6.1 指针基础

6.1.1 指针的基本概念

6.1.2 指针变量的定义与使用

6.1.3 指针变量做函数参数

6.1.4 指针与数组

6.1 指针基础

6.1.1 指针的基本概念

内存地址和内存内容

内存地址

内存中每一字节的编号，称为地址

32位系统，可支持2^{32}字节内存，即4G内存

地址范围：0x00000000-0xffffffff

16位：$0 \sim 2^{16}-1$

64位：$0 \sim 2^{64}-1$

6.1 指针基础

6.1.1 指针的基本概念

内存地址和内存内容

内存地址

内存中每一字节的编号，称为地址

32位系统，可支持2^{32}字节内存，即4G内存

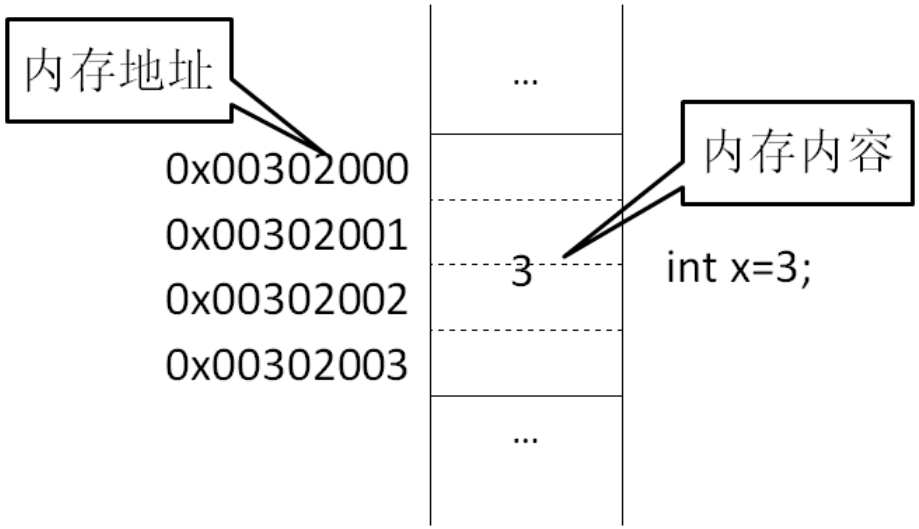
地址范围：0x00000000-0xffffffff

16位：$0 \sim 2^{16}-1$

64位：$0 \sim 2^{64}-1$

内存内容

以字节为单位，用一个或几个字节来表示的数据值



6.1 指针基础

6.1.1 指针的基本概念

地址运算符 &　取地址

例：

```
#include <iostream>
using namespace std;
int main()
{
    int donuts = 6;
    double cups = 4.5;
    cout << "donuts value = " << donuts;
    cout << " and donuts address =" << &donuts << endl;
    cout << "cups value = " << cups;
    cout << " and cups address =" << &cups << endl;
    return 0;
}
```

6.1 指针基础

6.1.1 指针的基本概念

地址运算符 &　取地址

例：

```
#include <iostream>
using namespace std;
int main()
{
    int donuts = 6;
    double cups = 4.5;
    cout << "donuts value = " << donuts;
    cout << " and donuts address =" << &donuts << endl;
    cout << "cups value = " << cups;
    cout << " and cups address =" << &cups << endl;
    return 0;
}
```

VS2019运行示例

```
donuts value = 6 and donuts address =0046FD6C
cups value = 4.5 and cups address =0046FD5C
```

Dev C++运行示例

```
donuts value = 6 and donuts address =0x28fecc
cups value = 4.5 and cups address =0x28fec0
```

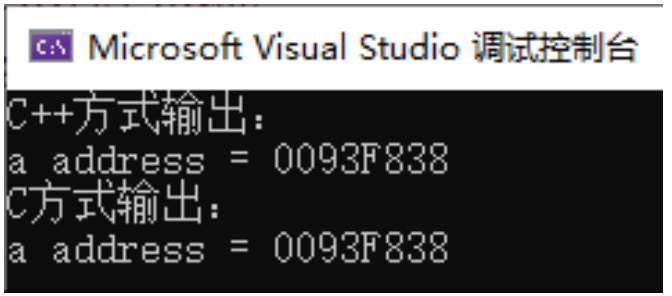
不同系统地址表示格式、数值都可能不同

6.1 指针基础

6.1.1 指针的基本概念

例：

```
#include <iostream>
using namespace std;
int main()
{
    int a = 3;
    cout << "C++方式输出：" << endl;
    cout << "a address = " << &a << endl;
    cout << "C方式输出：" << endl;
    printf("a address = %p\n", &a);
    return 0;
}
```



C语言方式：printf函数 %p格式控制符

6.1 指针基础

6.1.1 指针的基本概念

例：

```
#include <iostream>
using namespace std;
int main()
{
    char c = 'a';
    cout << "C++方式输出： " << endl;
    cout << "c address = " << &c << endl;
    cout << "C方式输出： " << endl;
    printf("c address = %p\n", &c);
    return 0;
}
```

