程序设计基础

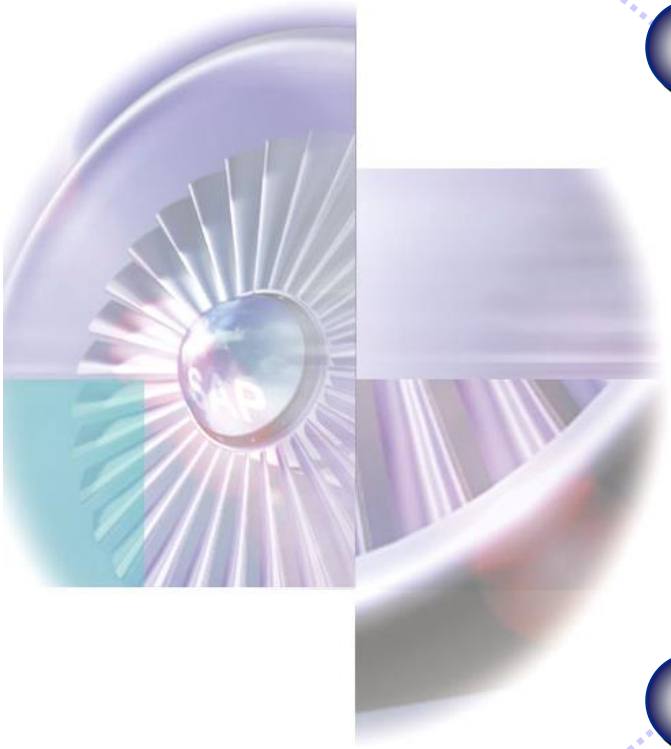
Fundamental of Programming

清华大学软件学院

刘玉身

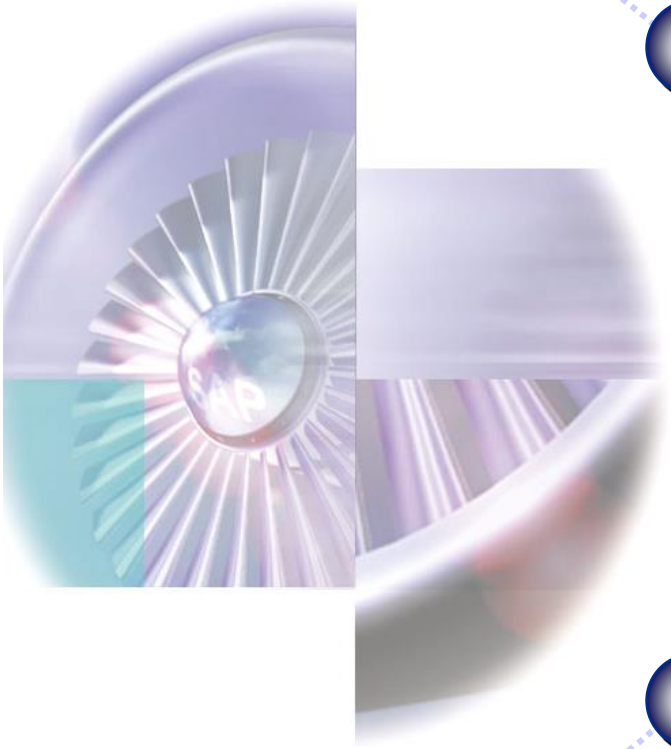
liuyushen@tsinghua.edu.cn

Lecture 7: 指针



1. 基本概念
2. 指针变量
3. 指针与数组
4. 指针与字符串

Lecture 7: 指针



1. 基本概念
2. 指针变量
3. **指针与数组**
4. 指针与字符串

回顾

- C语言程序设计中使用指针的好处
  - 使程序简洁、紧凑、高效
  - 有效地表示复杂数据结构（如: 链表、八叉树）
  - 动态分配内存
  - 得到函数的多个返回值

数组元素的访问

数组的定义： **int a[5];**

空间的分配：

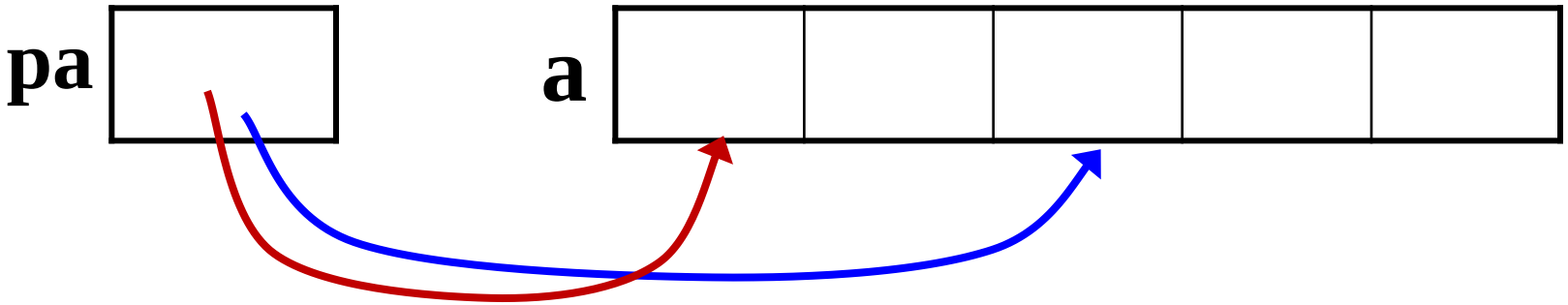
	a[0]	a[1]	a[2]	a[3]	a[4]
a					

元素的访问：

```
for(i = 0; i < 5; i++)
{
    a[i] = 0;
}
```

通过指针来访问数组元素

指针　　数组



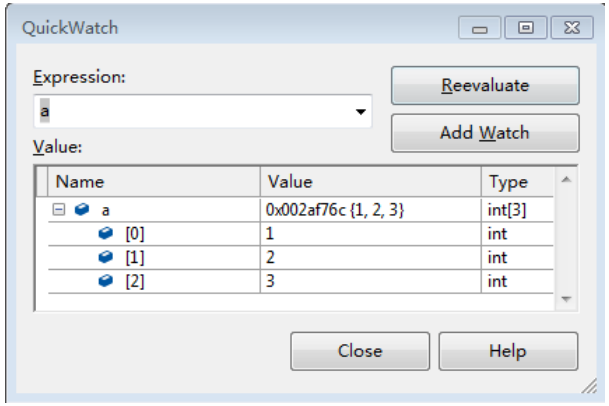
pa = &a[2]

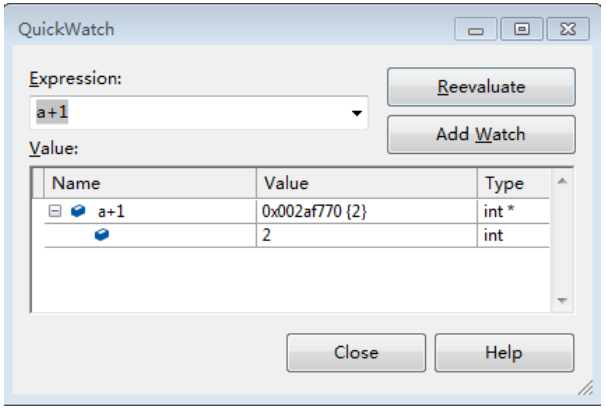
pa = a 等价于 **pa = &a[0]**

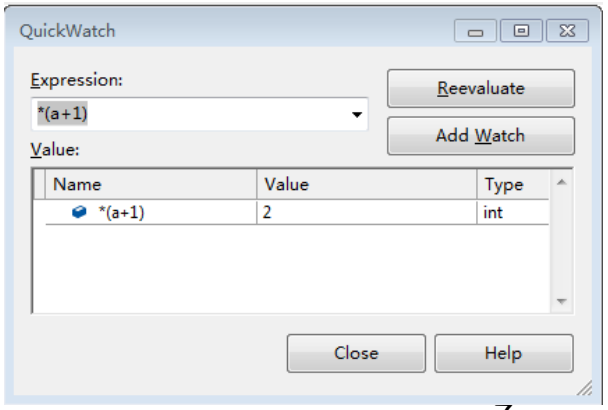
程序示例

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a[3] = {1, 2, 3};

    int *p = a;
    int b = *a;
    int c = *(a+1);

    printf("%d %d %d", *p, b, c);
    return 0;
}
```

```
1 1 2
```







指针的算术和关系运算

指针加一个整数： **ptr + k**

指针减一个整数： **ptr – k**

两个指针相减： **p1 – p2**

指针的关系运算： >、>=、<
<=、==、!=

- 指针的算术运算是以数据元素为单元；
-
```
int *p;
p = (int *)1000;
p = p + 2;
```
- p+k的计算方法是：p+k*sizeof(int)
  基类型长度的补偿由系统自动完成。

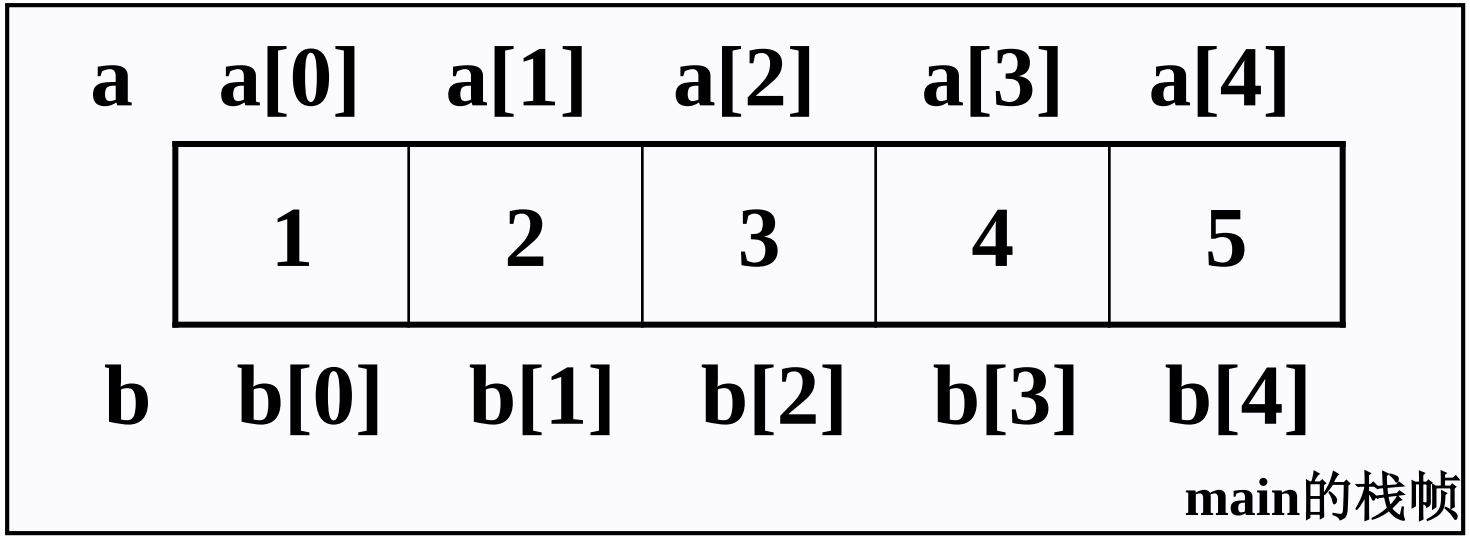
```
int a[5], *pa;
```

想做的事情	编写的代码
把a的起始地址放入pa	pa = a; pa = &a[0];
pa指向第i个数组元素	pa = &a[i]; pa = a + i;
访问第i个数组元素	*(pa + i) pa[i]

