4 更复杂的数据结构

郑海永
zhenghaiyong@ouc.edu.cn

中国海洋大学 电子工程学院

目录 I

内容提要 I

内容提要 I

什么是“指针”？

互联网上的资源——地址

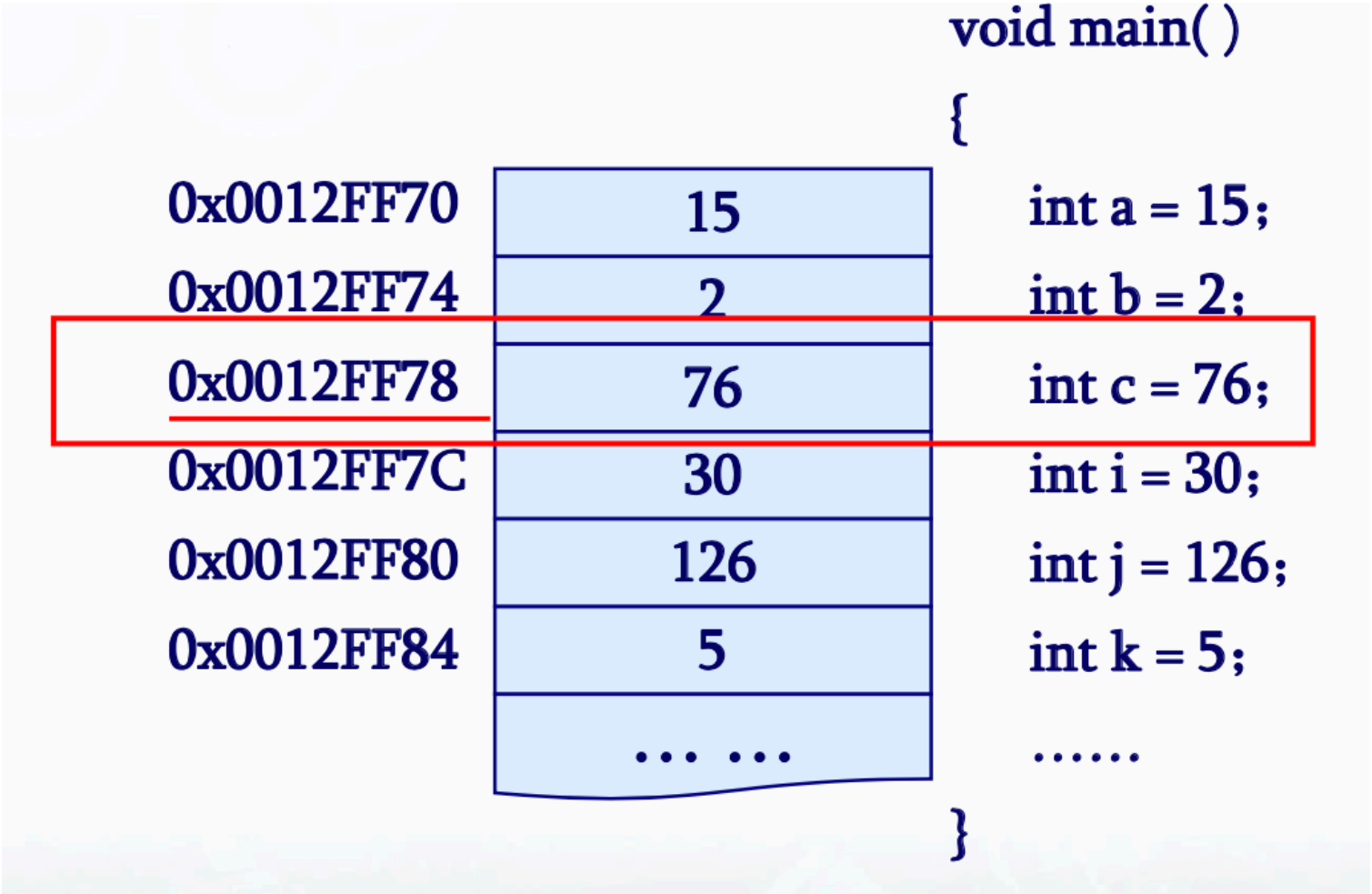
http://www.nasa.gov/images/content/166502.jpg

N49 Nebula

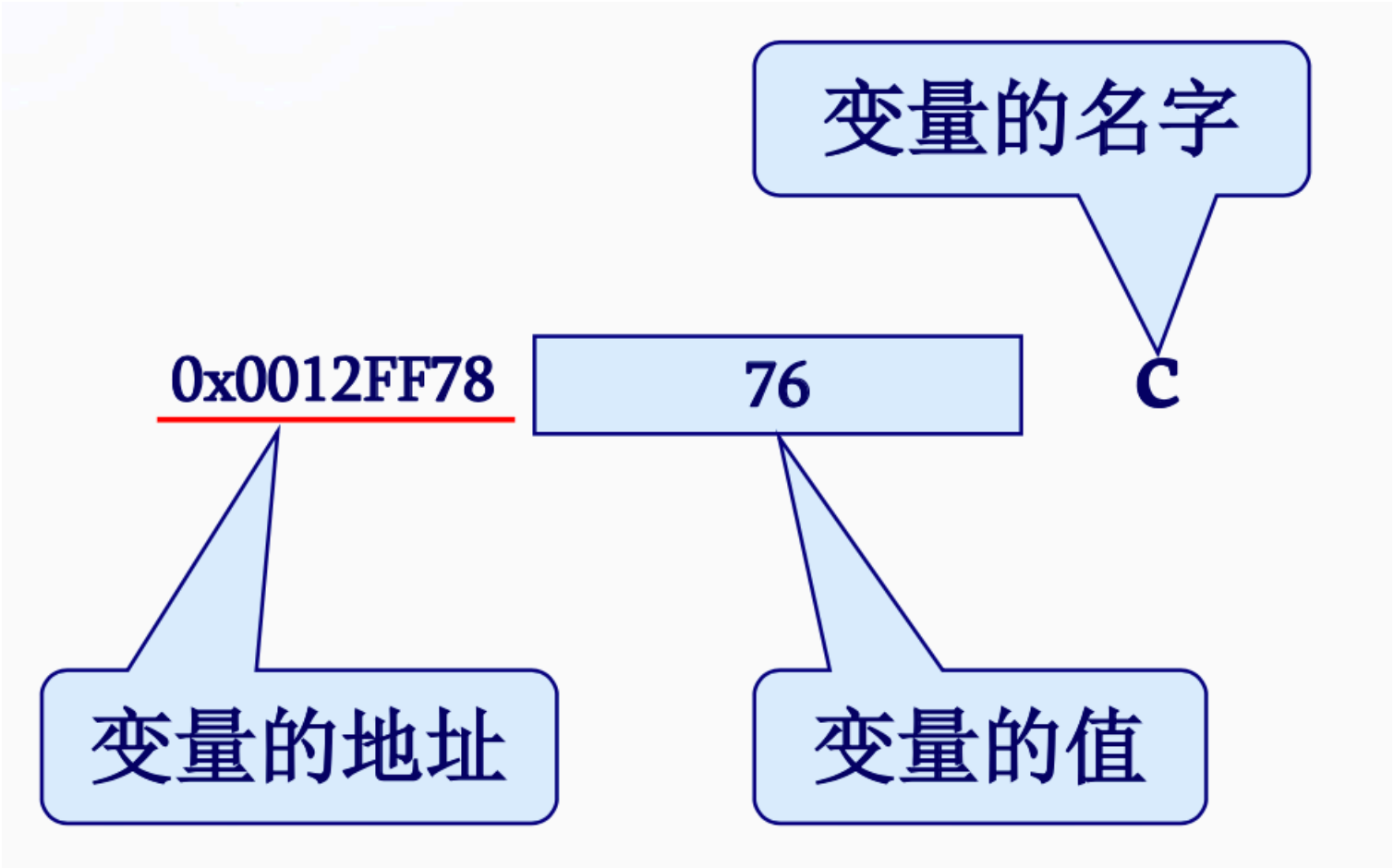
可以把“网址”称为 指向资源的“指针”

什么是“指针”？

内存中的位置——地址



变量的三要素



内存中的资源——地址

把某个变量的地址称为“指向该变量的指针”

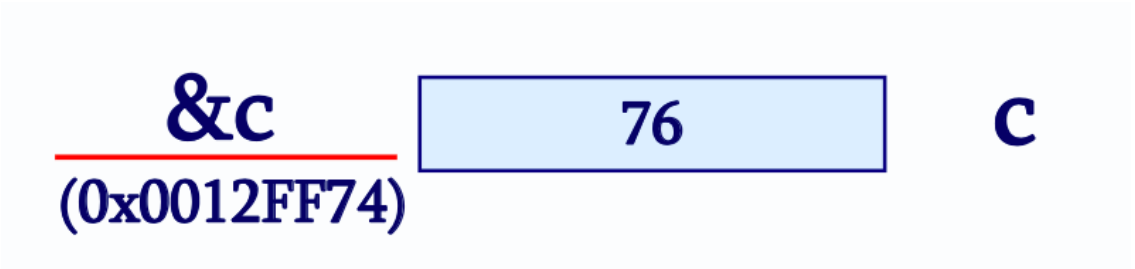
0x0012FF74 | 76 | C

http://www.nasa.gov/images/content/166502.jpg

N49 Nebula

如何知道一个变量的地址？

取地址运算符&



- `cout<<&c<<endl;`
- `cout<<sizeof(&c)<<endl;`

变量地址（指针）的作用

- 我们通过资源地址（指针）访问网络资源

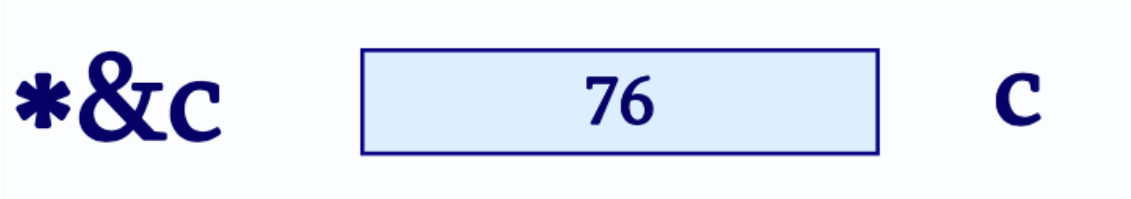
http://www.nasa.gov/images/content/166502.jpg