6.1 指针基础

6.1.1 指针的基本概念

例：

```
#include <iostream>
using namespace std;
int main()
{
    char c = 'a';
    cout << "C++方式输出： " << endl;
    cout << "c address = " << &c << endl;
    cout << "C方式输出： " << endl;
    printf("c address = %p\n", &c);
    return 0;
}
```

问题：为何char型变量c用cout<<&c不能输出地址？等后续内容学完再回头看

6.1 指针基础

6.1.1 指针的基本概念

内存的访问方式

✓ 直接访问：通过变量地址访问

✓ 间接访问：通过存放变量地址的变量访问

6.1 指针基础

6.1.1 指针的基本概念

指针和指针变量

变量的地址称为指针，存放地址的变量称为指针变量

如果一个指针变量存放的是某个变量的地址，则称这个指针变量指向该变量

内存中存放的变量、字符串、数组和函数等都有地址，其地址都可以用指针变量来存放

6.1 指针基础
6.1.2 指针变量的定义与使用

指针变量的定义形式

数据类型 *指针变量名;

说明：

*不是指针变量名的一部分，表示定义的是指针变量，位置可以靠左、靠右或居中

数据类型是指针指向的对象的数据类型，称为基类型。

6.1 指针基础

6.1.2 指针变量的定义与使用

例：

```
int *ptr;              //定义指向int型数据的指针变量ptr
int *p1,*p2;           //定义指向int型数据的指针变量p1,p2
double *tax_ptr;       //定义指向double型数据的指针变量tax_ptr
char *str;             //定义指向char型数据的指针变量str
```

注意：

同时定义多个指针变量时，每个指针变量名前都要有*，如

int *p1,p2;是定义一个指向int的指针变量p1和一个int变量p2

6.1 指针基础

6.1.2 指针变量的定义与使用

指针变量的类型和占用空间大小

指针是一种复合类型，其类型是基类型加*号。如有int *p;，则p的类型为int *。说指针变量时应完整地说明其是指向某特定类型的指针变量，如p是指向int的指针变量。

6.1 指针基础

6.1.2 指针变量的定义与使用

指针变量的类型和占用空间大小

指针是一种复合类型，其类型是基类型加*号。如有int *p;，则p的类型为int *。说指针变量时应完整地说明其是指向某特定类型的指针变量，如p是指向int的指针变量。

指针变量本身也占用一定的内存空间，占用空间大小与基类型无关，与系统有关，可以用sizeof(指针变量类型)或sizeof(指针变量名)求得。

例：

```
#include <iostream>
using namespace std;
int main()
{
    cout << sizeof(char) << endl;
    cout << sizeof(char*) << endl;
    cout << sizeof(short) << endl;
    cout << sizeof(short*) << endl;
    cout << sizeof(int) << endl;
    cout << sizeof(int*) << endl;
    cout << sizeof(long) << endl;
    cout << sizeof(long*) << endl;
    cout << sizeof(float) << endl;
    cout << sizeof(float*) << endl;
    cout << sizeof(double) << endl;
    cout << sizeof(double*) << endl;
    return 0;
}
```

6.1 指针基础

6.1.2 指针变量的定义与使用

指针变量指向一个变量

例：

```
int a=5;
int *p=&a;  //声明指针变量同时赋初值
```

或者：

```
int a=5;
int *p;
p=&a;      //先声明再赋值，注意不要写成*p=&a;
```

6.1 指针基础
6.1.2 指针变量的定义与使用

利用指针访问对象

解引用运算符 *，又称间接值运算符，取地址处存储的值

如果指针指向了一个对象，可以用解引用运算符来访问对象

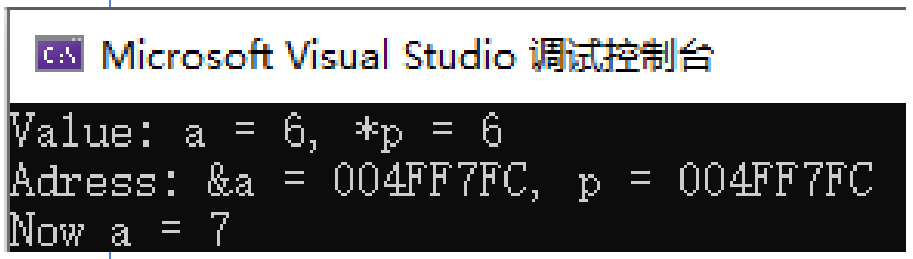
若p=&a,则*p即为a

6.1 指针基础

6.1.2 指针变量的定义与使用

例：

```
int main()
{
    int a = 6;
    int* p;
    p = &a;
    //两种表示值的方式
    cout << "Value: a = " << a;          //直接访问
    cout << ", *p = " << *p << endl;  //间接访问
    //两种表示地址的方式
    cout << "Adress: &a = " << &a;
    cout << ", p = " << p << endl;
    //使用指针改变值
    *p = *p + 1;
    cout << "Now a = "<< a;
    return 0;
}
```