```c
#include <stdio.h>
void shift(int a[], int N);
int main()
{
    int N=0, b[20], i, M;

    while(1)
    {
        scanf("%d", &b[N]);
        if(b[N] == 0) break;
        else N++;
    }
    scanf("%d", &M);
    for(i = 1; i <= M; i++) shift(b, N);
    for(i = 0; i < N; i++) printf("%d ",b[i]);
}
```

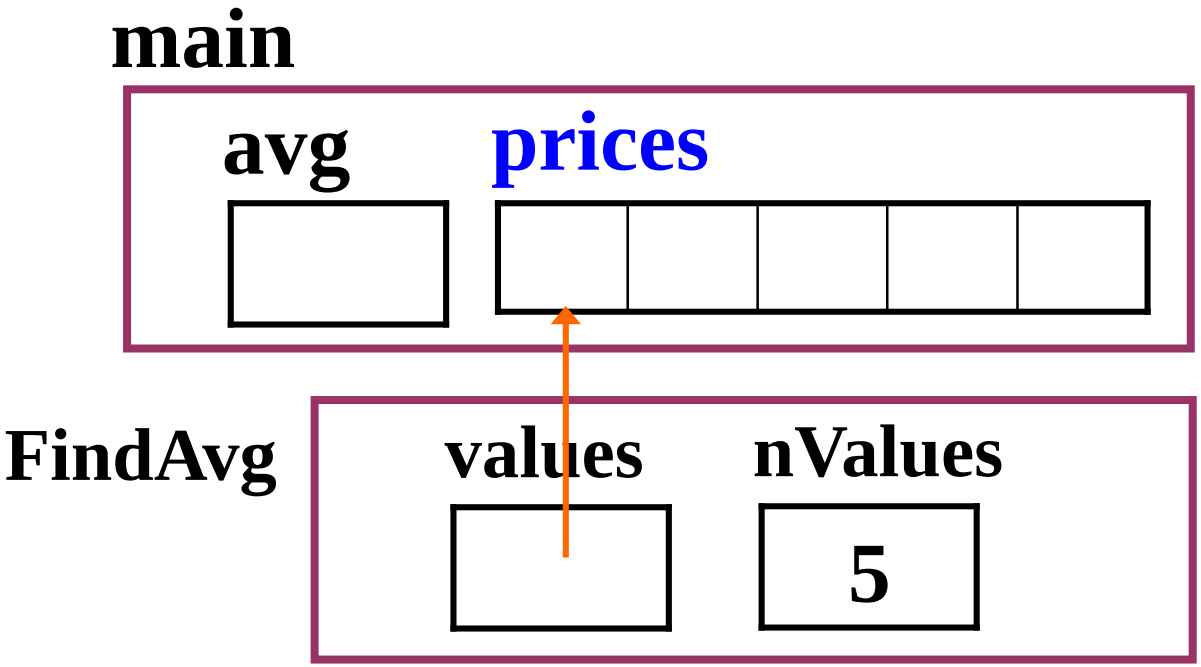
```c
void shift(int a[], int N)
{
    int temp, k;

    temp = a[N-1];
    for(k = N-1; k > 0; k--)
    {
        a[k] = a[k-1];
    }
    a[0] = temp;
}
```

在函数调用时，传地址而不传值



why?

```
int main( )
{
    double prices[5], avg;
    avg = FindAvg(prices, 5);
}
double FindAvg(double values[], int nValues)
{
}
```



调用子函数时，并没有真正创建一个数组。仅是实参数组首地址传递给形参数组变量。

动态数组

在定义一个数组时，必须事先指定其长度。

例如：

```
int   a[6];
```

或者：

```
#define   MAX_SIZE   100;
int   scores[MAX_SIZE];
```

如果在编程时并不知道数组的确切长度？

☺太小了：装不下

☺太大了：浪费内存空间

有时，数组的长度只有当程序运行以后才知道。因此，我们希望能这样定义数组：

```
int  n;
printf("请输入学生人数：");
scanf("%d", &n);
int scores[n];
```

但是在C99之前的版本，这是不可能的。

回顾：变长数组（C99）

- C99标准中新增了变长数组(Variable length arrays)
  - 数组 (a[N]) 的长度可以是一个变量 (int N)
  - 但数组长度变量 (N) 的值必须在runtime 赋值
- C99标准更新:
  - http://www.comeaucomputing.com/techtalk/c99/#bool

- 但VC中的C编译器现在还不支持C99
- 如要在ANSI C中实现可变长度数组，可用动态数组 (malloc( ) / free( ) ， C++是new / delete)，后续课程会介绍

变长数组示例 (C99)

```
#include <stdio.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    int a[n];
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    return 0;
}
```

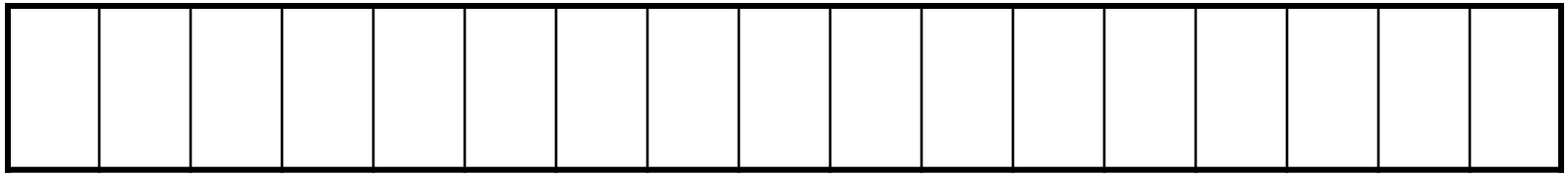
我们能做的事情是：动态地为该数组分配所需的内存空间，即在程序运行时分配。具体做法是：定义一个指针，然后把动态分配的内存空间的起始地址保存在该指针中，如：

```
int   *scores;
scores = 动态分配的内存空间的起始地址;
```

动态数组

void *malloc(size_t size)

参数：申请的字节数；
功能：申请一块内存；
返回值：一个指向该内存的指针或NULL；
头文件：#include <stdlib.h>



字节为单位...

典型编程方式

```
#include <stdlib.h>  // 头文件
```

```
int   *scores,  N;

scanf("%d",   &N);
scores  =  (int *)malloc(N * sizeof(int));
if(scores  ==  NULL) ...  /* 内存分配失败 */
scores[3]  =  99;
```

等价于：*(scores + 3) = 99;

动态数组实现示例

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    int *a = (int *)malloc( n * sizeof(int));
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    free(a);
    return 0;
}
```

作为局部变量的数组在函数调用结束后，其内存空间即被释放。而对于**动态数组**，即使在函数调用结束后，其**内存空间依然存在**。

程序分析

```
#include <stdio.h>
#define   MAX_SIZE   10
int *Add(int a[], int b[], int num)
{
    int i, c[MAX_SIZE];
    for (i = 0;  i < num;  i++)
        c[i] = a[i] + b[i];
    return c;
}
int main( )
{
    int a[5] = {1, -1, 2, -2, 0};
    int b[5] = {3, 1, -2, 4, 1};
    int *c,  i;
    c = Add(a, b, 5);
    for(i = 0;  i < 5;  i++)
        printf("%d ", c[i]);
}
```

```
4 4344416 1244916 4198895 1
```

程序分析

```
#include <stdio.h>
#define   MAX_SIZE   10
int *Add(int a[], int b[], int num)
{
    int i, c[MAX_SIZE];
    for (i = 0;  i < num;  i++)
        c[i] = a[i] + b[i];
    return c;
}
int main( )
{
    int a[5] = {1, -1, 2, -2, 0};
    int b[5] = {3, 1, -2, 4, 1};
    int *c,  i;
    c = Add(a, b, 5);
    for(i = 0;  i < 5;  i++)
        printf("%d ", c[i]);
}
```

```
Before printf ……

C[] =

4 0 0 2 1 -858993460 …
```

```
After printf ……

C[] =

4 -160183613 4848044
4847756 2130567168 …
```