N49 Nebula

- 计算机通过**变量地址（指针）**操作变量

通过变量地址（指针）操作变量

指针运算符*



- `cout<<*&c<<endl;`
- `cout<<c<<endl;`

通过变量地址（指针）操作变量

指针运算符*

***&c 等价于 c**

编译时，编译器建立 变量名 到 地址 的映射

`cout<<a<<endl;` ⇔ `cout<<*&a<<endl;`

- 找到变量a的地址；
- 从地址 0x0012FF74 开始的四个字节中取出数据；
- 将取出的数据送到显示器。

什么叫“指针变量”？

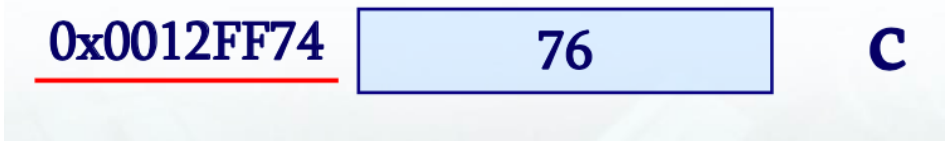
存放地址（指针）的变量

- 我们可以设置一个变量，来存放网络资源的地址

http://www.nasa.gov/images/content/166502.jpg

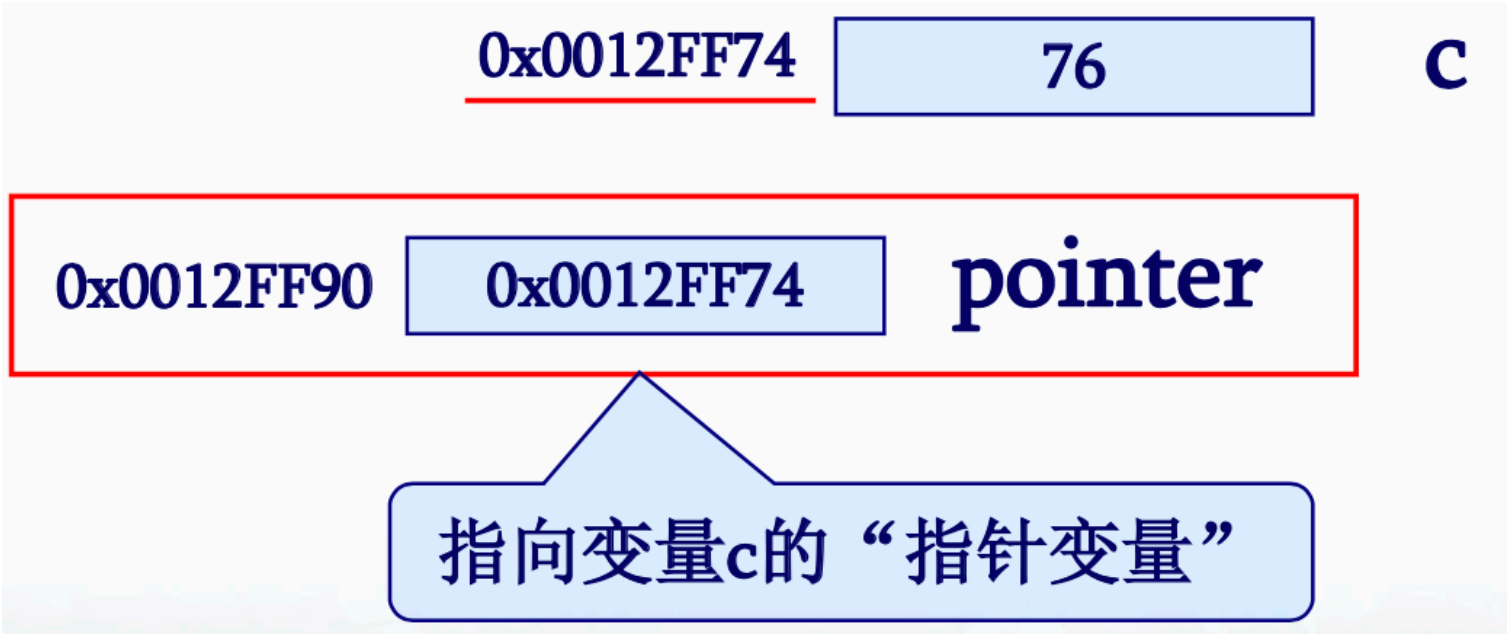
N49 Nebula

- 我们也可以设置一个变量，来存放**变量的地址（指针变量）**

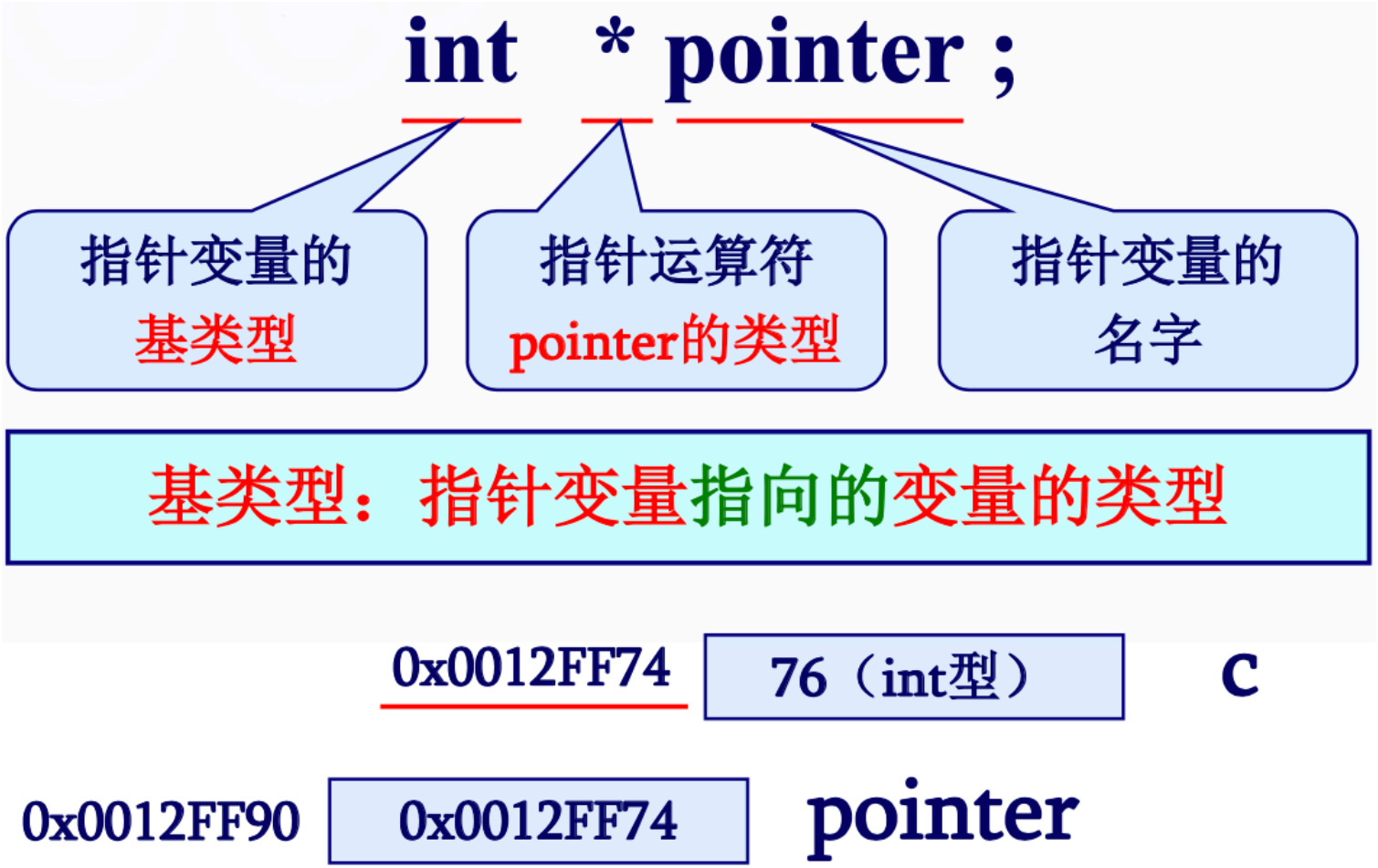


指针变量

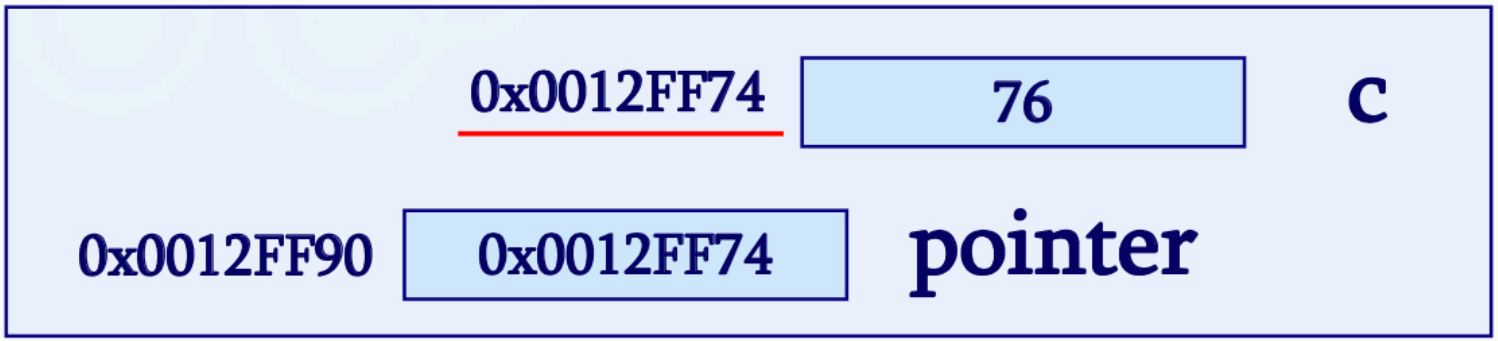
专门用于存放指针（某个变量的地址）的变量



定义一个指针变量

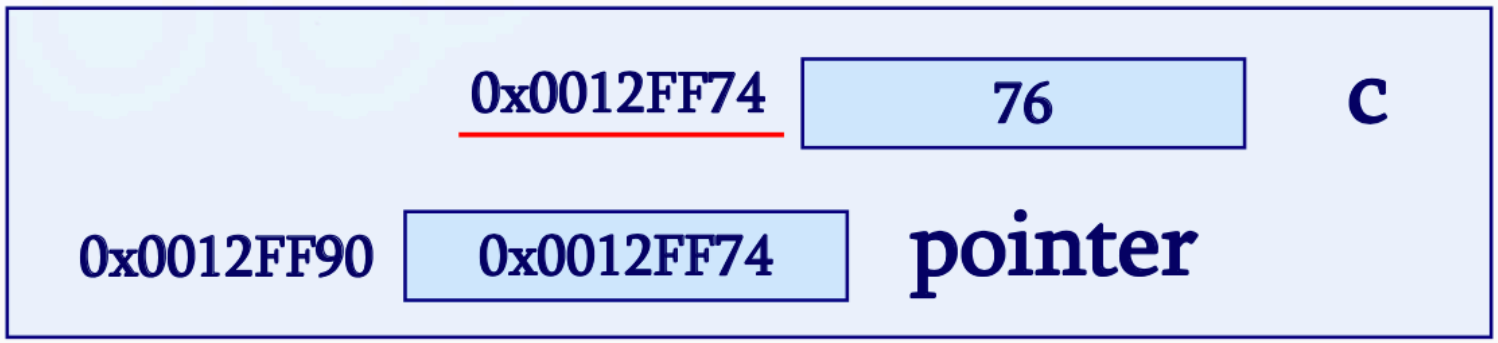

int * pointer ;
指针变量的
基类型
指针运算符
pointer的类型
指针变量的
名字
基类型：指针变量指向的变量的类型
0x0012FF74
76（int型）
c
0x0012FF90
0x0012FF74
pointer

指针变量的定义




```
int c = 76;
int *pointer;    //定义名字为pointer的指针变量;
pointer = &c;
        //将变量c的地址赋值给指针变量pointer ;
        //赋值后，称指针变量pointer指向了变量c
```

指针变量的定义




```
int c = 76;
int *pointer;    //定义名字为pointer的指针变量;
pointer = &c;
```

能不能写成

```
pointer = c ;
```

绝对不行！

- 因为pointer是存放地址的变量，所以只能存放地址！

指针变量的使用

- **问题：**
  - **既然指针变量中存放的是“某个变量的地址”；**
  - **可否通过“指针变量”访问“它所指向的变量”呢？**
- **例如：**