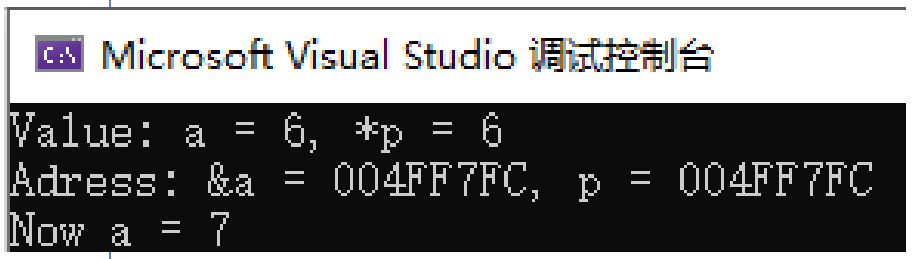
6.1 指针基础

6.1.2 指针变量的定义与使用

例：

```
int main()
{
    int a = 6;
    int* p;
    p = &a;
    //两种表示值的方式
    cout << "Value: a = " << a;          //直接访问
    cout << ", *p = " << *p << endl;  //间接访问
    //两种表示地址的方式
    cout << "Adress: &a = " << &a;
    cout << ", p = " << p << endl;
    //使用指针改变值
    *p = *p + 1;
    cout << "Now a = "<< a;
    return 0;
}
```