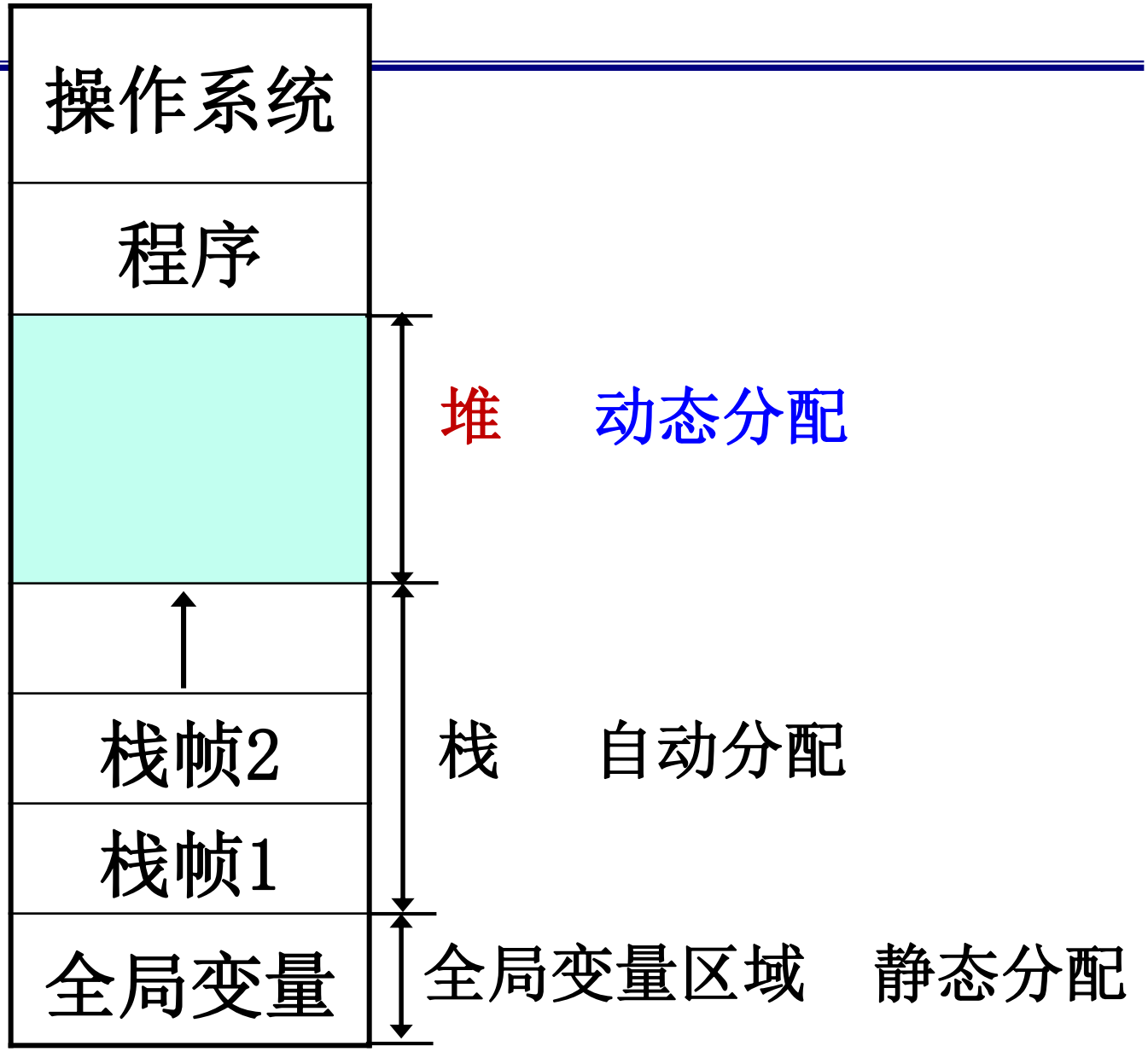
系统重新使用了临时变量空间！

```
int main( )
{
    int a[5] = {1, -1, 2, -2, 0};
    int b[5] = {3, 1, -2, 4, 1};
    int *c,  i;
    c = Add(a, b, 5);
    if(c == NULL) return;
    for(i=0; i<5; i++) printf("%d ", c[i]);
}
int *Add(int a[], int b[], int num)
{
    int i, *c;
    c =(int*)malloc(MAX_SIZE*sizeof(int));
    if(c == NULL)  return(NULL);
    for (i = 0;  i < num;  i++)
        c[i] = a[i] + b[i];
    return c;
}
```

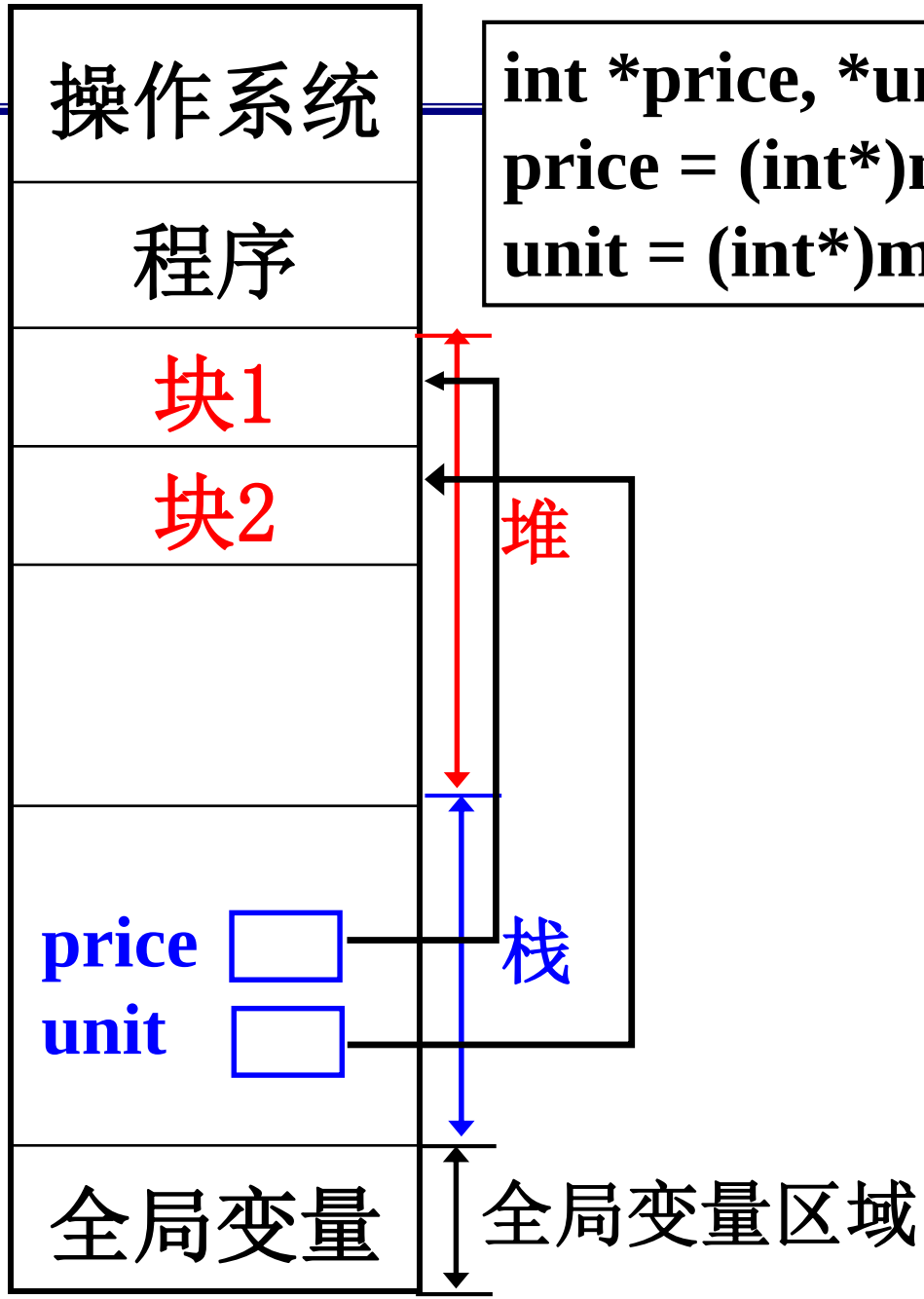
4 0 0 2 1

还有何问题？*thinking…*

内存分布状况



内存分布状况

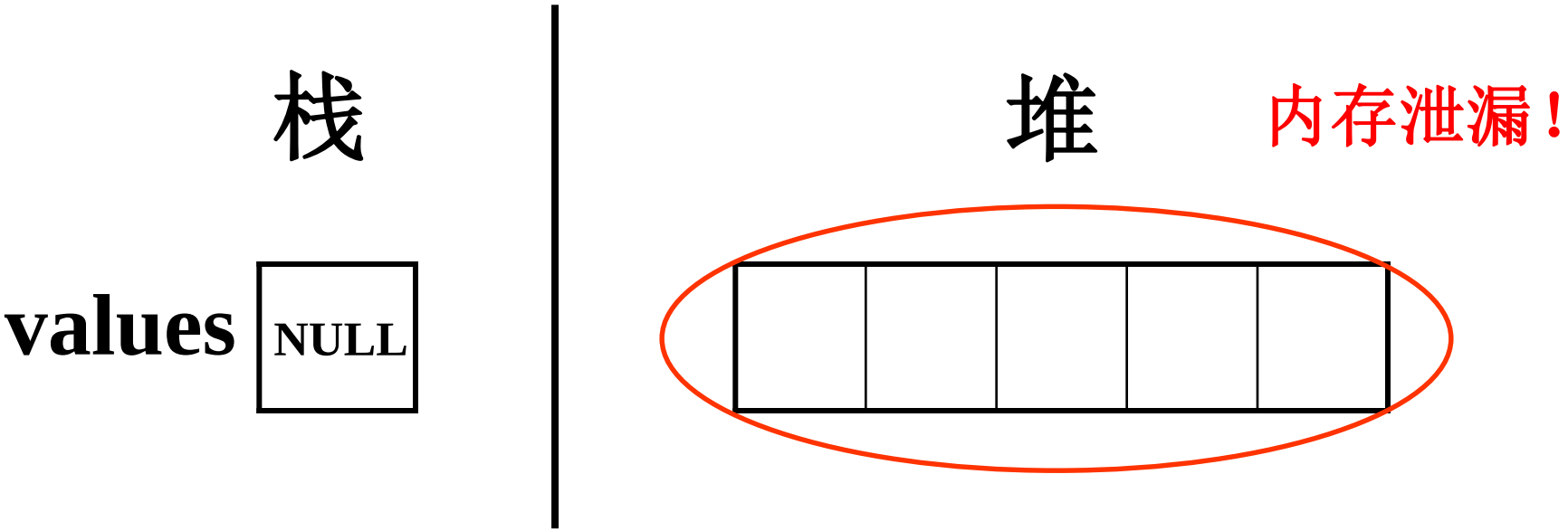



```
int *price, *unit;
price = (int*)malloc(n*sizeof(int));
unit = (int*)malloc(n*sizeof(int));
```

动态分配

```
values = (int *)malloc(5 * sizeof(int));
free(values);
values = NULL;
```