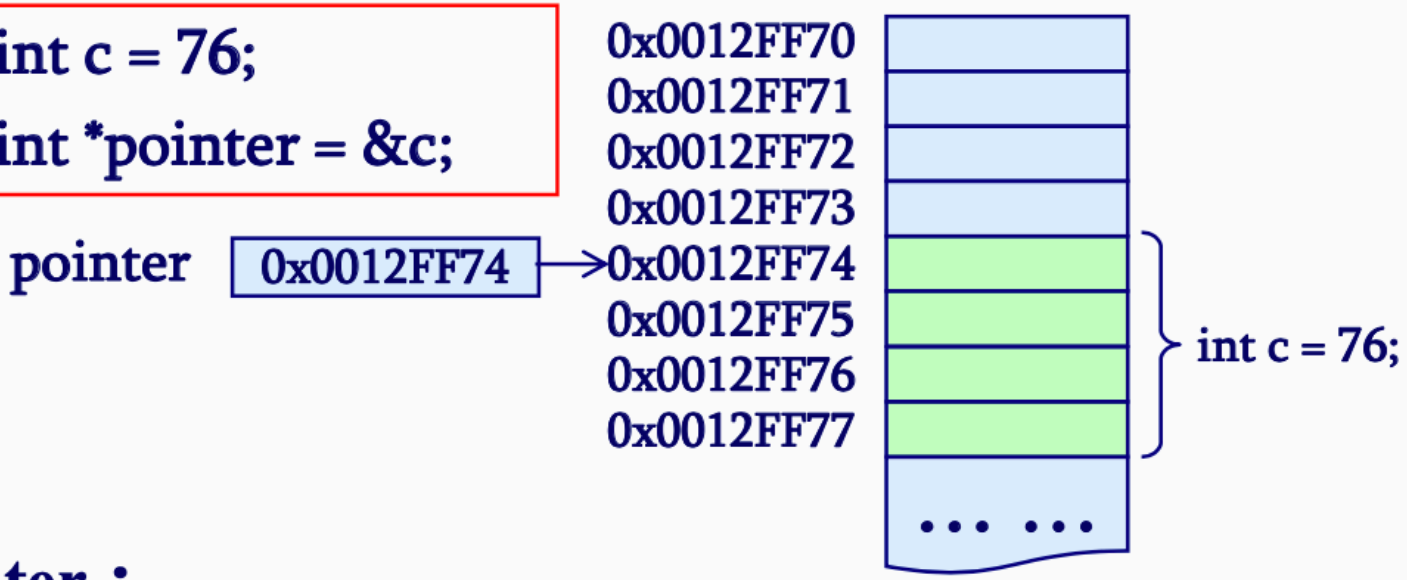
  - **可否利用pointer访问到变量c的值“76”呢？**

指针变量的使用

■ 也利用 指针运算符* 实现

若有：

```
int c = 76;
int *pointer = &c;
```



则*pointer：

- 为“pointer所指向的存储单元的内容”；
- “pointer所指向的存储单元的内容”是变量c

指针变量的地址

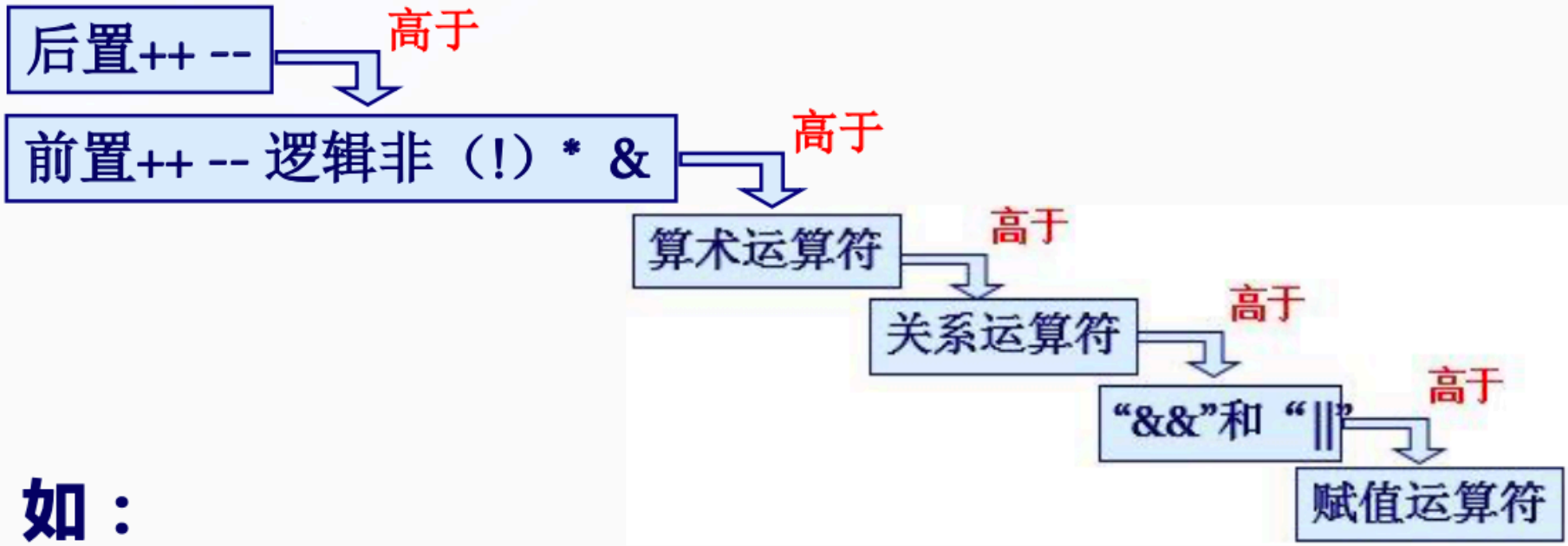
指针变量也是变量，是变量就有地址

```
#include<iostream>
using namespace std;
int main()
{
    int iCount=18;
    int *iPtr=&iCount;
    *iPtr=58;
    cout<<iCount<<endl;
    cout<<iPtr<<endl;
    cout<<&iCount<<endl;
    cout<<*iPtr<<endl;
    cout<<&iPtr<<endl;
    return 0;
}
```

指针变量示例

```
#include<iostream>
using namespace std;
int main()
{
    int a=0,b=0,temp;
    int *p1=NULL,*p2=NULL;
    cin>>a>>b;
    p1=&a;
    p2=&b;
    if(*p1<*p2) {
        temp=*p1;*p1=*p2;*p2=temp;
    }
    cout<<"max="<<*p1<<",min="<<*p2<<endl;
    return 0;
}
```

& 与 * 的运算优先级



- 如：
  - &*pointer = &(*pointer)
  - *&a = * (&a)
  - (*pointer) ++ 不等于 *pointer++

pointer++ 的含义


0x0012FF70
0x0012FF71
0x0012FF72
0x0012FF73
0x0012FF74
0x0012FF75
p2
0x0012FF76
0x0012FF76
0x0012FF77
0x0012FF78
0x0012FF79
int;
p2++
0x0012FF7A
0x0012FF7A
0x0012FF7B
体现了基类型的作用！

pointer++ 示例

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

讨论：iPtr++ 的含义

假设：iPtr当前所存的地址是 0x00000100

- 若iPtr指向一个**整型**元素（占 4 个字节）
  则iPtr++等于iPtr+1*4=0x00000104
- 若iPtr指向一个**实型**元素（占 4 个字节）
  则iPtr++等于iPtr+1*4=0x00000104
- 若iPtr指向一个**字符**元素（占 1 个字节）
  则iPtr++等于iPtr+1*1=0x00000101

小结

- 什么是“指针”
- 什么是“指针变量”
  - *pointer的含义
  - pointer++的含义

内容提要 I

指向数组元素的指针

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a[5]={1,2,3,4,5};
6      int *p=&a[3];
7      cout<<*p<<endl;
8      *p=100;
9      cout<<a[3]<<endl;
10     return 0;
11 }
```

指针与数组

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a[5]={10,11,12,13,14};
6      int *p=NULL;
7      cout<<a<<endl;
8      p=a;
9      cout<<p<<endl;
10     return 0;
11 }
```

指针与数组

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a[5]={10,11,12,13,14}
6      cout<<a<<endl;
7      cout<<*a<<endl;
8      cout<<&a[0]<<endl;
9      cout<<a[0]<<endl;
10     return 0;
11 }
```

数组的地址（数组的指针）

- 数组名代表数组首元素的地址

  数组名是指向数组第一个元素的指针

- 对于数组a[10]，数组名a代表数组a[10]中第一个元素a[0]的地址，即a与&a[0]等价。
- 注意：a是地址常量，不是变量，不能给其赋值。

指针与数组

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a[5]={10,11,12,13,14};
6      int *p=NULL;
7      cout<<a<<endl;
8      p=a;
9      cout<<p<<endl;
10     cout<<*p<<endl;
11     cout<<*p++<<endl;
12     cout<<*p++<<endl;
13     return 0;
14 }
```

利用指针变量引用数组元素

- 定义：
  - 数组`int a[10];`
  - 指针`int *pointer;`
- 则：
  - `pointer=a;` ⇔ `pointer=&a[0];`
- 数组访问：
  - `pointer+i` ⇔ `a+i` ⇔ `&a[i]`
  - `*(pointer+i)` ⇔ `*(a+i)` ⇔ `a[i]`
- 表示形式：
  - `pointer[i]` ⇔ `*(pointer+i)`

注意事项

- `int *p=&a[0];`
  - a++是没有意义的，但是p++会引起p变化。
  - p可以指向数组最后一个元素以后的元素。
- **指针做加减运算时一定注意有效的范围**

```
1 int a[5];
2 int *iPtr=&a[1];
3 iPtr--;  //now ok
4 *iPtr=3; //ok
5 iPtr--;  //dangerous
6 *iPtr=6; //damage
```

注意事项

- 定义：`int a[5]={1,2,3,4,5}; int *p;`
  - 设当前：`i=3; a[i]=4;`
  - 则：`t=*p--` ⇔ `t=a[i--]`
- 特别注意：
  - `*++p` ⇔ `a[++i]`，先将p自加，再做*运算。
  - `*--p` ⇔ `a[--i]`，先使p自减，再做*运算。
  - `*p++` ⇔ `a[i++]`，先做*运算，再将p自加。
  - `*p--` ⇔ `a[i--]`，先做*运算，再将p自加。

```cpp
#include<iostream>
using namespace std;
int main()
{
    int a[5]={1,2,3,4,5};
    int *p=&a[3];
    *p=100;
    cout<<*p++<<endl;
    cout<<*p--<<endl;
    cout<<*--p<<endl;
    return 0;
}
```

利用指针代替数组下标

```
1 int main()
2 {
3     int a[10];
4     for(i=0;i<10;i++)
5         cin>>a[i];
6     for(i=9;i>=0;i--)
7     cout<<setw(3)<<a[i];
8     return 0;
9 }
```

```
1 int main()
2 {
3     int a[10],i,*p=a;
4     for(i=0;i<10;i++)
5         cin>>*p++;
6     for(p--;p>=a;)
7     cout<<setw(3)<<*p--;
8     return 0;
9 }
```

倒置数组元素

倒置数组元素

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    int a[10],*p=NULL,*q=NULL,temp;
    for(p=a;p<a+10;p++)
        cin>>*p;
    for(p=a,q=a+9;p<q;p++,q--)
    {
        temp=*p;*p=*q;*q=temp;
    }
    for(p=a;p<a+10;p++)
        cout<<setw(3)<<*p;
    return 0;
}
```

内容提要 I

字符串与指针

- 指向数组的指针

```
1 int a[10];
2 int *p;
3 p=a;
```

- 指向字符串的指针

```
1 char a[10];
2 char *p;
3 p=a;
```

字符串指针示例

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    char a[]="How are you?",b[20];
    char *p1,*p2;
    for(p1=a,p2=b;*p1!='\0';p1++,p2++)
        *p2=*p1;
    *p2='\0';
    cout<<"string a is: "<<a<<endl;
    cout<<"string b is: "<<b<<endl;
    return 0;
}
```

```
1  int main()
2  {
3      int a=5;
4      int *pa=&a;
5      int b[6]={1,2,3,4,5,6};
6      int *pb=b;
7      char c[6]={'h','e','l','l','o','\0'};
8      char *pc=c;
9      cout<<a<<endl;
10     cout<<pa<<endl<<endl;
11     cout<<b<<endl;
12     cout<<pb<<endl<<endl;
13     cout<<c<<endl;
14     cout<<pc<<endl;
15     return 0;
16 }
```

```cpp
int main()
{
    int a=5;
    int *pa=&a;
    int b[6]={1,2,3,4,5,6};
    int *pb=b;
    char c[6]={'h','e','l','l','o','\0'};
    char *pc=c;
    cout<<a<<endl;
    cout<<pa<<endl<<endl;
    cout<<b<<endl;
    cout<<pb<<endl<<endl;
    cout<<static_cast<void*>(c)<<endl;
    cout<<static_cast<void*>(pc)<<endl;
    return 0;
}
```