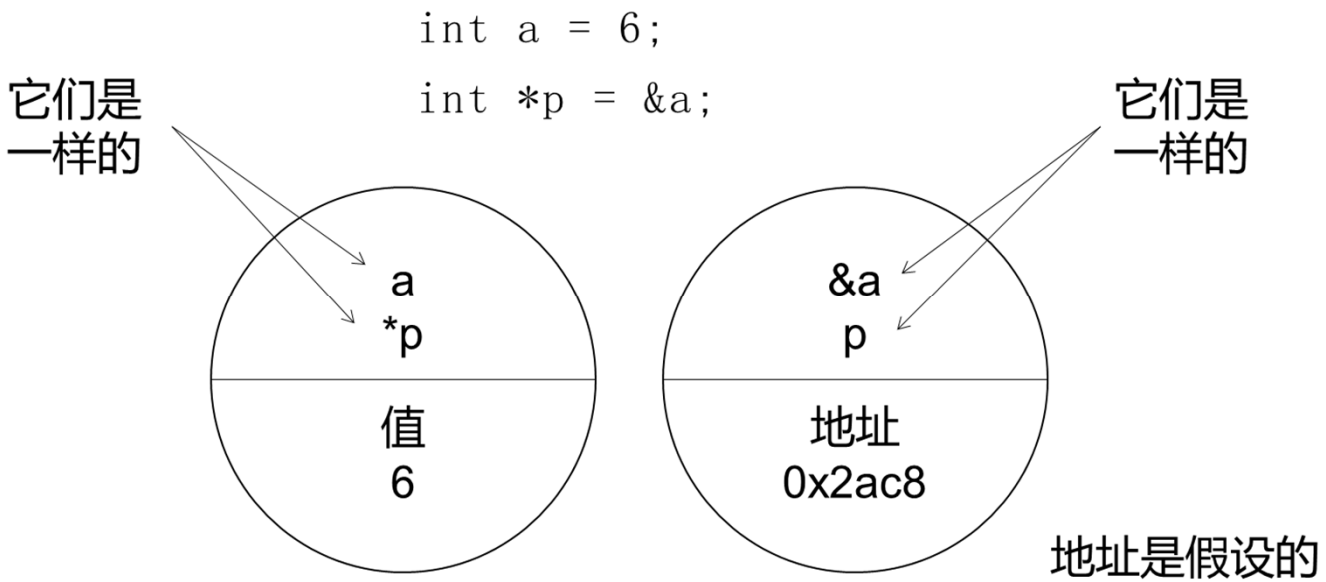
指针变量=地址
*指针变量=值



6.1 指针基础

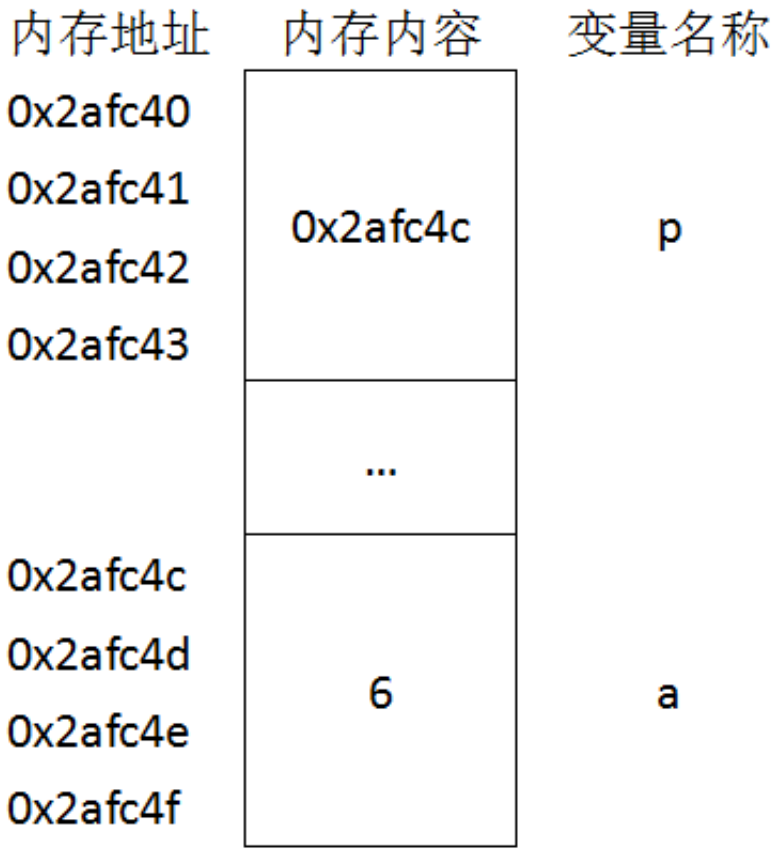
6.1.2 指针变量的定义与使用

若p指向a，则a ⇔ *p，&a ⇔ p



6.1 指针基础

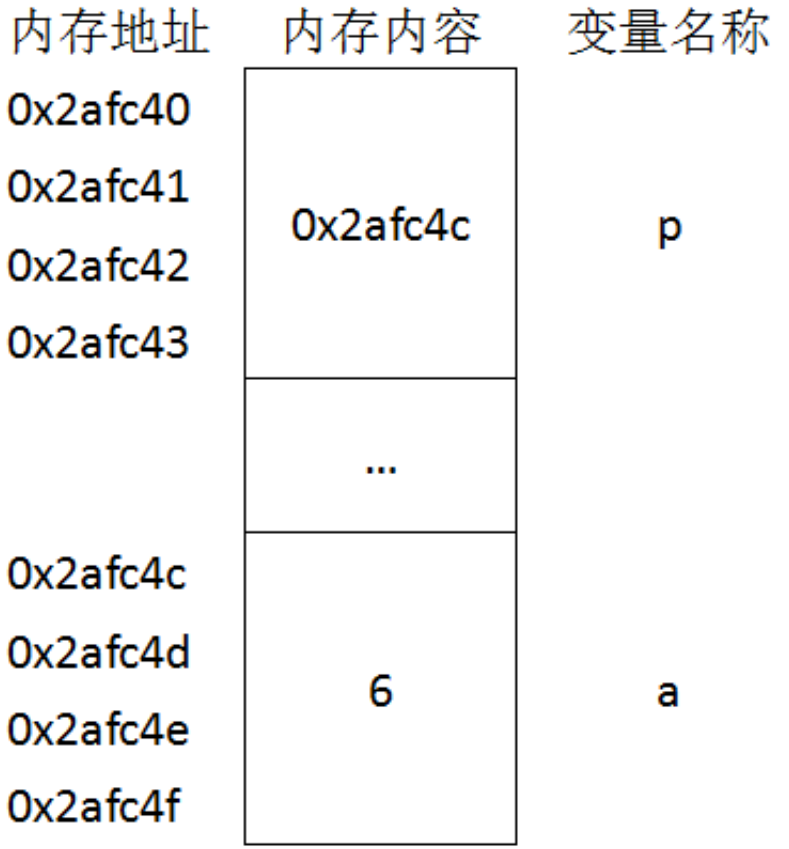
6.1.2 指针变量的定义与使用



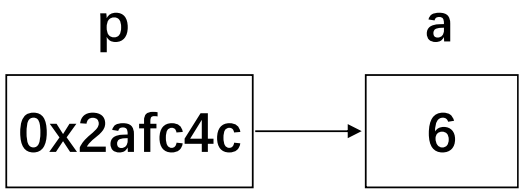

```
int a=6;
int *p=&a;
```

6.1 指针基础

6.1.2 指针变量的定义与使用

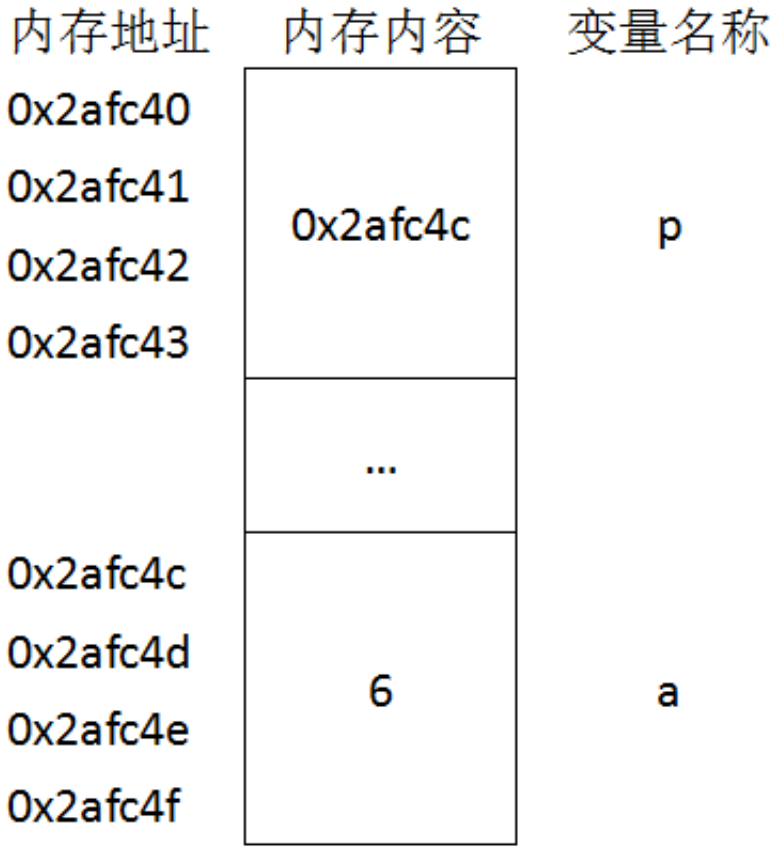



```
int a=6;
int *p=&a;
```

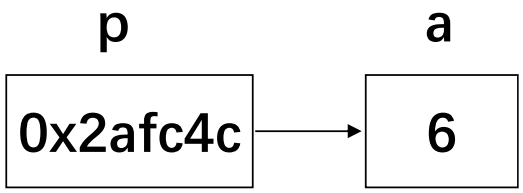


6.1 指针基础

6.1.2 指针变量的定义与使用




```
int a=6;
int *p=&a;
```



p的类型：int *

p所指向的变量类型：int

p的值：a的地址

p指向的变量(即*p)的值：a的值6

6.1 指针基础
6.1.2 指针变量的定义与使用

&和* 优先级、结合性

都是优先级第三组、结合性自右向左

若有int *p=&a;则有如下等价关系成立：

&*p ⇔ &(*p) ⇔ &(a) ⇔ &a ⇔ p

*&a ⇔ *(&a) ⇔ *(p) ⇔ *p ⇔ a

6.1 指针基础

6.1.2 指针变量的定义与使用

指针变量赋值

使用未初始化的指针变量的危险

未初始化的指针变量，很多书上称为野指针

例：

```
int *p;          //定义指针变量p
...
*p=6;            //将6存储在p所指向的内存位置
...
```

6.1 指针基础
6.1.2 指针变量的定义与使用

VS2019编译报错

```
error C4700: 使用了未初始化的局部变量“p”
```

Dev C++编译不报错，但可能会运行出错

```
Process exited after 5.896 seconds with return value 3221225477