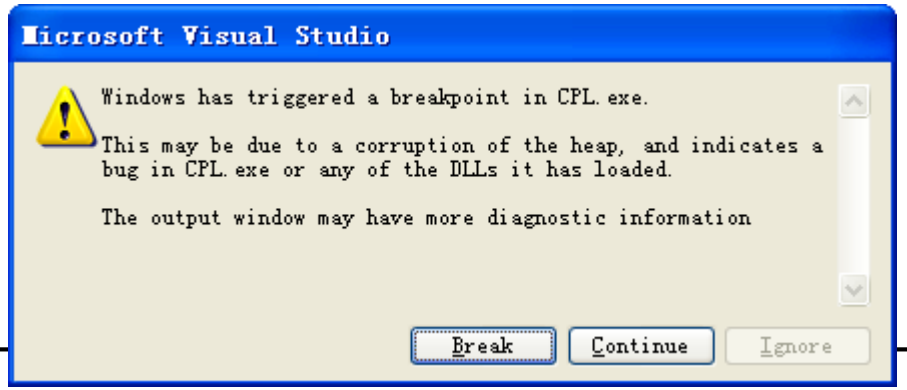
values原来指向的内存单元现在无法访问。



free(p)函数中参数p是一个指针，指向待释放内存空间的起始地址。注: p应指向之前malloc分配的首地址

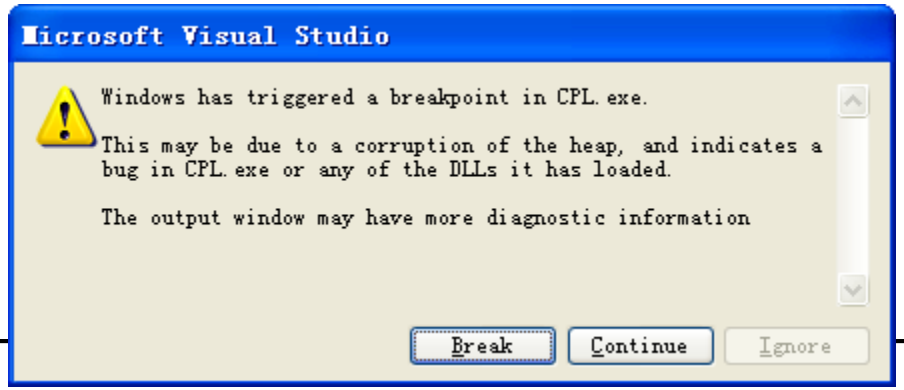
```
int main()
{
    int *pnums;

    pnums = (int*)malloc(9*sizeof(int));
    pnums ++;
    free(pnums);
    return 0;
}
```

有何问题？



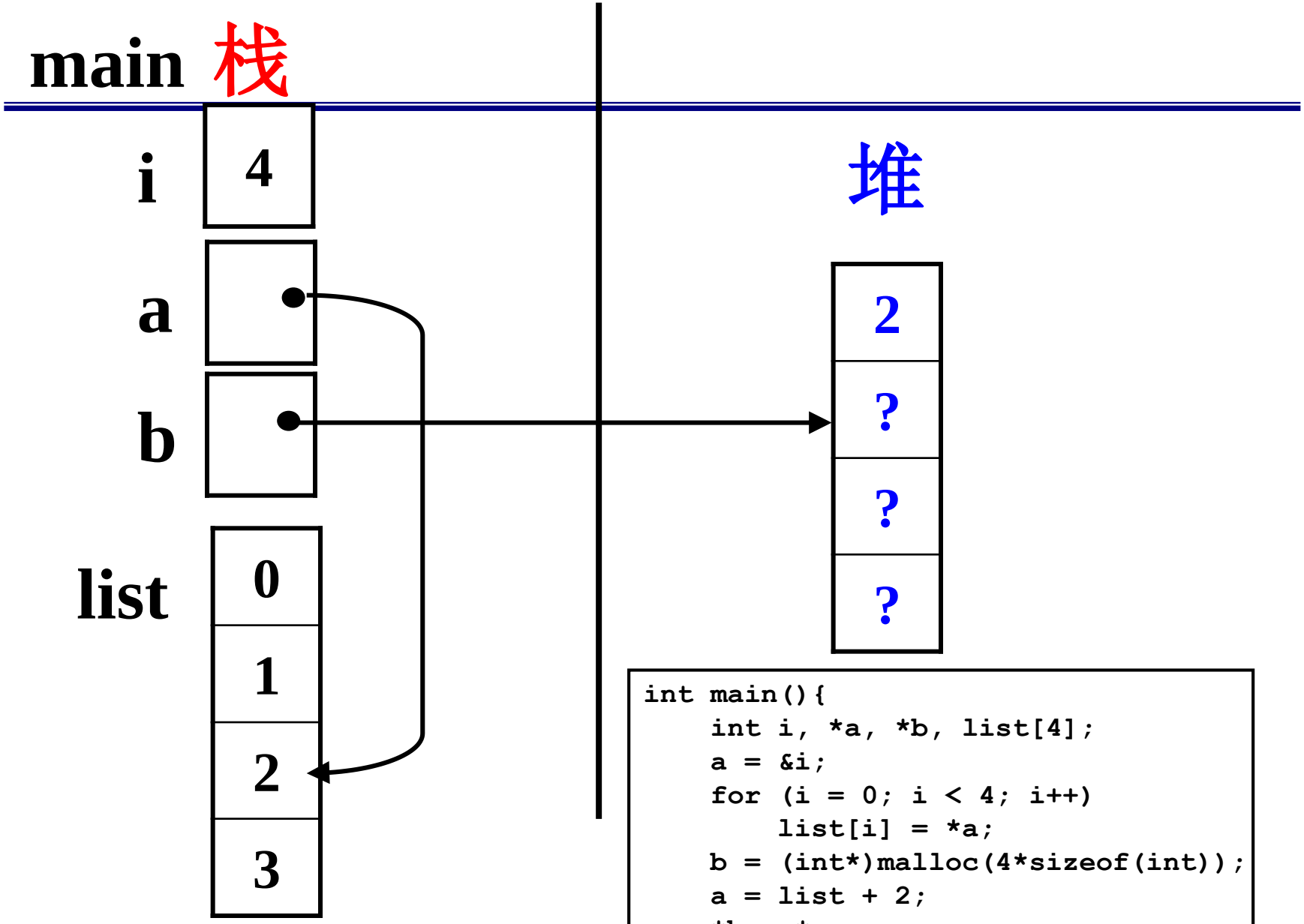
```
int main()
{
    int *pnums, *plist;

    pnums = (int*)malloc(9*sizeof(int));
    plist = pnums;
    free(plist);
    free(pnums);
    return 0;
}
```

有何问题？



代码分析

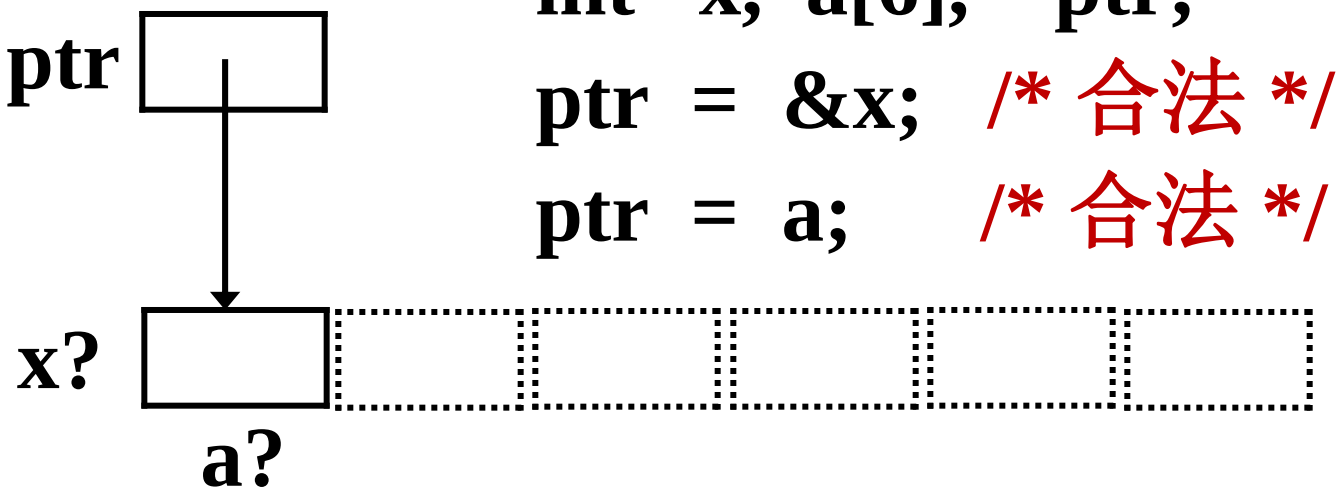
```
int main()
{
    int i, *a, *b, list[4];
    a = &i;
    for (i = 0; i < 4; i++)
        list[i] = *a;
    b = (int*)malloc(4*sizeof(int));
    a = list + 2;
    *b = *a;
    b++;
    //画出此时内存状态（栈、堆、变量的取值）
}
```




```
int main(){
    int i, *a, *b, list[4];
    a = &i;
    for (i = 0; i < 4; i++)
        list[i] = *a;
    b = (int*)malloc(4*sizeof(int));
    a = list + 2;
    *b = *a;
    b++;
}
```

指针的两种用法

一个指针可能是指向单一的一个变量，也可能是指向一组相同类型的数组元素，从外观上，两种指针的形式完全一样。




```
int   x,  a[6],  *ptr;
ptr  =  &x;   /* 合法 */
ptr  =  a;    /* 合法 */
```

分析结果

```
void bar(int p2[ ])
{
    p2[1] = 15;
}

void foo(int p1[ ])
{
    *p1 += 5;
}
```

```
int main( )
{
    int a[ ] = {1, 3, 5};
    int b[ ] = {2, 4, 6};
    int  *p;
    p = &a[0];
    bar(p);
    printf("%d %d %d\n", a[0], a[1], a[2]);
    p = &b[0];
    p ++;
    foo(p);
    bar(p);
    printf("%d %d %d\n", b[0], b[1], b[2]);
}
```

```
1   15   5
2   9    15
```

指针数组

一个数组，其元素均为指针类型变量，称为**指针数组**。即数组中的每一个元素都是一个指针。

定义形式：**类型名　*数组名[数组长度];**

例如：

```
int   *pa[4];
```

指针数组作为main函数的形参

int main(int argc, char *argv[ ]);

例如，假设程序名为sort，在运行时命令行的情况如下：

sort source.txt destination.txt

argv[0] argv[1] argv[2]

argc = 3

命令行参数

int main(int argc, char *argv[ ]);