字符串指针示例

```
#include<iostream>
using namespace std;
int main()
{
    char buffer[10]="ABC";
    char *pc;
    pc="hello";
    cout<<pc<<endl;
    pc++;
    cout<<pc<<endl;
    cout<<*pc<<endl;
    pc=buffer;
    cout<<pc<<endl;
    return 0;
}
```

内容提要 I

一维数组

数组名相当于指向数组第一个元素的指针

若a是指向数组第一个元素的指针，即a相当于&a[0]

- &a是“指向数组”的指针，相当于管辖范围“上升”了一级；
- *a是数组第一个元素a[0]，相当于管辖范围“下降”了一级。

二维数组

- 二维数组a[3][4]包含三个元素：a[0],a[1],a[2]
- 每个元素都是一个“包含四个整型元素”的数组
- 二维数组的第一个元素是a[0]
- a[0]是一个“包含四个整型元素”的一维数组

二维数组

```
1 int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

- a ⇔ &a[0]
- a[0] ⇔ &a[0][0]
- a[0] ⇔ *a
- a[0][0] ⇔ **a

小结

- **数组名**相当于指向数组**第一个元素**的指针
- &E相当于把E的管辖范围**上升**了一个级别
- *E相当于把E的管辖范围**下降**了一个级别

二维数组的地址

```
1  int main()
2  {
3      int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
4      cout<<"a="<<a<<endl;
5      cout<<"&a[0]="<<&a[0]<<endl<<endl;
6      cout<<"a+1="<<a+1<<endl;
7      cout<<"&a[0]+1="<<&a[0]+1<<endl<<endl;
8      cout<<"*a="<<*a<<endl;
9      cout<<"a[0]="<<a[0]<<endl;
10     cout<<"&a[0][0]="<<&a[0][0]<<endl<<endl;
11     cout<<"*a+1="<<*a+1<<endl;
12     cout<<"a[0]+1="<<a[0]+1<<endl;
13     cout<<"&a[0][0]+1="<<&a[0][0]+1<<endl<<endl;
14     return 0;
15 }
```

二维数组的地址

```
1  int main()
2  {
3      int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
4      cout<<"a="<<a<<endl;
5      cout<<"&a[0]="<<&a[0]<<endl<<endl;
6      cout<<"a+1="<<a+1<<endl;
7      cout<<"&a[0]+1="<<&a[0]+1<<endl<<endl;
8      cout<<"a[1]="<<a+1<<endl;
9      cout<<"&a[1]="<<&a[1]<<endl;
10     cout<<"*(a+1)="<<*(a+1)<<endl<<endl;
11     cout<<"*a+1="<<*a+1<<endl<<endl;
12     cout<<"&a="<<&a<<endl;
13     cout<<"&a+1="<<&a+1<<endl;
14     return 0;
15 }
```

小结

数组名相当于指向数组第一个元素的指针

- *a等价于a[0]，相当于让a下降了一级；
- &a表示“指向二维数组”的指针，相当于上升了一级。

遍历数组元素

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
6      int *p;
7      for(p=____;p<&a[0][0]+12;p++)
8          cout<<p<<" "<<*p<<endl;
9      return 0;
10 }
```

遍历数组元素

```
#include<iostream>
using namespace std;
int main()
{
    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    int *p;
    for(p=&a[0][0];p<&a[0][0]+12;p++)
        cout<<p<<" "<<*p<<endl;
    return 0;
}
```

程序填空并判断正误

- 输入i,j，输出a[i][j]。

```
1  int main()
2  {
3      int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
4      int ____,i,j; //填空：完成变量 p 的定义
5      p=a;
6      cin>>i>>j;
7      cout<<setw(4)<<*(*(p+i)+j); //判断：访问 a[i][j]
8      return 0;
9  }
```

问题分析

从p=a开始

- a相当于指向a[3][4]“第一个元素”的指针；
- 所谓“第一个元素”是指一个“包含4个int型元素的一维数组”；
- 所以a相当于一个“包含4个int型元素的一维数组”的地址；
- 因此p的基类型应该是：“包含4个int型元素的一维数组”。
- 如何定义一个指向“包含4个int型元素的一维数组”的指针变量？
- `int (*p)[4];`

程序填空并判断正误

- 输入i,j，输出a[i][j]。

```
1 int main()
2 {
3     int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
4     int (*p)[4],i,j; //填空：完成变量 p 的定义
5     p=a;
6     cin>>i>>j;
7     cout<<setw(4)<<*(*(p+i)+j); //判断：访问 a[i][j]
8     return 0;
9 }
```

利用指针变量引用多维数组中的数组

((p+i)+j)是什么？

- p指向一个“包含4个int型元素的一维数组”；
- p+i是第i+1个“包含4个int型元素的一维数组“的地址；
- p+i等价于&a[i]；
- *(p+i)等价于a[i]；
- *(p+i)+j等价于a[i]+j；
- a[i]+j等价于&a[i][j]；
- *(*(p+i)+j)等价于a[i][j]。

程序填空并判断正误

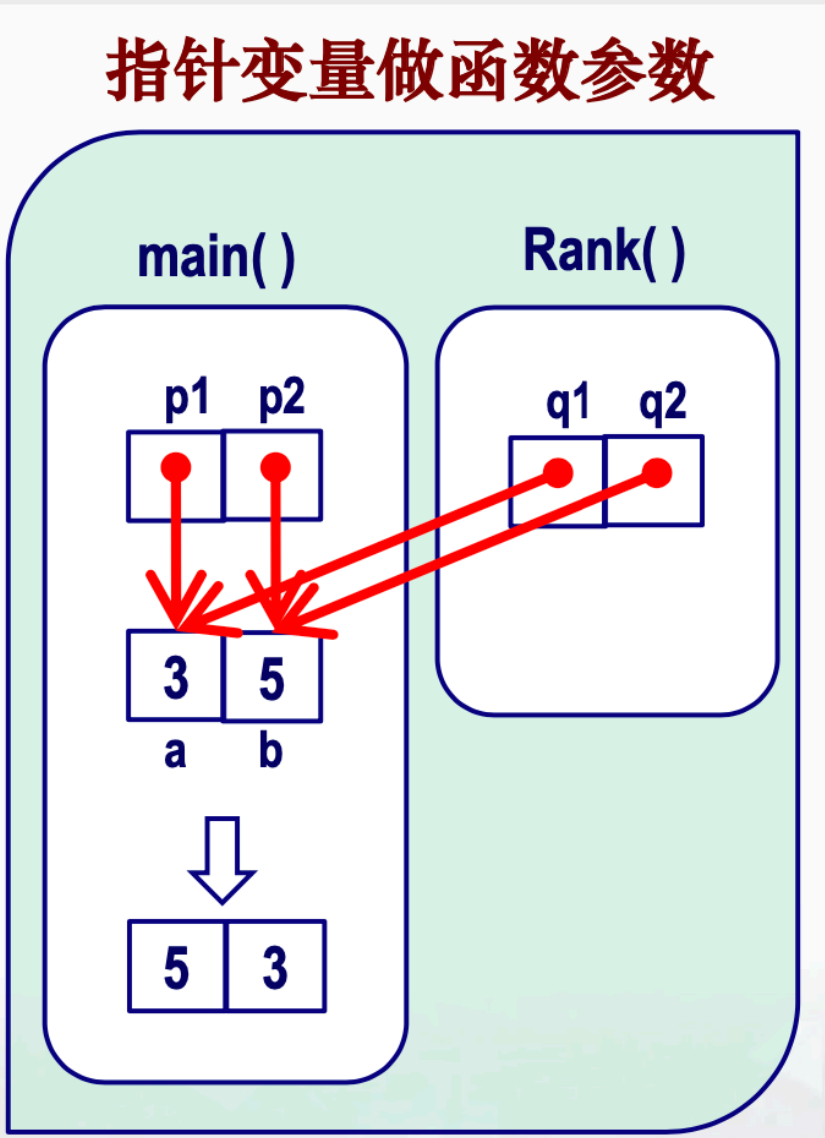
- 输入i,j，输出a[i][j]。

```
1  int main()
2  {
3      int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
4      int (*p)[4],i,j; //填空：完成变量 p 的定义
5      p=a;
6      cin>>i>>j;
7      cout<<setw(4)<<p[i][j]; //判断：访问 a[i][j]
8      return 0;
9  }
```

内容提要 I

指针变量做函数参数

```
#include<iostream>
using namespace std;
void Rank(int *q1, int *q2)
{
    int temp;
    if (*q1 < *q2)
    {
        temp = *q1;
        *q1 = *q2;
        *q2 = temp;
    }
}
int main()
{
    int a, b, *p1, *p2;
    cin >> a >> b;
    p1 = &a; p2 = &b;
    Rank(p1, p2);
    cout << a << " " << b << endl;
    return 0;
}
```



数组名做函数参数

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

多维数组名做函数参数

例：有一个3×4的矩阵，求所有元素中的最大值。

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

数组名做形参

C++ 编译器将形参数组名作为**指针变量**来处理

```
1  int sum(int array[],int n)
2  {
3      for(int i=0;i<10-1;i++)
4      {
5          *(array+1)=*array++(array+1);//Error
6          array++;
7      }
8      return *array;
9  }
10 int main()
11 {
12     int a[10]={1,2,3,4,5,6,7,8,9,10};
13     cout<<sum(a,10);
14     return 0;
15 }
```

符号常量

- const 数据类型常量名 = 常量值;
- 数据类型 const 常量名 = 常量值;

```
1 int main()
2 {
3     const float PI=3.14159f;//float const PI=3.14159f;
4     float r;
5     cout<<"r:";
6     cin>>r;
7     cout<<PI*r*r<<endl;
8     return 0;
9 }
```

指向符号常量的指针

```
const int *p;
```

```
1 int main()
2 {
3     int a=256;
4     int *p=&a;
5     *p=257;
6     cout<<*p<<endl;
7     return 0;
8 }
```

```
1 int main()
2 {
3     int a=256;
4     const int *p=&a;
5     *p=257;//Error
6     cout<<*p<<endl;
7     return 0;
8 }
```

指向符号常量的指针

```
1 void mystrcpy(char *dest,const char *src)
2 { ... }
```

保证字符串src不被修改！

```
1 int main()
2 {
3     char a[20]="How are you!";
4     char b[20];
5     mystrcpy(b,a);
6     cout<<b<<endl;
7     return 0;
8 }
```

指向符号常量的指针

```
1  int main()
2  {
3      const int a=78;
4      const int b=28;
5      int c=18;
6      const int *pi=&a;
7      *pi=58;//error
8      pi=&b;//ok
9      *pi=69;//error
10     pi=&c;//ok
11     *pi=88;//error
12     return 0;
13 }
```

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

```
1 int max(int a,int y);
```

- 返回指针类型数据的函数

```
1 int *function(int x,int y)
```

函数名字前面表示函数的类型“*”

返回指针值的函数

```
int 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;
    return 0;
}
```