```

6.1 指针基础
6.1.2 指针变量的定义与使用

创建指针变量时只分配用来存储指针变量的内存，并不会分配用来存储指针所指向的数据的内存

未初始化的指针变量的值为随机值，使用未初始化的指针可能会引发程序异常终止或者引发其它隐匿错误

建议所有指针变量被创建之后要及时初始化，使其有确切的指向或初始化为空指针

6.1 指针基础

6.1.2 指针变量的定义与使用

赋值方法

✓ 将指向的数据对象地址赋给指针变量，如：

```
p=&a;          //将变量a的地址赋给p
p=array;       //将数组array首元素地址赋给p
p=max;         //将函数max的入口地址赋给p
```

数组名、函数名都代表首地址

6.1 指针基础

6.1.2 指针变量的定义与使用

赋值方法

✓ 将指向的数据对象地址赋给指针变量，如：

```
p=&a;          //将变量a的地址赋给p
p=array;       //将数组array首元素地址赋给p
p=max;         //将函数max的入口地址赋给p
```

数组名、函数名都代表首地址

✓ 同类型指针变量之间赋值，如：

```
p1=p2;         //p1和p2是同类型的指针变量，将p2的值赋给p1，
                 即p1和p2指向同一个对象
```

6.1 指针基础

6.1.2 指针变量的定义与使用

✓ 将指针变量赋值为空指针，表示不指向任何变量，空指针的表示方法：

```
int *p=nullptr;    //C++11 标准支持
int *p=NULL;       //系统宏定义NULL值为0
int *p=0;          //直接直接用字面值0赋值
```

注意：不是指针变量指向地址为0的内存块，而是表示不指向任何变量

6.1 指针基础

6.1.2 指针变量的定义与使用

注意以下错误的赋值方法

✓ 不要将一个整型量赋给一个指针变量，如：

int *p=0xB8000000; (×)

✓ 指针变量基类型和其指向对象的类型不一致，如：

假设有int a; 则float *p=&a; (×)

✓ 不同类型的指针变量间不可以相互赋值，如：

int *p1;float *p2; 则：p1=p2; (×)