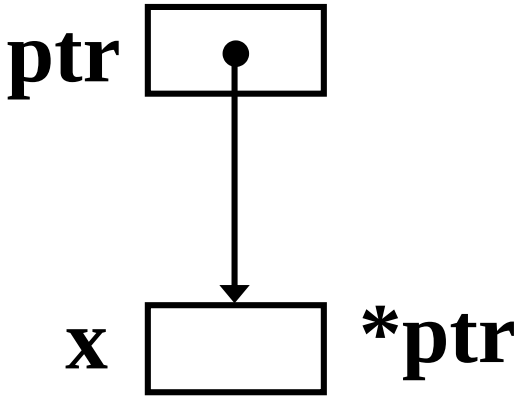
- **int argc** 表示命令行参数的个数（包括可执行程序名本身），
- **char *argv[]** 表示每个参数的具体内容，**argv[0]** 为命令行中可执行程序名本身，**argv[1]** 为命令行中第二个参数的内容，依次类推。

```
int main (int argc,char *argv[])
{
   int i;
   printf("\n命令行中可执行文件名为：%s", argv[0]);
   printf("\n总共有%d个参数：", argc);
   i = 0;
   while (argc >= 1)
   {
      printf("%s   ", argv[i++]);
      argc--;
   }
   return 0;
}
```

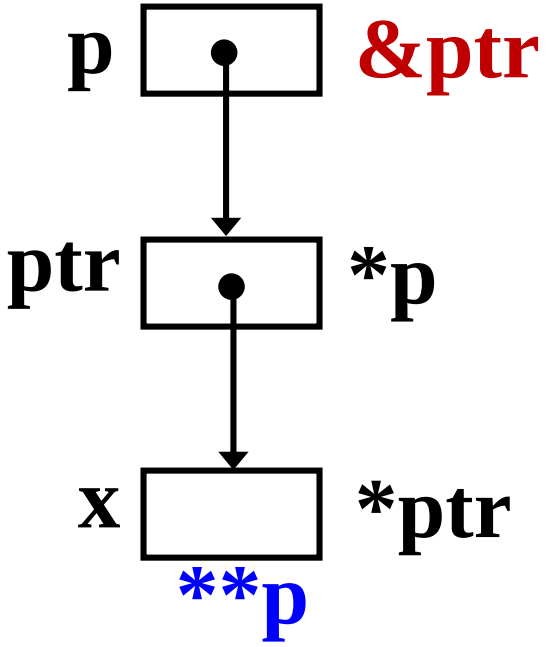
指向指针的指针

一种特殊的指针，其基类型也为指针类型。如 int **p。

通常的指针：



指向指针的指针：



已有的三种指针类型：

int * “pointer to int”

double * “pointer to double”

char * “pointer to char”

现在又可以增加三种新的数据类型：

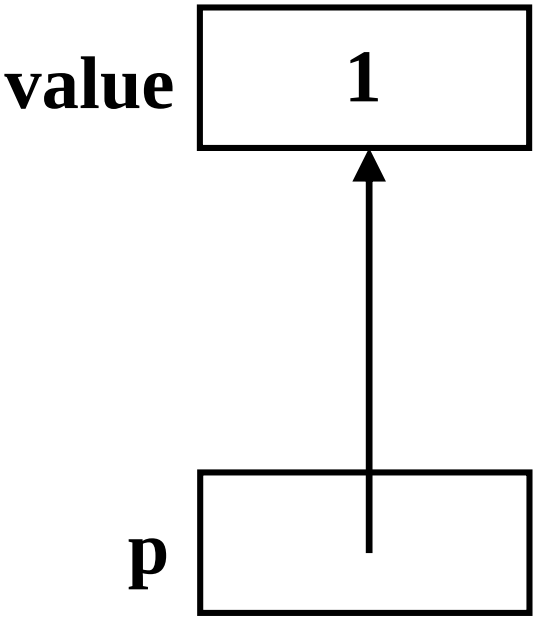
int ** “pointer to pointer to int”

double ** “pointer to pointer to double”

char ** “pointer to pointer to char”

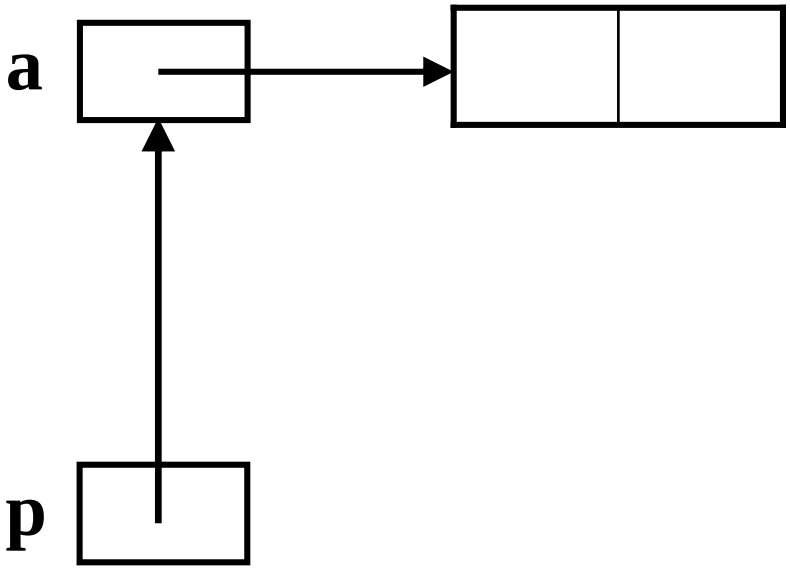
在foo函数中修改主函数中的变量

```
int main()
{
    int value;
    foo(???);
}

void foo(???) //p
{
    ??? = 1;
}
```

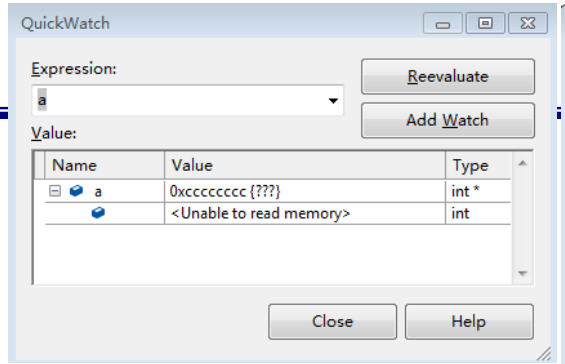


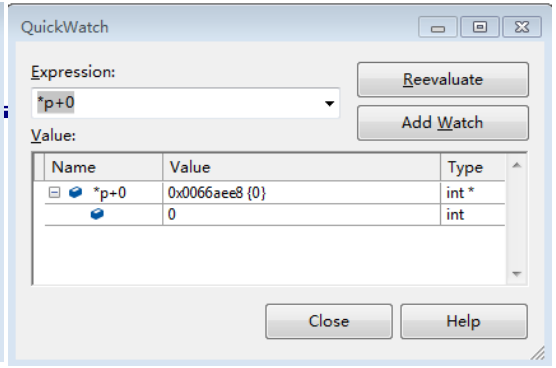
如果主函数中的变量是一个指针呢？

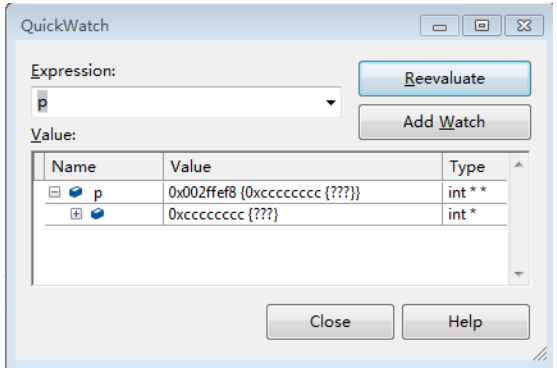
```
int main()
{
    int *a;
    foo(???);
}

void foo(???) //p
{
    ??? = malloc(8);
}
```

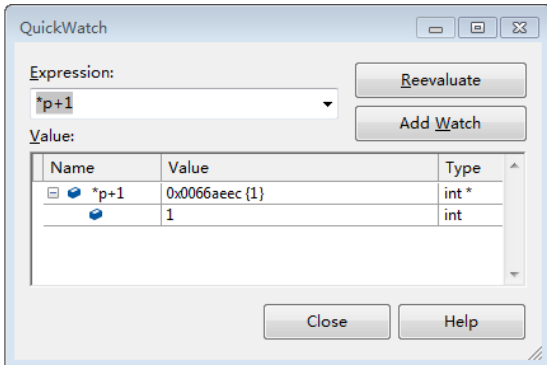


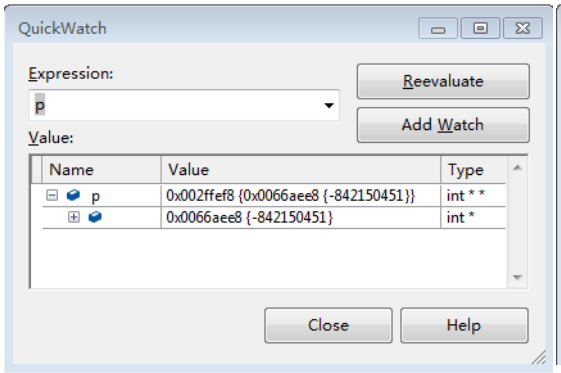
```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

int foo(int **p);

int main()
{
    int *a;
    foo(&a);
    return 1;
}

int foo(int **p) //p
{
    int i;
    *p = (int *) malloc(3*sizeof(int));
    for (i = 0; i < 3; i++)
        *(*p+i) = i;

    for (i = 0; i < 3; i++)
        printf("%d ", *(*p+i));
    printf("\n");

    return 1;
}
```

