+10;p++)
       printf(“%d ”,*p);
   printf("\n");
   return 0;
}
```

不可少！！！

例子

```
void sort(int x[ ],int n);
int main()
{
   int i,*p, a[10];
   p=a;
   for(i=0;i<10;i++)  scanf( “%d” ,p++);
   p=a;
   sort(p,10);
   for(p=a,i=0;i<10;i++)
   { printf( “%d ” ,*p);   p++;   }
   printf("\n");
   return 0;
}
```

例子

```
void sort(int x[],int n)
{ int i,j,k,t;
   for(i=0;i<n-1;i++)
   { k=i;
      for(j=i+1;j<n;j++)
       if(x[j]>x[k]) k=j;
          if(k!=i)
           { t=x[i];x[i]=x[k];x[k]=t; }
    }
}
```

void sort(int *x,int n)

if (*(x+j)>*(x+k)) k=j;

{t=*(x+i);*(x+i)=*(x+k);*(x+k)=t;}

```
12 34 5 689 -43 56 -21 0 24 65
689 65 56 34 24 12 5 0 -21 -43
```

指向函数的指针

- 函数指针
- 用函数指针变量调用函数
- 怎样定义和使用指向函数的指针变量
- 用指向函数的指针作函数参数

定义

- 如果在程序中定义了一个函数，在编译时，**编译系统为函数代码分配一段存储空间，这段存储空间的起始地址，称为这个函数的指针。**
- 可以定义一个**指向函数的指针变量，用来存放某一函数的起始地址，这就意味着此指针变量指向该函数**。例如：

```
int (*p)(int,int);
```

- 定义p是指向函数的指针变量，它可以指向类型为整型且有两个整型参数的函数。p的类型用int (*)(int,int)表示

用函数指针变量调用函数

- 用函数求整数a和b中的大者。
- 解题思路：定义一个函数max，实现求两个整数中的大者。**在主函数调用max函数，除了可以通过函数名调用外，还可以通过指向函数的指针变量来实现**。分别编程并作比较。

例子

（1）通过函数名调用函数

```
int max(int,int);
int main()
{
   int a,b,c;
   printf("please enter a and b:");
   scanf("%d,%d",&a,&b);
   c=max(a,b);
   printf( “%d,%d,max=%d\n",a,b,c);
   return 0;
}
```

例子

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

例子

(2)通过指针变量访问它所指向的函数

```
int max(int,int);
int main()
{
   int (*p)(int,int);  int a,b,c;
   p=max;
   printf("please en
   scanf("%d,%d",&
   c=(*p)(a,b);
   printf( “%d,%d,max=%d\n",a,b,c);
   return 0;
}
```

只能指向函数返回值为整型且有两个整型参数的函数

必须先指向，若写成 p=max(a,b); 错

怎样定义与使用

- 定义指向函数的指针变量的一般形式为
- 数据类型 (*指针变量名)(函数参数表列);

如 int (*p)(int,int);

p=max;　对

p=max(a,b); 错

p+n,p++,p--等运算无意义

例子

- 输入两个整数，然后让用户选择1或2，选1时调用max函数，输出二者中的大数，选2时调用min函数，输出二者中的小数。
- 解题思路：定义两个函数max和min，分别用来求大数和小数。在主函数中根据用户输入的数字1或2，使指针变量指向max函数或min函数。

例子

```
int max(int,int);   int min(int x,int y);
int main()
{
   int a,b,c,n;
   int (*p)(int,int);
  scanf("%d,%d",&a,&b);     scanf(“%d”,&n);
  if (n==1) p=max;
  else if (n==2) p=min;
  c=(*p)(a,b);
  printf("a=%d,b=%d\n",a,b);
  if (n==1) printf("max=%d\n",c);
  else  printf("min=%d\n",c);
  return 0;
}
```

只看此行看不出调用哪函数

例子

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

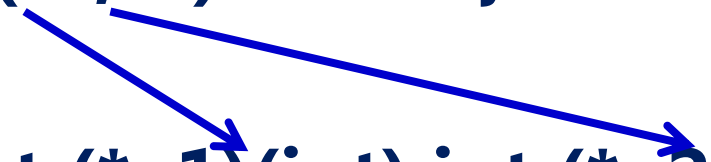
用指向函数的指针作函数参数

- 指向函数的指针变量的一个重要用途是把函数的地址作为参数传递到其他函数
- 指向函数的指针可以作为函数参数，把函数的入口地址传递给形参，这样就能够在被调用的函数中使用实参函数

指向函数的指针作函数参数

```
……
int main()
{  ……   fun(f1,f2)  ……   }

void  fun(int (*x1)(int),int (*x2)(int,int))
{  int a,b,i=3,j=5;
    a=(*x1)(i);        相当于a=f1(i);
    b=(*x2)(i,j);      相当于b=f2(i,j);
}
```



例子

- 有两个整数a和b，由用户输入1,2或3。如输入1，程序就给出a和b中大者，输入2，就给出a和b中小者，输入3，则求a与b之和。
- 解题思路：现在用一个函数fun来实现以上功能。

例子