6.1 指针基础

6.1.2 指针变量的定义与使用

注意以下错误的赋值方法

✓ 不要将一个整型量赋给一个指针变量，如：

int *p=0xB8000000; (×)

✓ 指针变量基类型和其指向对象的类型不一致，如：

假设有int a; 则float *p=&a; (×)

✓ 不同类型的指针变量间不可以相互赋值，如：

int *p1;float *p2; 则：p1=p2; (×)

如果要将数字值作为地址来使用，或者将一种类型指针转换成另一种类型可以通过强制类型转换来实现，此处不详述。

6.1 指针基础

6.1.2 指针变量的定义与使用

指向指针的指针

由于指针本身也是一种类型，因此指针变量也可以有对应的指针类型，如：

int **p;

p就是一个指向指针数据的指针变量，称为二级指针变量

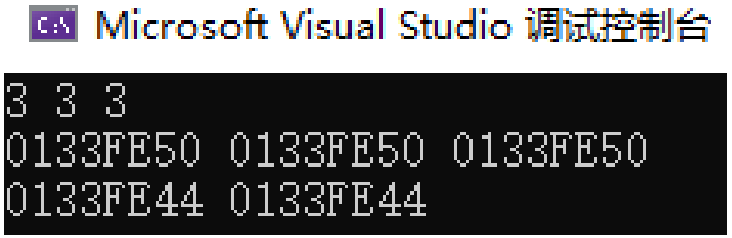
以此类推，就是多级指针

6.1 指针基础

6.1.2 指针变量的定义与使用

例：

```
int main()
{
    int a = 3;
    int* p1 = &a;
    int** p2 = &p1;
    cout << a << " " << *p1 << " " << **p2 << endl;  //都表示变量a
    cout << &a << " " << p1 << " " << *p2 << endl;  //都表示变量a的地址
    cout << &p1 << " " << p2 << endl;                //都表示变量p1的地址
    return 0;
}
```



6.1 指针基础

6.1.3 指针变量做函数参数

按值传递（模块4.1），例：编写函数实现两整数交换

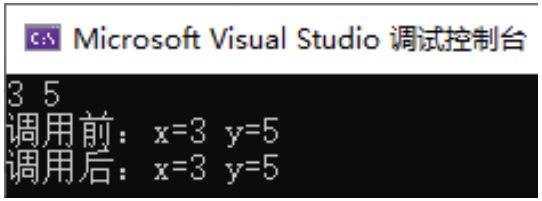
6.1 指针基础

6.1.3 指针变量做函数参数

按值传递（模块4.1），例：编写函数实现两整数交换

```
int main()
{
    int x, y;
    cin >> x >> y;
    cout << "调用前： ";
    cout << "x=" << x << " y=" << y << endl;
    interchange(x, y);
    cout << "调用后： ";
    cout << "x=" << x << " y=" << y << endl;
    return 0;
}
```

```
void interchange(int p1, int p2)
{
    int temp;
    temp = p1;
    p1 = p2;
    p2 = temp;
}
```



不能实现交换！

单向传值，形参的改变不能影响实参！

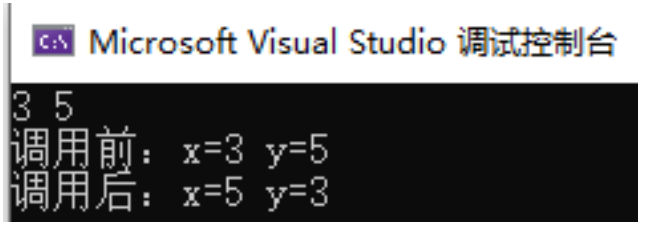
6.1 指针基础

6.1.3 指针变量做函数参数

按地址传递，例：

```
int main()
{
    int x, y;
    cin >> x >> y;
    cout << "调用前： ";
    cout << "x=" << x << " y=" << y << endl;
    interchange(&x, &y);
    cout << "调用后： ";
    cout << "x=" << x << " y=" << y << endl;
    return 0;
}
```

```
void interchange(int* p1, int* p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```



实现交换！

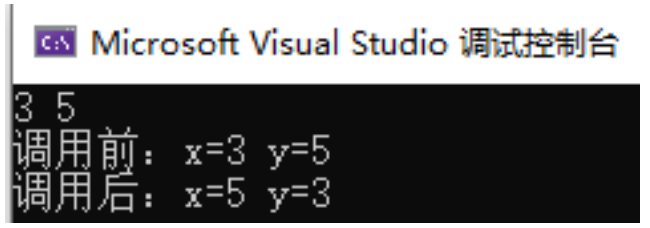
6.1 指针基础

6.1.3 指针变量做函数参数

按地址传递，例：

```
int main()
{
    int x, y;
    cin >> x >> y;
    cout << "调用前： ";
    cout << "x=" << x << " y=" << y << endl;
    interchange(&x, &y);
    cout << "调用后： ";
    cout << "x=" << x << " y=" << y << endl;
    return 0;
}
```

```
void interchange(int* p1, int* p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```



实现交换！

形参是指针变量，实参是变量地址

6.1 指针基础

6.1.3 指针变量做函数参数

为什么能实现交换?