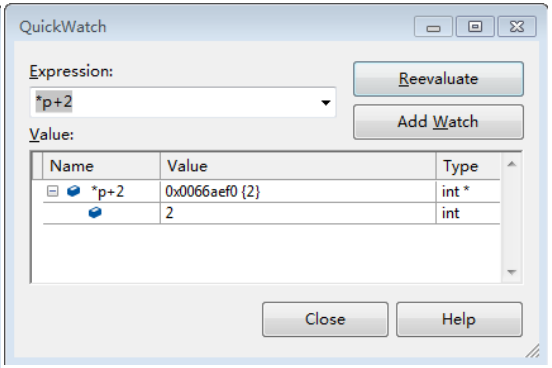










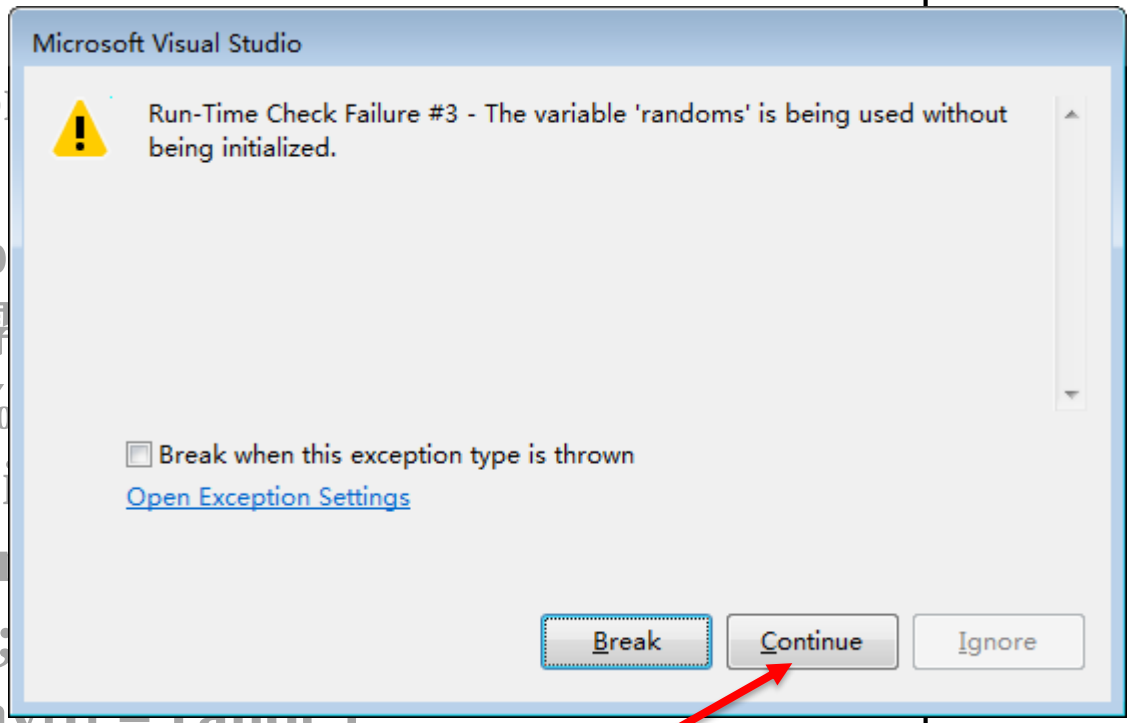



```
0 1 2
```

程序找错

```
int main()
{
        int  count,  *randoms;
        count  =  GetRandomArray(randoms);
        printf("最后的那个随机数是：  %d\n",
                randoms[count-1]);
}
int GetRandomArray(int array[])
{
        int  i,   count;
        printf("需要多少随机数? ");
        scanf("%d", &count);
        array = (int *) malloc(count * sizeof(int));
        srand((unsigned)time(NULL));
        for (i = 0; i < count; i++)
                array[i] = rand( );
        return count;
}
```

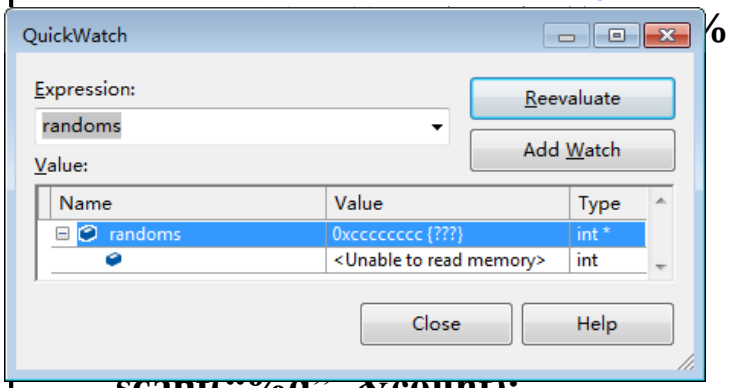
```
int main()
{
        int  count,  *randoms;
        count  =  GetRandomArray(randoms);
        printf("最后的那个随机数是：  %d\n",
                randoms[count-1]);
}
int GetRando
{
        int  i,   co
        printf("需
        scanf("%
        array = (
        srand((u
        for (i = 0;
                array[i] = rand();
        return count;
}
```

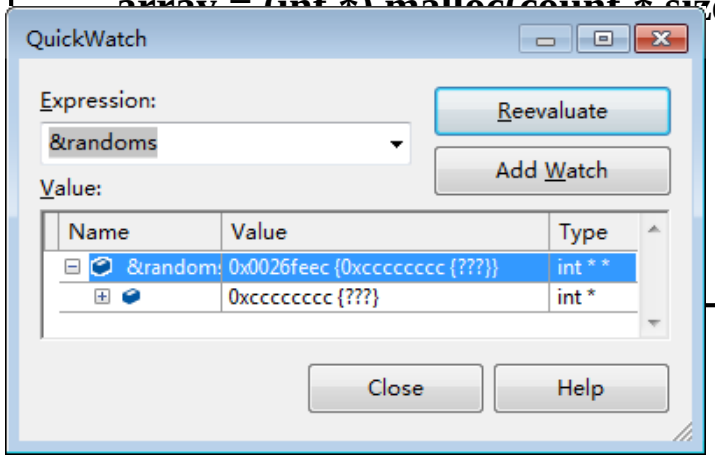


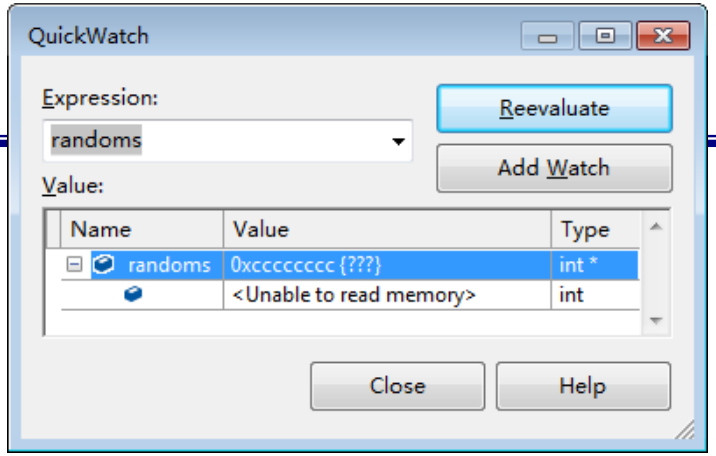
如果点击Continue？

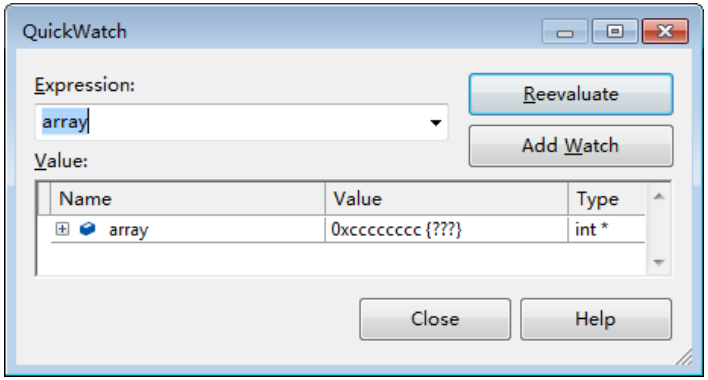
错误分析

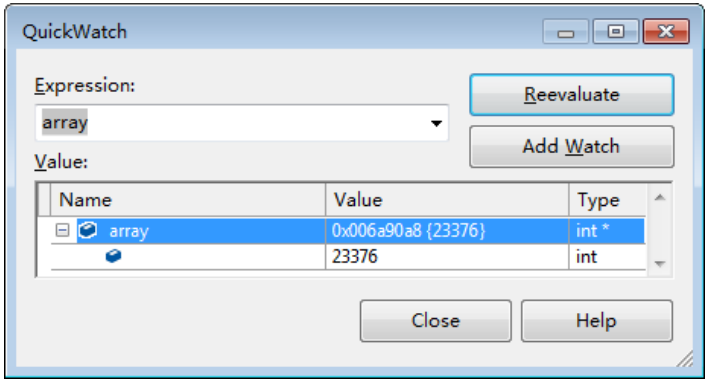
```
int main()
{
        int  count,  *randoms;
        count  =  GetRandomArray(randoms);
                                          %d\n",
        scanf("%d", &count);
        array = (int *) malloc(count * sizeof(int));
```