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

返回指针值的函数

```
1  #include<iostream>
2  using namespace std;
3  int *getIntAd()
4  {
5      int value=20;
6      return &value;
7  }
8  int main()
9  {
10     int *p;
11     p=getIntAd();
12     cout<<*p<<endl;
13     return 0;
14 }
```

```cpp
#include<iostream>
using namespace std;
int main()
{
    int *p,*q;
    p=getIntAd1();
    q=getIntAd2();
    cout<<*p<<endl;
    return 0;
}
```

```cpp
int *getIntAd1()
{
    int value1=20;
    return &value1;
}
int *getIntAd2()
{
    int value2=30;
    return &value2;
}
```

```cpp
#include<iostream>
using namespace std;
int value1=20;
int value2=30;
int main()
{
    int *p,*q;
    p=getIntAd1();
    q=getIntAd2();
    cout<<*p<<endl;
    return 0;
}
```

```cpp
int *getIntAd1()
{
    return &value1;
}
int *getIntAd2()
{
    return &value2;
}
```

```cpp
#include<iostream>
using namespace std;
int main()
{
    int *p,*q;
    p=getIntAd1();
    q=getIntAd2();
    cout<<*p<<endl;
    return 0;
}
```

```cpp
int *getIntAd1()
{
    static int value1=20;
    return &value1;
}
int *getIntAd2()
{
    static int value2=30;
    return &value2;
}
```

静态局部变量

- 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。
- 用关键词**static**进行声明：

```
1 static int value1=20;
```

```
#include<iostream>
using namespace std;
void function()
{
    int a = 0;
    static int b = 0;
    a = a + 1;
    b = b + 1;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
}
int main()
{
    for (int i = 0; i < 5; i++)
    {
        function();
        cout << "Call Again!" << endl;
    }
    return 0;
}
```

静态 vs. 动态

- 动态变量a的有效范围
- 静态变量b的有效范围

小结

指针与函数

- 指针用做函数参数
  - 函数拿到地址可对其所指内容进行修改；
  - 可以使用`const`来“限制”指针的功能。
- 指针用做函数返回值
  - 必须确保函数返回的地址是有意义的；
  - 返回全局变量或静态局部变量。

内容提要 I

内容提要 I

经常遇到的情况

用一组变量来描述同一个“事物”

```
1 int id; //声明学号为 int 型
2 char name[20]; //声明姓名为字符数组
3 char sex; //声明性别为字符型
4 int age; //声明年龄为整型
5 float score; //声明成绩为实型
6 char addr[20]; //声明地址为字符数组
```

结构体

```
1 struct student
2 {
3     int id;
4     char name[20];
5     char sex;
6     int age;
7     float score;
8     char addr[30];
9 };
```

定义结构体类型的变量

- 直接用已声明的结构体类型定义变量名

```
student student1,student2;
```

- 对比：

```
int a;
float b;
```

定义结构体类型的变量

- 在声明类型的同时定义变量

```
1 struct student
2 {
3     int id;
4     char name[20];
5     char sex;
6     int age;
7     float score;
8     char addr[30];
9 } ZhangSan,LiSi;
```

定义结构体类型的变量

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student mike={123,{'m','i','k','e','\0'}};
    mike.id_num=20130000+mike.id_num;
    for(int i=0;mike.name[i]!='\0';i++)
        mike.name[i]=toupper(mike.name[i]);
    cout<<mike.id_num<<" "<<mike.name<<endl;
    return 0;
}
```

结构体变量赋值

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student mike1={123,{'m','i','k','e','\0'}};
    student mike2;
    mike2=mike1; //copy
    mike2.id_num=20130000+mike2.id_num;
    for(int i=0;mike2.name[i]!='\0';i++)
        mike2.name[i]=toupper(mike2.name[i]);
    cout<<mike1.id_num<<" "<<mike1.name<<endl;
    cout<<mike2.id_num<<" "<<mike2.name<<endl;
    return 0;
}
```

结构体做函数参数

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
void renew(student one)
{
    one.id_num=20130000+one.id_num;
    for(int i=0;one.name[i]!='\0';i++)
        one.name[i]=toupper(one.name[i]);
    cout<<one.id_num<<" "<<one.name<<endl;
}
int main()
{
    student mike={123,{'m','i','k','e','\0'}};
    renew(mike); //copy
    cout<<mike.id_num<<" "<<mike.name<<endl;
    return 0;
}
```

结构体做函数返回值

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
student newone()
{
    student one={20130123,{'M','I','K','E','\0'}},
    return one;
}
int main()
{
    student mike=newone(); //copy
    cout<<mike.id_num<<" "<<mike.name<<endl;
    return 0;
}
```

指向结构体的指针

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[20];
}
int main()
{
    student mike={123,{'m','i','k','e','\0'}};
    student *one=&mike;
    cout<<(*one).id_num<<" "<<(*one).name;
    return 0;
}
```

指向结构体的指针

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[20];
}
int main()
{
    student mike={123,{'m','i','k','e','\0'}};
    student *one=&mike;
    cout<<one->id_num<<" "<<one->name;
    return 0;
}
```

指向结构体的指针

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[20];
}
void renew(student *one)
{
    one->id_num=201300000+one->id_num;
    for(int i=0;one->name[i]!='\0';i++)
        one->name[i]=toupper(one->name[i]);
}
int main()
{
    student mike={123,{'m','i','k','e','\0'}};
    renew(&mike);
    cout<<mike.id_num<<" "<<mike.name;
    return 0;
}
```

结构体数组

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student myclass[3]=
    {123,{'m','i','k','e','\0'},
     133,{'t','o','m','\0'},
     143,{'j','a','c','k','\0'}};
    student * one=myclass;
    cout<<one->id_num<<" "<<one->name<<endl;
    one++;
    cout<<one->id_num<<" "<<one->name<<endl;
    return 0;
}
```

小结

结构体数据类型的特性与普通数据类型的特性一致

生日相同问题

- Description
  - 在一个有 100 人的大班级中，存在多个生日相同的同学概率非常大。
  - 现给出每个学生的学号，出生月日，请列出所有生日相同的同学。
- Input
  - 第一行为整数n，表示有n个学生，n<100。
  - 此后每行包含一个字符串和两个整数，分别表示学生的学号（字符串长度小于 10）和出生月（1<=m<=12）日（1<=d<=31）。
  - 学号、月、日之间用一个空格分隔。
- Output
  - 对每组生日相同的学生，输出一行。
  - 其中前两个数字表示月和日，后面跟着所有在当天出生的学生的学号，数字、学号之间都用一个空格分隔。
  - 对所有的输出，要求按日期从前到后的顺序输出。
  - 对生日相同的学号，按输入的顺序输出。

生日相同问题

- 数组统计

```
1 struct student
2 {
3     char ID[10];
4     int month;
5     int day;
6 } stu[100];
```

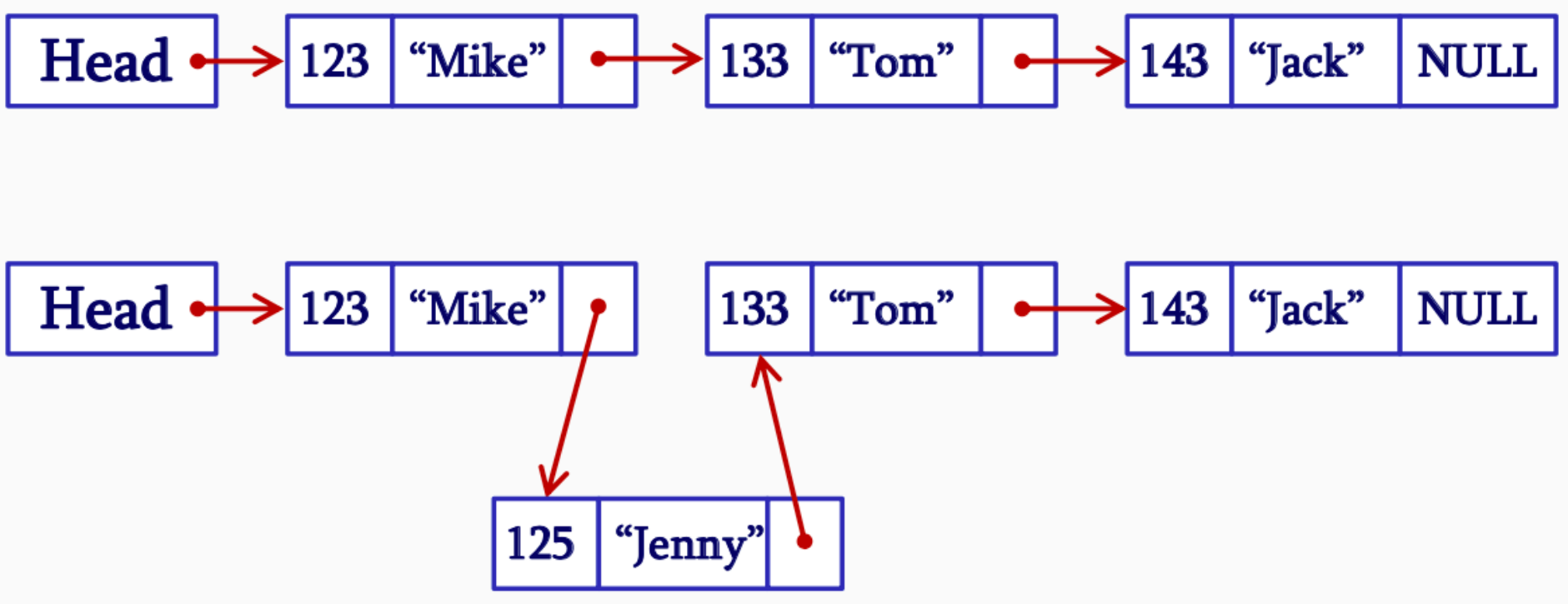
```
int main()
{
    int i,j,k,n,flag,count[100]={0};
    cout<<"How many students?"; cin>>n;
    for(int i=0;i<n;i++)
        cin>>stu[i].ID>>stu[i].month>>stu[i].day;
    for(int m=1;m<=12;m++)
        for(int d=1;d<=31;d++)
        {
            flag=0;j=0;
            for(int i=0;i<n;i++)
                if(stu[i].month==m&&stu[i].day==d)
                {count[++j]=i;flag++;}
            if(flag>1) {
                cout<<m<<" "<<d<<" ";
                for(k=1;k<=j;k++)
                cout<<stu[count[k]].ID<<" "<<endl;
            }
        }
    return 0;
}
```

内容提要 I

结构体数组

```
#include<iostream>
using namespace std;
struct student
{
    int id_num;
    char name[10];
};
int main()
{
    student myclass[3]=
    {123,{'m','i','k','e','\0'},
     133,{'t','o','m','\0'},
     143,{'j','a','c','k','\0'}};
    student * one=myclass;
    cout<<one->id_num<<" "<<one->name<<endl;
    one++;
    cout<<one->id_num<<" "<<one->name<<endl;
    return 0;
}
```

用指针把结构体链起来


Head
123
“Mike”
133
“Tom”
143
“Jack”
NULL
Head
123
“Mike”
133
“Tom”
143
“Jack”
NULL
125
“Jenny”