分析程序执行过程:

6.1 指针基础

6.1.3 指针变量做函数参数

为什么能实现交换?

分析程序执行过程：

	x	y
主函数	3	5

6.1 指针基础

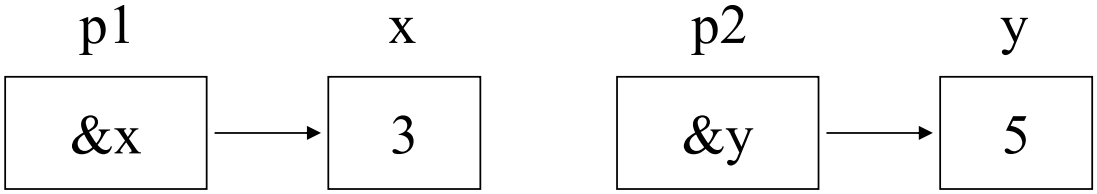
6.1.3 指针变量做函数参数

为什么能实现交换?

分析程序执行过程:

主函数

参数传递　　&x->p1,p1指向x,&y->p2,p2指向y

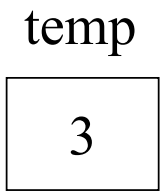


6.1 指针基础

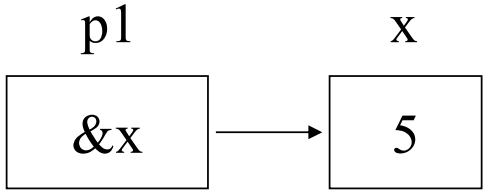
6.1.3 指针变量做函数参数

执行interchange函数内语句

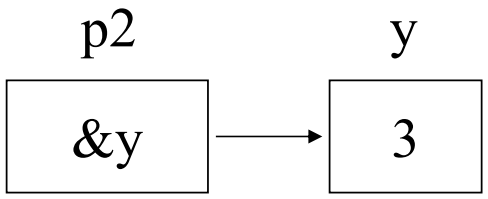
temp = *p1;



*p1 = *p2;



*p2 = temp;



6.1 指针基础
6.1.3 指针变量做函数参数

返回主函数

形参p1、p2被释放

但x、y值已改变

注意：x、y的地址值并未改变

6.1 指针基础

6.1.3 指针变量做函数参数

按地址传递，例：

```
int main()
{
    float x, y;
    cin >> x >> y;
    cout << "调用前：";
    cout << "x=" << x << " y=" << y << endl;
    interchange(&x, &y);
    cout << "调用后：";
    cout << "x=" << x << " y=" << y << endl;
    return 0;
}
```

```
void interchange(int* p1, int* p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```

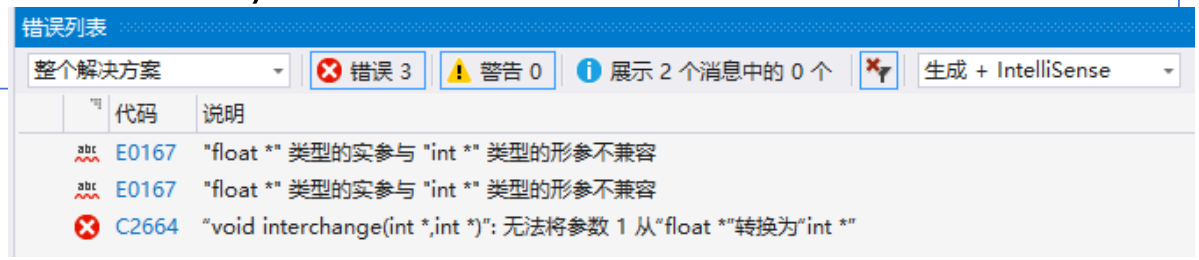
6.1 指针基础

6.1.3 指针变量做函数参数

按地址传递，例：

```
int main()
{
    float x, y;
    cin >> x >> y;
    cout << "调用前：";
    cout << "x=" << x << " y=" << y << endl;
    interchange(&x, &y);
    cout << "调用后：";
    cout << "x=" << x << " y=" << y << endl;
    return 0;
}
```

```
void interchange(int* p1, int* p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```



注意：

形参指针类型要和实参类型匹配

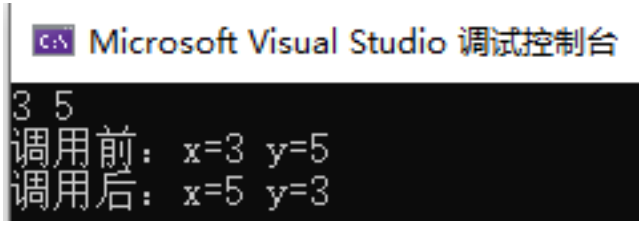
6.1 指针基础

6.1.3 指针变量做函数参数

按地址传递，例：

```
int main()
{
    int x, y, * px = &x, * py = &y;
    cin >> x >> y;
    cout << "调用前： ";
    cout << "x=" << x << " y=" << y << endl;
    interchange(px, py);
    cout << "调用后： ";
    cout << "x=" << x << " y=" << y << endl;
    return 0;
}
```

```
void interchange(int* p1, int* p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```



形参是指针变量，实参是指针变量

6.1 指针基础

6.1.3 指针变量做函数参数

例：分析以下函数能否实现x、y交换？