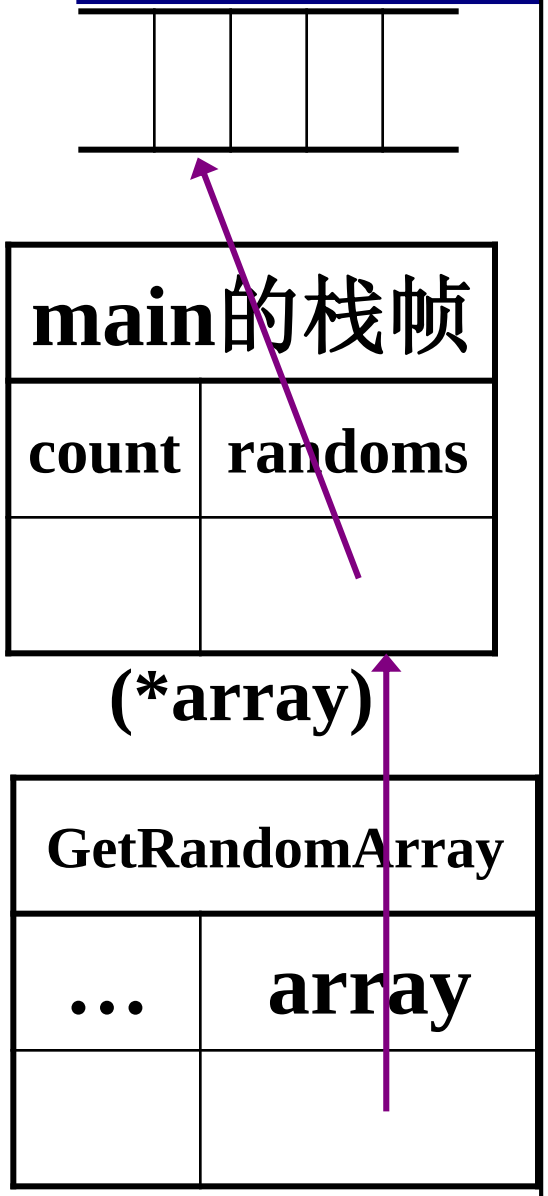








动态数组

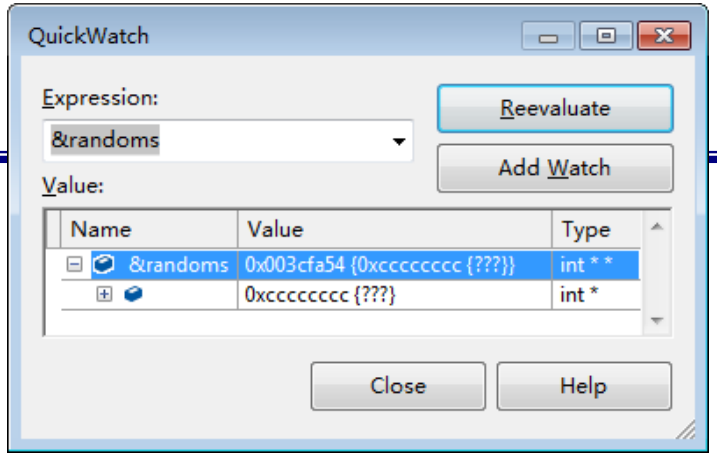


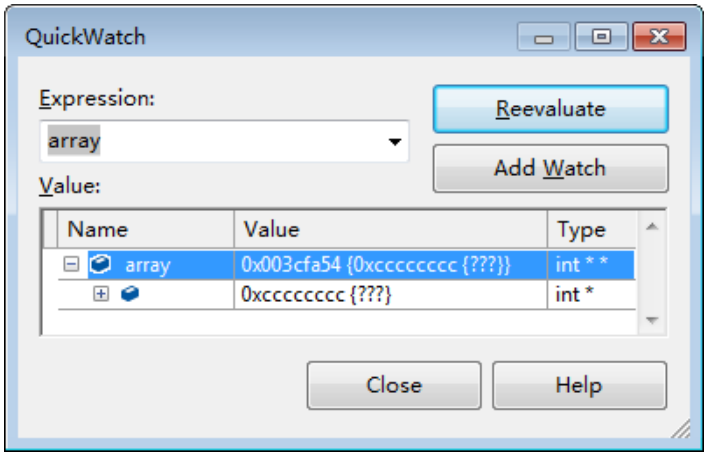

```
int  main()
{
        int  count,  *randoms;
        count  =  GetRandomArray(&randoms);
        printf("最后的那个随机数是：  %d\n",
                randoms[count-1]);
}
int GetRandomArray(int **array)
{
        int  i,   count;
        printf("需要多少随机数? ");
        scanf("%d", &count);
        *array = (int *) malloc(count * sizeof(int));
        srand((unsigned)time(NULL));
        for (i = 0; i < count; i++)
                (*array)[i] = rand( );
        return count;
}
```

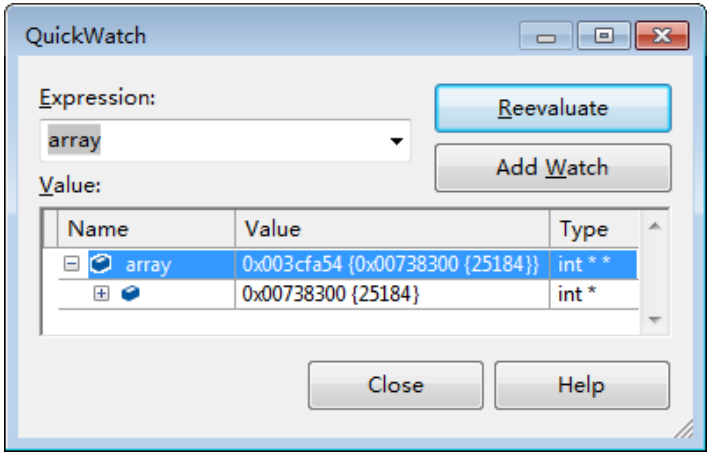
```
需要多少随机数? 5
最后的那个随机数是：10971
```

程序分析

```
int  main()
{
        int  count,  *randoms;
        count  =  GetRandomArray(&randoms);
        printf("最后的那个随机数是：  %d\n",
                randoms[count-1]);
}
int GetRandomArray(int **array)
{
        int  i,   count;
        printf("需要多少随机数? ");
        scanf("%d", &count);
        *array = (int *) malloc(count * sizeof(int));
        srand((unsigned)time(NULL));
        for (i = 0; i < count; i++)
                (*array)[i] = rand( );
        return count;
}
```

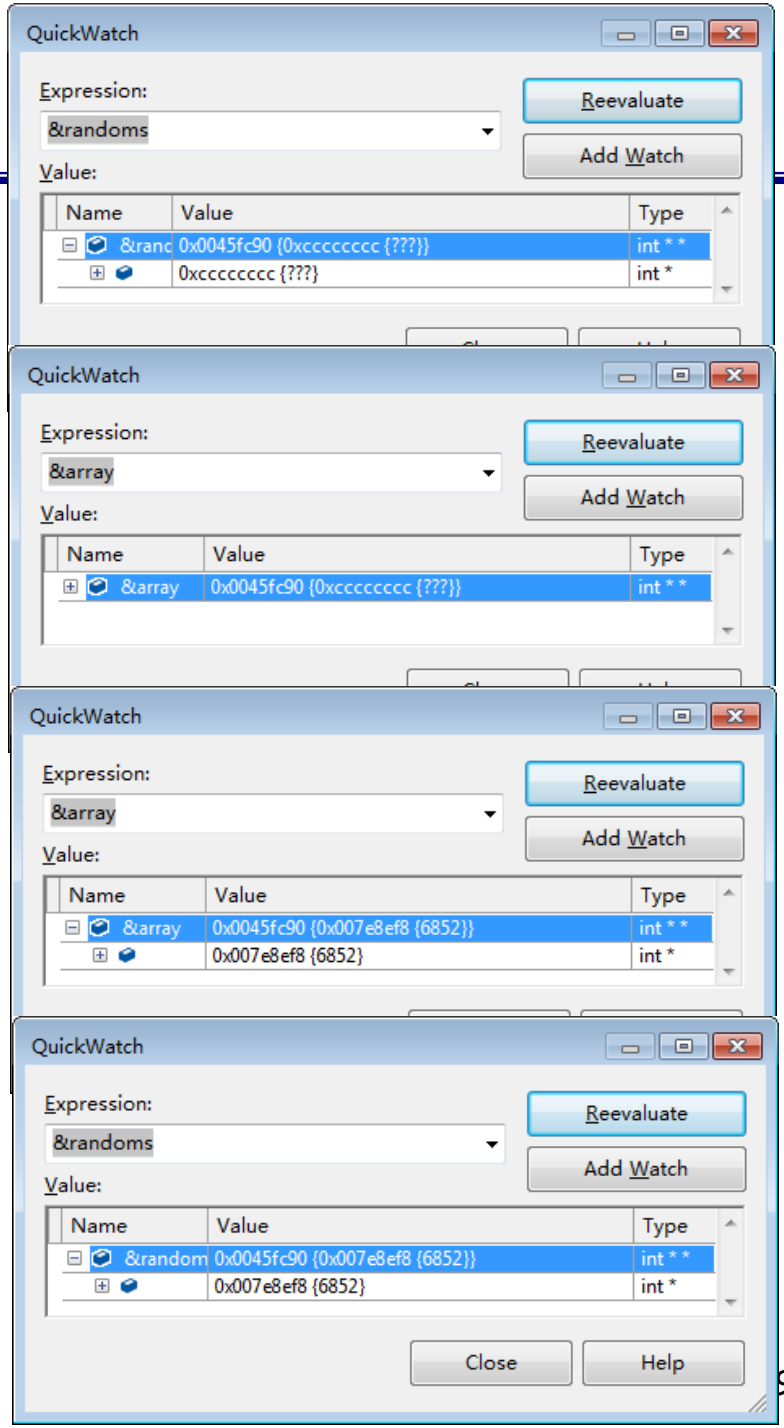






程序分析

```
int main()
{
        int  count,  *randoms;
        count  =  GetRandomArray(randoms);
        printf("最后的那个随机数是：  %d\n",
                randoms[count-1]);
}
int GetRandomArray(int *&array)
{
        int  i,   count;
        printf("需要多少随机数? ");
        scanf("%d", &count);
        array = (int *) malloc(count * sizeof(int));
        srand((unsigned)time(NULL));
        for (i = 0; i < count; i++)
                array[i] = rand( );
        return count;
}
```



函数之间的地址传递方法：

- 如果你是想把主函数中的地址，从上往下传，传到被调用的函数里面，那么形参和实参均为指针；
- 如果你是想把被调用函数中的地址，从下往上传，传到主函数当中去，那么形参和实参均为指向指针的指针。

二维动态数组的创建与删除-指针数组

```
#include <stdio.h>
#include <stdlib.h>
# define m   3
int main()
{
        int *p[m], n, i, j;
        scanf("%d", &n);

        for(i = 0; i < m; i++)
        {
                p[i] = (int *)malloc(n * sizeof(int));
                for(j = 0; j < n; j++)
                {
                        p[i][j] = i*m + j;
                        printf("%d ", p[i][j]);
                }
                printf("\n");
        }
        for(i = 0; i < m; i++)
                free(p[i]);

        return 0;
}
```

行数已知(3)
列数未知(?)

```
4
0 1 2 3
3 4 5 6
6 7 8 9
```

二维动态数组大小(行列)未知怎么办？
thinking…

二维动态数组的创建与删除-指针的指针

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int **p = NULL, m, n, i, j;
        scanf("%d %d", &m, &n);
        p = (int **)malloc(m * sizeof(int *));
        for(i = 0; i < m; i++)
        {
                p[i] = (int *)malloc(n * sizeof(int));
                for(j = 0; j < n; j++)
                {
                        p[i][j] = i*m + j;
                        printf("%d ", p[i][j]);
                }
                printf("\n");
        }
        for(i = 0; i < m; i++)
                free(p[i]);
        free(p);
        return 0;
}
```