链表

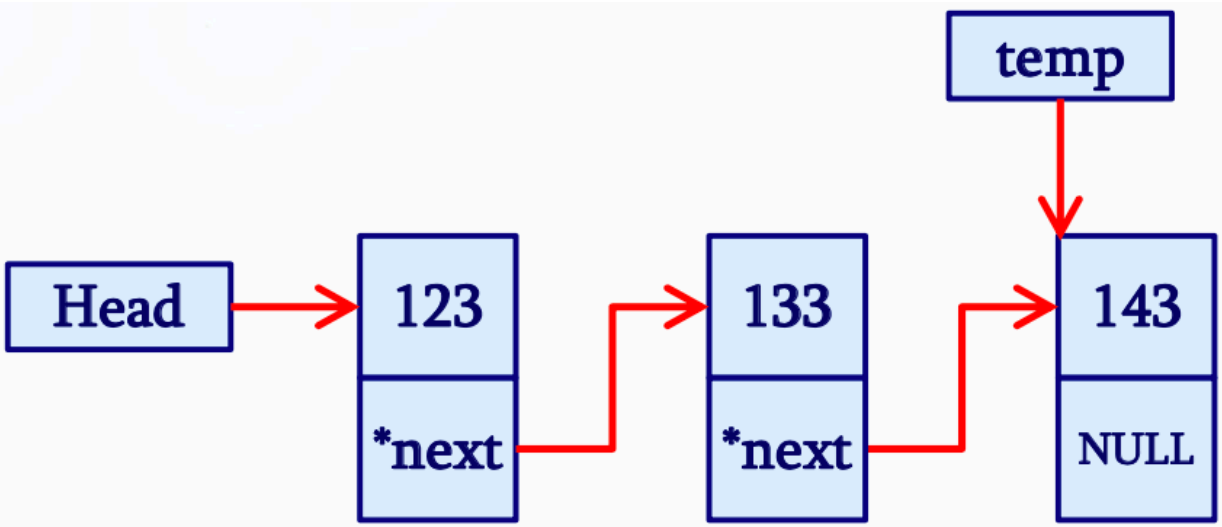
- 链表头：指向第一个链表结点的指针。
- 链表结点：链表中的每一个元素，包括：
  - 当前节点的数据；
  - 下一个结点的地址。
- 链表尾：不再指向其他结点的结点，其地址部分放一个NULL，表示链表到此结束。

链表可以动态地创建

- **动态**申请内存空间
  - `int *pint=new int(1024);` `delete pint;`
  - `int *pia=new int[4];` `delete [] pia;`
- **动态**建立链表节点

```
1 struct student
2 {
3     int id;
4     student *next;
5 };
6 student *head;
7 head = new student;
```

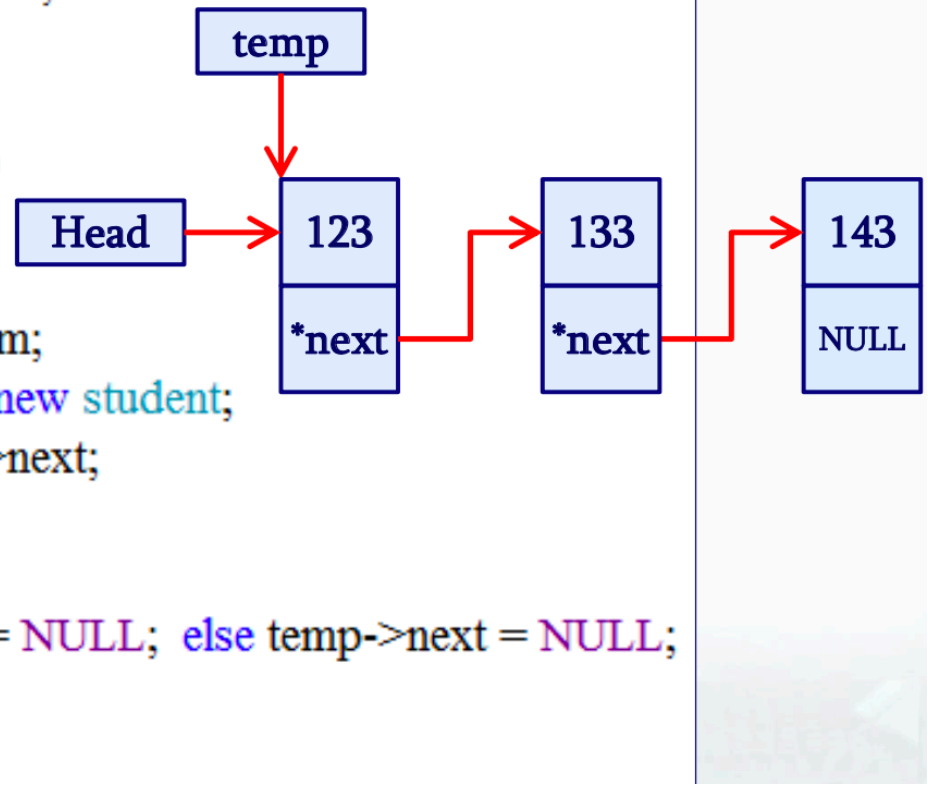
链表–创建



- Step1:
  - head = new student;
  - student *temp = head;
- Step2:
  - Continue?
- Y:
  - temp->next = new student;
  - temp = temp->next
  - goto Step2
- N:
  - temp->next = NULL

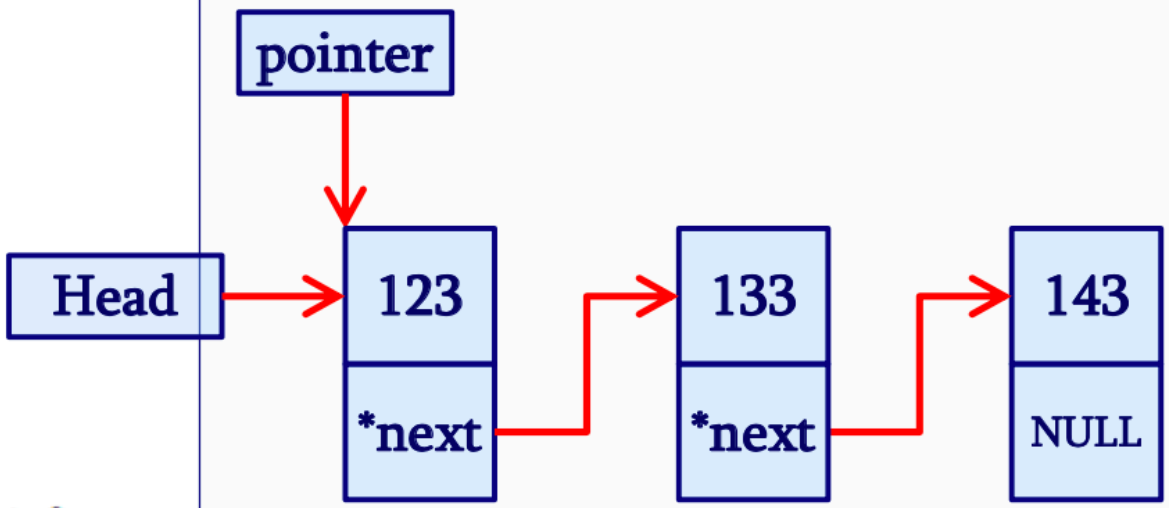
链表–创建

```
student *create()
{
   student *head, *temp;   int num, n = 0;
   head = new student;
   temp = head;
   cin >> num;
   while (num != -1)
   {
      n++;
      temp->id = num;
      temp->next = new student;
      temp = temp->next;
      cin >> num;
   }
   if (n == 0) head = NULL;  else temp->next = NULL;
   return head;
}
```



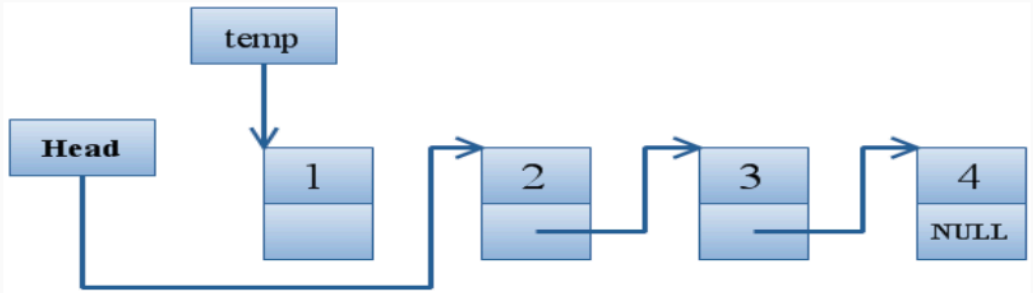
链表–遍历

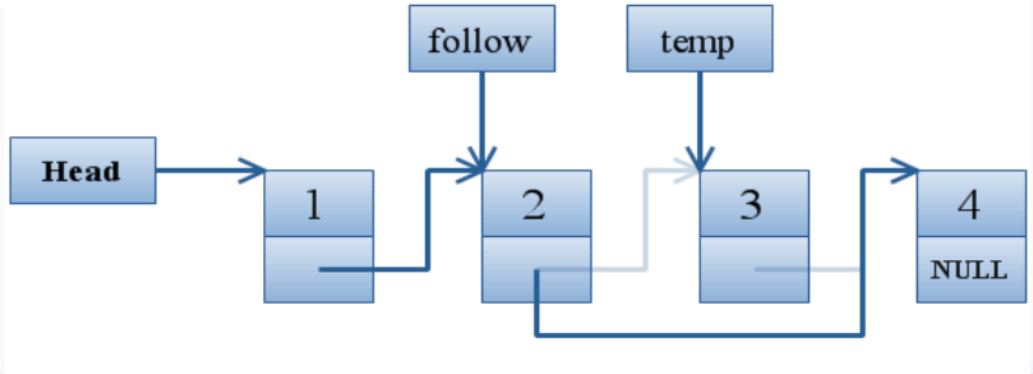
```
#include<iostream>
using namespace std;
struct student
{
    int id;
    student *next;
};
student *create();
int main()
{
    student *pointer = create();
    while (pointer->next != NULL)
    {
        cout << pointer->id <<endl;
        pointer = pointer->next;
    }
    return 0;
}
```



链表–删除

temp=head; head = head→next; delete temp;





follow→next = temp→next; delete temp;

链表–删除

在链表中将值为n的元素删掉

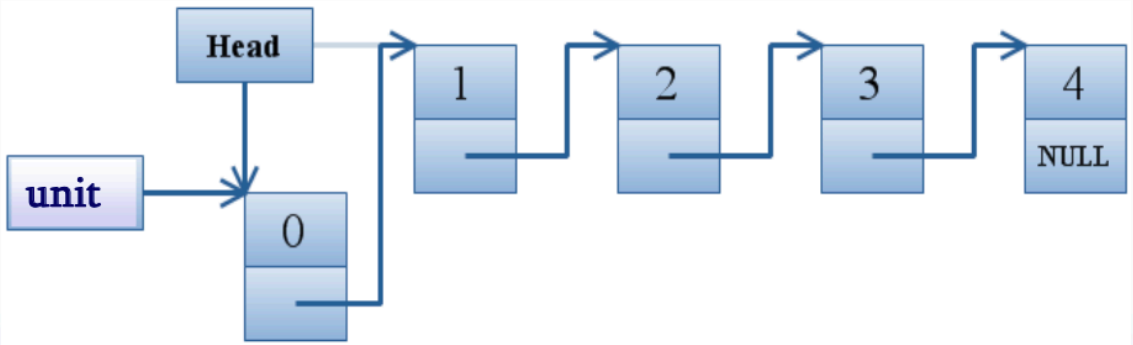
```
linker *dele(student *head; int n)
{   student *temp,* follow;
    temp = head;
    if (head == NULL) {                    //head为空，空表的情况
        return(head);
    }
    if (head->num = = n) {                 //第一个节点是要删除的目标；
        head = head->next;
        delete temp;
        return(head);
    }
    while(temp != NULL && temp->num != n){     //寻找要删除的目标；
        follow= temp;
        temp = temp->next;
    }
    if (temp == NULL) cout<<“not found”;       //没寻到要删除的目标；
    else {
        follow->next =temp->next;              //删除目标节点；
        delete temp;
    }
    return(head);
}
```