```
void interchange(int* p1, int* p2)
{
    int* temp;
    temp = p1;
    p1 = p2;
    p2 = temp;
}
```

主函数中有调用语句：

```
interchange(&x,&y);
```

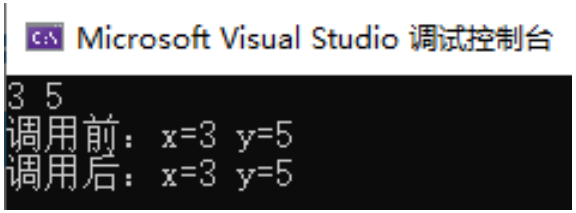
6.1 指针基础

6.1.3 指针变量做函数参数

例：分析以下函数能否实现x、y交换？

```
void interchange(int* p1, int* p2)
{
    int* temp;
    temp = p1;
    p1 = p2;
    p2 = temp;
}
```

主函数中有调用语句：
interchange(&x,&y);



不能实现交换！

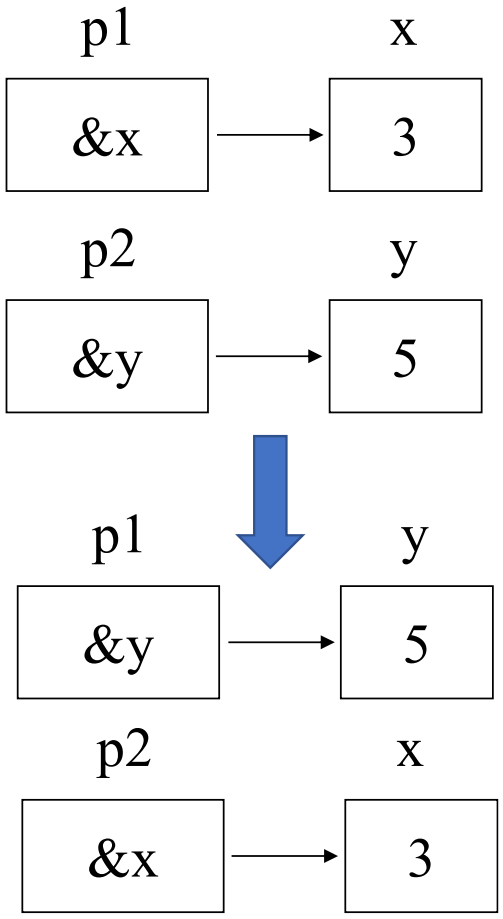
6.1 指针基础

6.1.3 指针变量做函数参数

例：分析以下函数能否实现x、y交换？

```
void interchange(int* p1, int* p2)
{
    int* temp;
    temp = p1;
    p1 = p2;
    p2 = temp;
}
```

主函数中有调用语句：
interchange(&x,&y);



通过改变形参指针变量值无法影响实参指针指向的变量！

6.1 指针基础

6.1.3 指针变量做函数参数

例：分析以下函数能否实现x、y交换？

```
void interchange(int* p1, int* p2)
{
    int* temp;
    *temp = *p1;
    *p1 = *p2;
    *p2 = *temp;
}
```

主函数中有调用语句：

```
interchange(&x,&y);
```

6.1 指针基础

6.1.3 指针变量做函数参数

例：分析以下函数能否实现x、y交换？

```
void interchange(int* p1, int* p2)
{
    int* temp;
    *temp = *p1;
    *p1 = *p2;
    *p2 = *temp;
}
```

指针变量未初始化！

主函数中有调用语句：
interchange(&x,&y);

6.1 指针基础

6.1.3 指针变量做函数参数

例：分析以下函数能否实现x、y交换？

```
void interchange(int* p1, int* p2)
{
    int* temp;
    *temp = *p1;
    *p1 = *p2;
    *p2 = *temp;
}
```

主函数中有调用语句：
interchange(&x,&y);

指针变量未初始化！

VS2019编译报错

Dev C++编译不报错，有输出，但本质上是有问题的！

6.1 指针基础

6.1.3 指针变量做函数参数

总结：

指针变量做函数参数，形参是指针变量，实参是变量地址或指针变量（还可以是数组名，见6.1.4）。

指针变量做函数参数，仍遵循单向值传递方式，调用函数时不会改变实参指针的值，但可以改变实参指针所指向变量的值。

指针变量做函数参数，欲改变主调函数中变量的值，需要操作形参所指向的值的方式进行，仅改变指针值无法完成。

指针变量做函数参数，可以通过指针变量改变主调函数中多个变量的值，相当于通过函数调用从被调函数得到多个值。