```
3 4
0 1 2 3
3 4 5 6
6 7 8 9
```

C++ 实现： new & delete

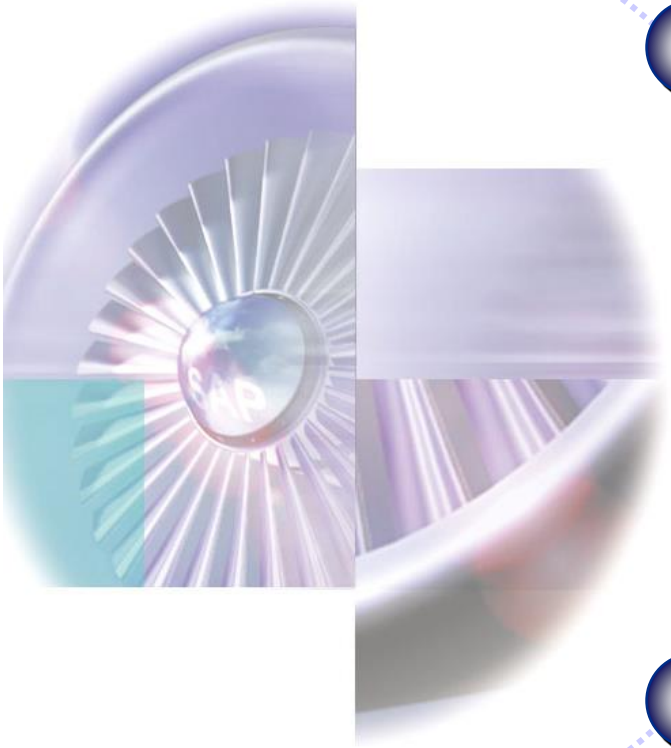
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int **p = NULL, m, n, i, j;
        scanf("%d %d", &m, &n);
        p = new int*[m];
        for(i = 0; i < m; i++)
        {
                p[i] = new int[n];
                for(j = 0; j < n; j++)
                {
                        p[i][j] = i*m + j;
                        printf("%d ", p[i][j]);
                }
                printf("\n");
        }
        for(i = 0; i < m; i++)
                delete [] p[i];
        delete [] p;
        return 0;
}
```

```
3 4
0 1 2 3
3 4 5 6
6 7 8 9
```

小结

- 创建过程：
  - 先动态创建一个指针数组 p (int **)；
  - 然后为指针数组的每个元素p[i] (int *)再动态指向一个数组的办法来完成的。
- 销毁过程：
  - 在销毁的过程，先销毁指针数组每个元素p[i]指向的数组；
  - 然后再销毁这个指针数组p 。

Lecture 7: 指针



1. 基本概念
2. 指针变量
3. 指针与数组
4. 指针与字符串

字符串的访问

- 对字符串中某个字符的访问方法：
  - 数组下标法：如 **str[i]**；
  - 指针法：如 ***(p + i)**，***p**；
- 对字符串的访问方法：
  - 数组名法：如 **printf(“%s”, str)**；
  - 指针法：如 **p = str**；**printf(“%s”, p)**；

程序找错

```
char  *GetMorning( )
{
        char str[] = "Morning";
        return str;
}
int main( )
{
        char msg[32];
        int i;
        for(i = 1; i <= 32; i++) msg[i] = '\0';   /* 将msg 清零 */
        msg = "Good";   /* 拷贝 “Good” 字符串 */
        msg[5] = ' ';    /* 添加一个空格 */
        strcat(msg, GetMorning());   /* 添加"Morning"字符串 */
        printf("%s",  msg);
}
```

Good Morning

调整的程序

```
char  *GetMorning( )
{
    char str[] = "Morning";
    return str;
}
int main( )
{
    char msg[32];
    int i;
    for(i = 0; i < 32; i++) msg[i] = '\0';   // 将msg 清零
    strcpy(msg, "Good");   // 拷贝 “Good” 字符串
    msg[4] = ' ';    // 添加一个空格
    strcat(msg, GetMorning());   // 添加"Morning"字符串
    printf("%s",  msg);
}
```

Good Morning

风险: 返回临时变量的地址!!

程序分析

```
void func(char *str)
{
    int len, i, j;
    char val[50], *p;
    len = strlen(str);
    i = 0;   j = len − 1;
    p = val;
    while((i < len / 2) && (j >= len / 2))
    {
        *p++ = str[j--];
        *p++ = str[i++];
     }
    *p = 0;
    strcpy(str, val);
}
```

```
int main( )
{
  char str[50] =
     "ogauain!sotltrnc";
   func(str);
   printf("%s", str);
}
```

输出结果是：

congratulations!

Arrays are constant pointers

```
int a[10];
int *pa;

pa=a;

pa++;
```

OK. 指针是变量，可以赋值和变化 (除了const 指针)

```
int a[10];
int *pa;

a=pa;

a++;
```

error C2440: '=' : cannot convert from 'int *' to 'int [10]'

数组名(a) 是常量，在内存中有固定的位置，不能用一个变量赋值常量

字符数组←→字符串指针

```
#include <stdio.h>

int main()
{
    char* a = "hello";
    char b[] = "world!";
    a = b;
    puts(a);
    puts(b);
    return 0;
}
```

```
world!
world!
```

```
#include <stdio.h>

int main()
{
    char* a = "hello";
    char b[] = "world!";
    b = a;
    puts(a);
    puts(b);
    return 0;
}
```

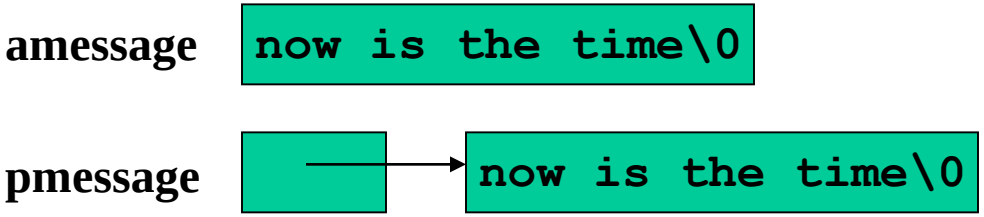
error C2440: '=' : cannot convert from 'char *' to 'char [7]'

```
#include <stdio.h>
#include <string.h>
int main()
{
    char* a = "hello";
    char b[] = "world!";
    strcpy(b, a);
    puts(a);
    puts(b);
    return 0;
}
```

```
hello
hello
```

字符数组 vs.字符串指针

```
char amessage[] = "now is the time"; /* an array */
char *pmessage = "now is the time"; /* a pointer */
```



程序举例

问题描述：

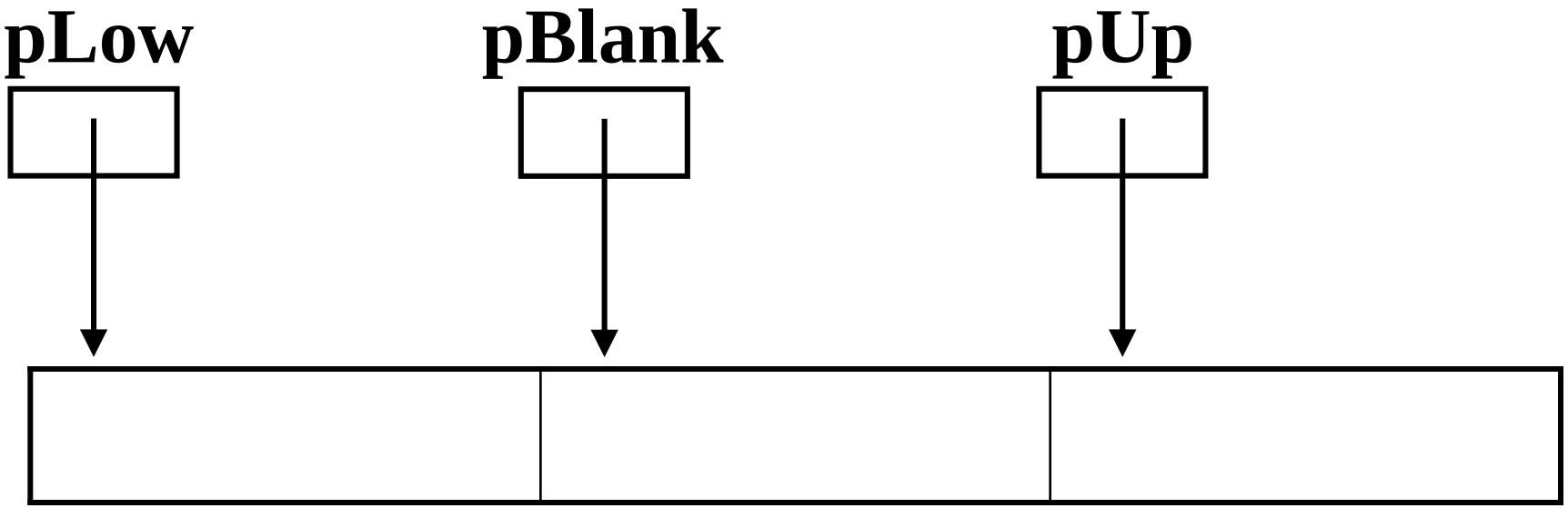
编写一个程序，输入一个字符串，该字符串只包含三种类型的字符：小写字母、大写字母和空格。然后生成一个新的字符串，把所有的小写字母放在最前面，所有的空格放在中间，所有的大写字母放在最后，而且这些小写、大写字母原来的顺序不能乱。最后把这个新的字符串输出。

输入字符串：**B_a_Ab**

ab__BA

讨论

1. 计算出各部分的长度；
2. 用三个指针分别指向其起始位置
3. 逐个字符的拷贝



```
int main()
{
    char src[100], dest[100];
    int  i, length;
    int up, low, blank;
    char *pUp, *pLow, *pBlank;

    up = 0;
    low = 0;
    blank = 0;
    gets(src);
    length = strlen(src);
    for(i = 0; i < length; i++)
    {
        if(src[i] >= 'a' && src[i] <= 'z')
            low++;
        else if(src[i] >= 'A' && src[i] <= 'Z')
            up++;
        else if(src[i] == ' ') blank++;
    }
```

B a Ab

ab BA

low = 2;

up = 2;

blank = 2;

```
    pLow = dest;
    pBlank = pLow + low;
    pUp = pBlank + blank;
    for(i = 0; i < length; i++)
    {
        if(src[i] >= 'a' && src[i] <= 'z')
            *pLow++ = src[i];
        else if(src[i] >= 'A' && src[i] <= 'Z')
            *pUp++ = src[i];
        else if(src[i] == ' ')
            *pBlank++ = src[i];
    }
    dest[length] = '\0';
    printf("%s\n", dest);
}
```

B a Ab

ab BA

a	b	_	_	B	A	\0	

Lecture 7 - Summary

- **Topics covered:**
  - **Pointers and Addresses**
  - **Pointers and Function Arguments**
  - **Pointers and Arrays**
  - **Pointer Arithmetics**
  - **Pointers and strings**
  - **Dynamic memory allocation**
  - **Pointer arrays. Pointers to pointers**
  - **Structures and pointers**