```
void fun(int x, int y, int (*p)(int,int));
int max(int,int);  int min(int,int);  int
   add(int,int);
int main()
{
  int a=34,b=-21,n;
  printf("please choose 1,2 or 3:");
  scanf( “%d” ,&n);
  if (n==1)        fun(a,b,max);
  else if (n==2) fun(a,b,min);
  else if (n==3) fun(a,b,add);
  return 0;
 }
```

例子

```
void fun(int x,int y,int (*p)(int,int))
{ int result;
   result=(*p)(x,y);
   printf("%d\n",result);
}
int max(int x,int y)
{ int z;
   if(x>y)  z=x;
   else   z=y;
   printf("max=" );
   return(z);
  }
```

输入的选项为1时

相当于max(x,y)

例子

```
void fun(int x,int y,int (*p)(int,int))
{ int result;
   result=(*p)(x,y);
   printf("%d\n",result);
}
int max(int x,int y)
{ int z;
   if(x>y)  z=x;
   else   z=y;
   printf("max=" );
   return(z);
  }
```

输入的选项为2时

相当于min(x,y)

例子

```
int fun(int x,int y,int (*p)(int,int))
{ int result;
   result=(*p)(x,y);
   printf("%d\n",result);
}
int max(int x,int y)
{ int z;
   if(x>y)  z=x;
   else   z=y;
   printf("max=" );
   return(z);
  }
```

输入的选项为3时

相当于add(x,y)

例子

```
int min(int x,int y)
{ int z;
   if(x<y) z=x;
   else z=y;
   printf("min=");
   return(z);
}
int add(int x,int y)
{ int z;
   z=x+y;
   printf("sum=");
   return(z);
}
```

返回指针值的函数

- 一个函数可以返回一个整型值、字符值、实型值等，也可以返回指针型的数据，即地址。其概念与以前类似，只是返回的值的类型是指针类型而已
- 定义返回指针值的函数的一般形式为
- 类型名 *函数名(参数表列);

例子

- 有a个学生，每个学生有b门课程的成绩。要求在用户输入学生序号以后，能输出该学生的全部成绩。**用指针函数实现（参数含指针参数、返回值含指针参数）。**

例子

- 解题思路：
  - 定义二维数组score存放成绩
  - 定义输出某学生全部成绩的函数search，它是返回指针的函数，形参是行指针和整型
  - 主函数将score和要找的学号k传递给形参
  - 函数的返回值是&score[k][0](k号学生的序号为0的课程地址)
  - 在主函数中输出该生的全部成绩

例子

```
float  *search(float (*pointer)[4],int n);
int main()
{
    float score[ ][4]={{60,70,80,90},
       {56,89,67,88},{34,78,90,66}};
    float  *p;  int i,k;
  scanf( “%d” ,&k);
  printf("The scores of No.%d are:\n",k);
  p=search(score,k);
  for(i=0;i<4;i++)
     printf( “%5.2f\t” ,*(p+i));
  printf("\n");
  return 0;  }
```

返回k号学生课程首地址

例子

```
float *search(float (*pointer)[4],int n)
{ float *pt;
   pt=*(pointer+n);     //指针寻址，类别下标寻址
   return(pt);
}
```

指针数组作main函数的形参

- main函数的形参
- 在以往的程序中，main函数的第一行一般写成以下形式：int main()或 int main(void) 表示main函数没有参数，调用main函数时不必给出实参。
- 实际上，在某些情况下，main函数可以有参数，例如
- int main(int argc, char *argv[ ]) //argument
- 其中，argc和argv就是main函数的形参，它们是程序的“命令行参数”。
- argv是*char指针数组，数组中每一个元素(其值为指针)指向命令行中的一个字符串。

指针数组作main函数的形参

- 通常main函数和其他函数组成一个文件模块，有一个文件名。
- 对这个文件进行编译和连接，得到可执行文件（后缀为.exe）。用户执行这个可执行文件，操作系统就调用main函数，然后由main函数调用其他函数，从而完成程序的功能。

指针数组作main函数的形参

- main函数的形参是从哪里传递给它们的呢？
- 显然形参的值不可能在程序中得到。
- main函数是操作系统调用的，实参只能由操作系统给出。（OS与APP）
- 第一个参数，int型的argc，为整型，用来统计程序运行时发送给main函数的命令行参数的个数
- 第二个参数，char*型的argv[]，为字符串数组，用来存放指向的字符串参数的指针数组，
- argv[0]指向程序运行的全路径名
- argv[1]指向在DOS命令行中执行程序名后的第一个字符串
- argv[2]指向执行程序名后的第二个字符串

本章结束！