链表–插入

- 将结点unit插入链表：

插在最前面的情况

```
unit->next = head;
head = unit;
```

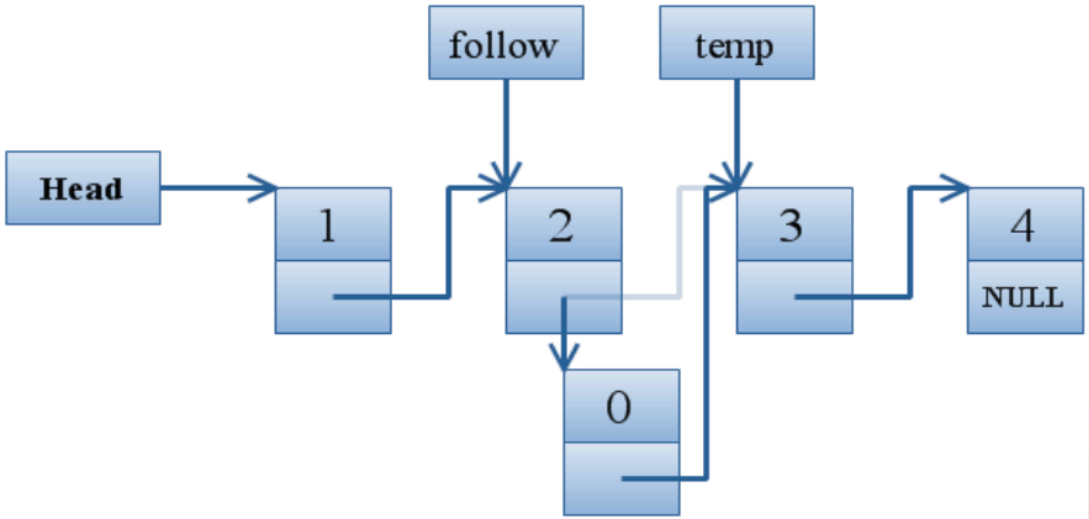


链表–插入

- 将结点unit插入链表：

插在中间的情况

```
unit->next = temp;
follow->next = unit;
```

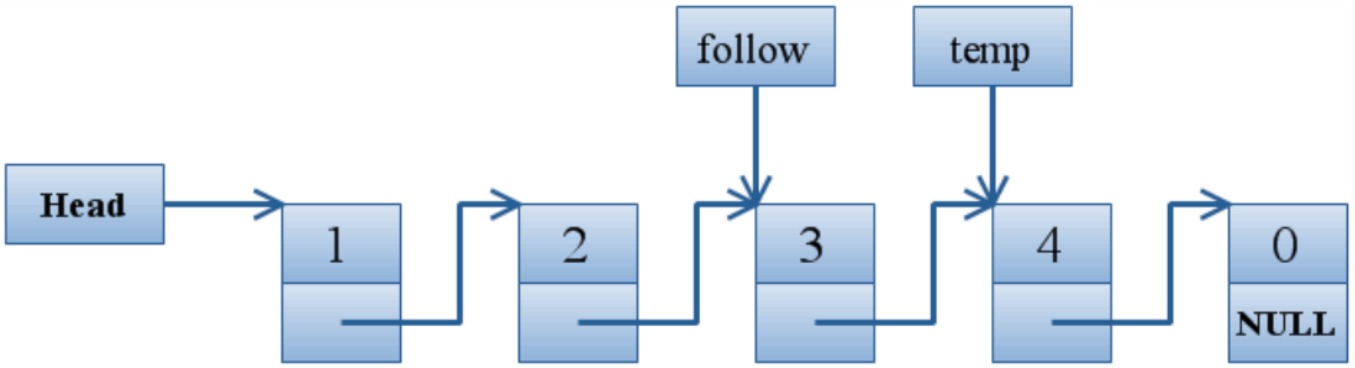


链表–插入

■ 将结点unit插入链表：

插在最后面的情况

```
temp->next = unit;
unit->next = NULL;
```

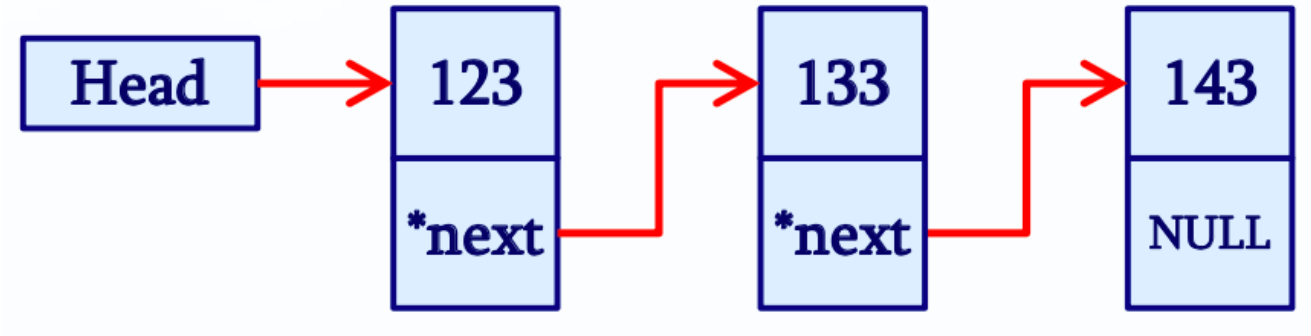


链表–插入

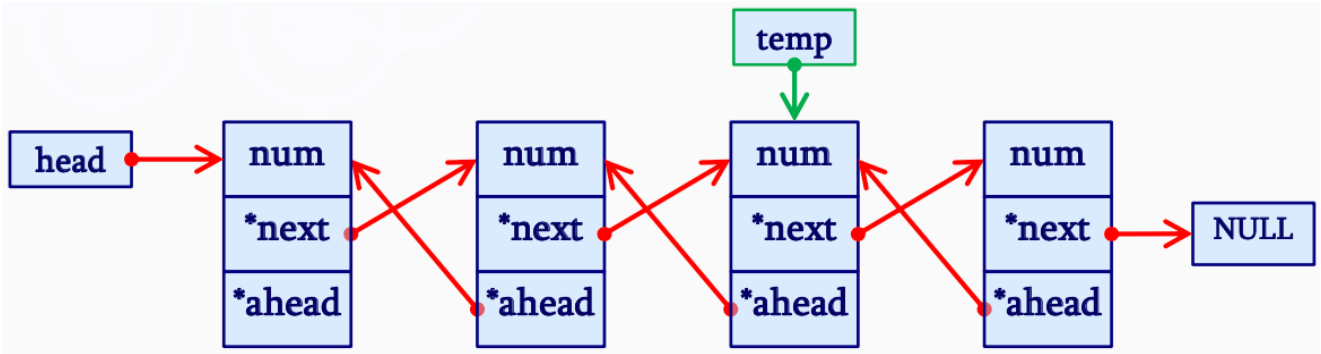
```
Student *insert(student *head; int n) {
    student *temp,*unit,*follow;                    //插入结点值为n的结点
    temp = head;   unit = new student;
    unit->num = n;    unit->next = NULL;
    if (head==NULL) {                               //如果链表为空，直接插入
        head=unit;
        return(head);
    }                                   //寻找第一个不小于n或结尾的结点temp
    while((temp→next! = NULL)&&(temp->num < n)){
        follow=temp;  temp=temp->next;
    }
    if (temp==head){                                //如果temp为第一个结点
        unit->next=head; head=unit;
    }
    else{                                           //如果temp为最后一个结点
        if( temp->next == NULL) temp->next = unit;
        else{                                       //如果temp为一个中间结点
            follow->next=unit; unit->next=temp;
        }
    }
    return(head);
}
```

单向链表和双向链表

- 单向链表

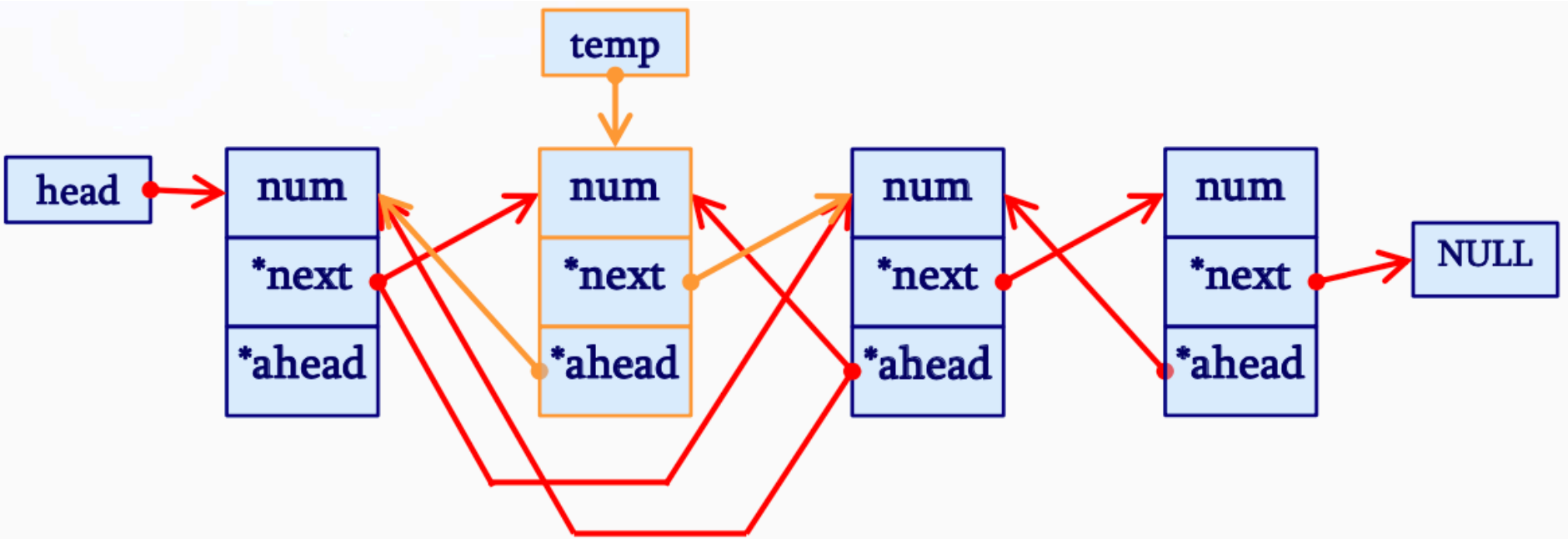


- 双向链表



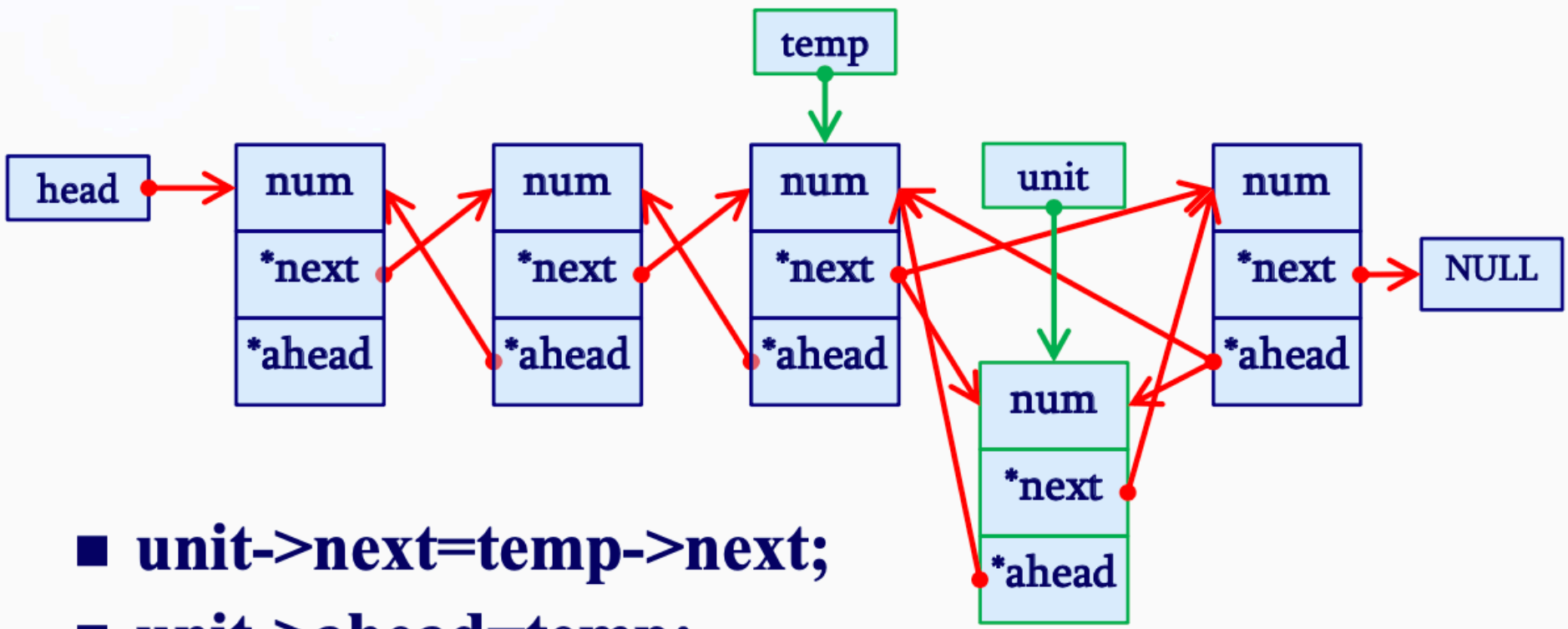
`temp->num` `temp->next` `temp->ahead`

双向链表



- temp->ahead->next=temp->next;
- temp->next->ahead=temp->ahead;

双向链表

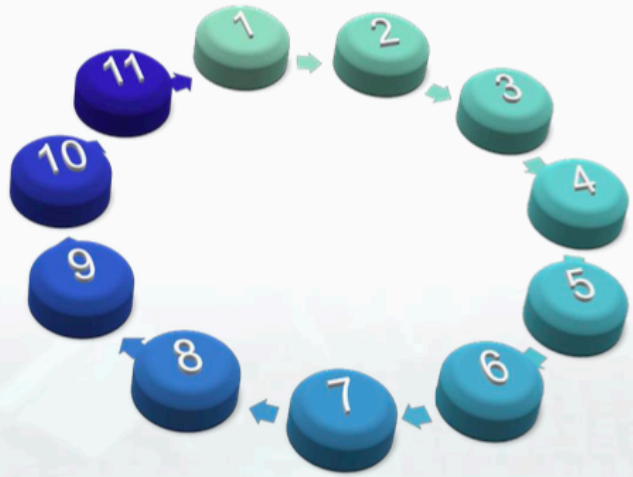


- unit->next=temp->next;
- unit->ahead=temp;
- temp->next->ahead=unit;
- temp->next=unit;

约瑟夫问题

问题描述

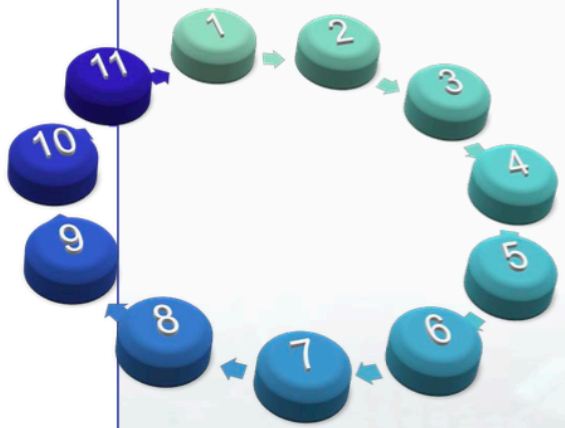
- 编号为1 – N的N个人围坐在一起形成一个圆圈，从第P个人开始，依次按照顺时针的方向报数，数到第M的人出列，直到最后剩下一个人。
- 请写一个程序，对于给定的 N, P, M，计算并打印出依次出列的人的编号。



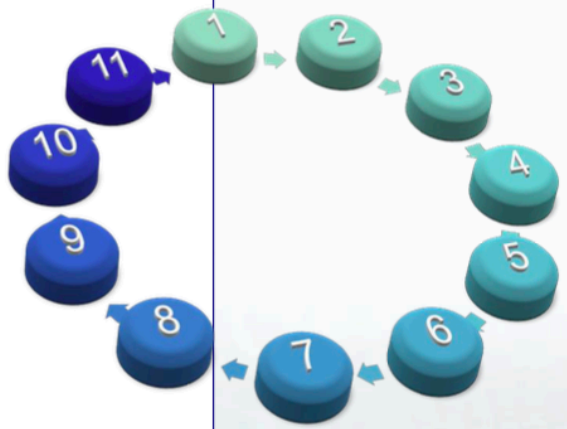
约瑟夫问题

```
#include<iostream>
using namespace std;
struct Node
{
    int num;
    Node * ahead;
    Node * next;
};
Node *Create(int N);
Node *Search(Node *head, int P);
Node *Release(Node *head, int M);
int main()
{
    int N, P, M = 0;                //N-起始节点数，P-开始节点
    cin >> N >> P >> M;             //每次释放第M个节点
    Node * head = Create(N); //创建N个节点的环
    head = Search(head, P);     //找到第P个节点
    while (head->next != head)  //不断释放第M个元素
    {                                       //直到只剩一个元素
        head = Release(head, M); //释放第M个节点
    }
    cout << head->num;
    return 0;
}
```

```
11 3 7
9,5,2,11,10,1,4,8,3,6,7
```

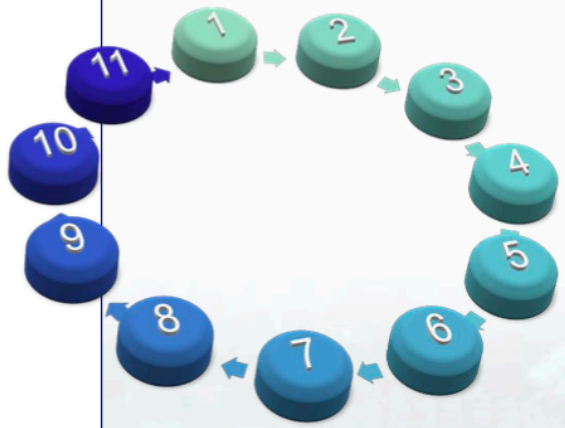


```
Node *Create(int N)  //创建包含N个节点的双向循环链表
{
    int n = 1;
    Node * node = new Node;
    node->num = n;
    Node * head = node;     //指向第一个节点
    Node * tail = head;     //指向最后一个节点
    while (n++ < N)
    {
        node = new Node;    //建新节点
        node->num = n;      //赋值
        tail->next = node;  //接入新节点
        node->ahead = tail;
        tail = tail->next;  //尾巴后移
    }
    tail->next = head;      //尾巴处理
    head->ahead = tail;
    return head;
}
```



```
Node *Search(Node *head, int P) //从Head开始寻找第P个节点
{
   while (head->num != P)
   {
      head = head->next;
   }
   return head;
}

Node *Release(Node *head, int M) //释放Head开始的第M个节点
{
   int count = 1;
   Node *temp = head;
   while (count < M)        //寻找第M个节点
   {
      temp = temp->next;
      count++;
   }
   temp->ahead->next = temp->next;     //移除第M个节点
   temp->next->ahead = temp->ahead;  //移除第M个节点
   cout << temp->num << ";";
   head = temp->next;   //释放第M个节点所占内存空间
   delete temp;
   return head;
}
```



内容提要 I

内容提要 I

```
struct student
{
    int id;             //学号
    char name[20];      //姓名
    char sex;           //性别
    int age;            //年龄
    float score;        //入学成绩
    char addr[30];      //宿舍地址
} stu1;
```


选课管理系统

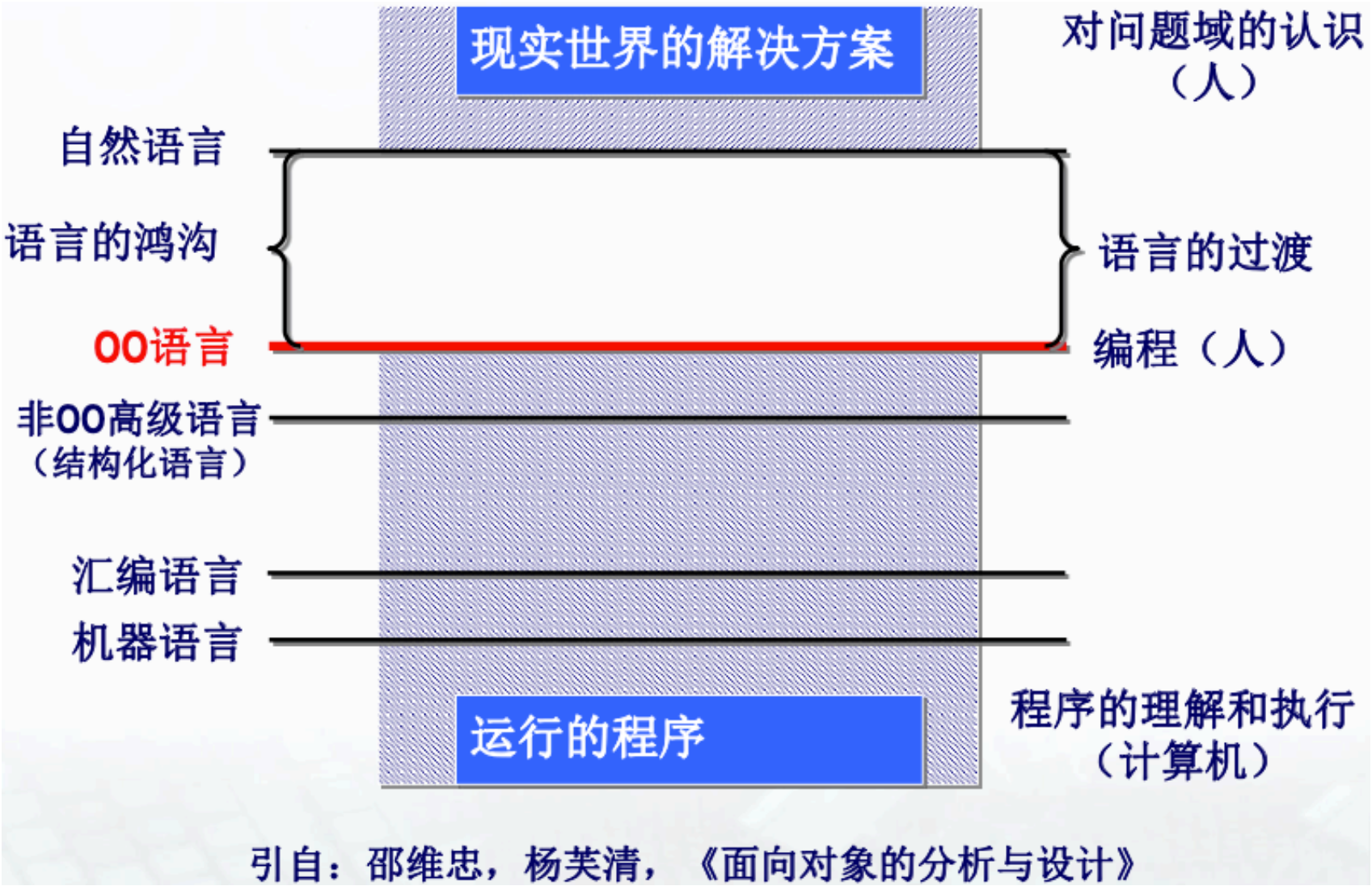
软件的本质

现实世界的解决方案在计算机系统中的映射

软件的本质

现实世界的解决方案在计算机系统中的映射

程序设计语言的发展



引自：邵维忠，杨芙清，《面向对象的分析与设计》

接下来的问题

- 怎么理解软件的本质？
- 什么是对象？
- 什么是面向对象的程序设计？
- 面向对象语言的作用？
- 如何通过面向对象的视角来观察世界？
- 如何进行面向对象的分析？
- 如何进行面向对象的设计？
- 如何进行面向对象的编程？
- ……