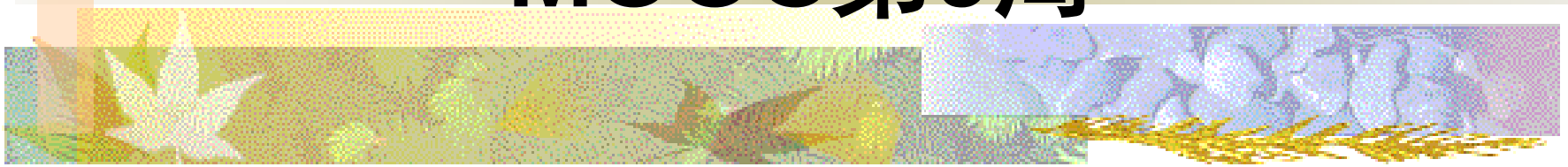
第9讲 指针和函数指针

教材9.1~9.6

MOOC第9周



哈尔滨工业大学

苏小红

sxh@hit.edu.cn

你想成为C语言世界中的齐天大圣吗？

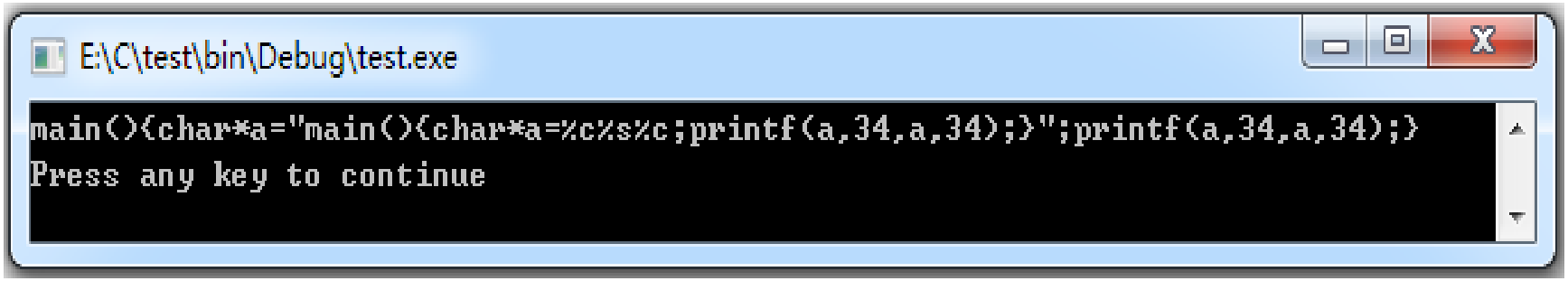
- 指针是C语言中的如意金箍棒
  - “稀饭（C Fans）”最挚爱的武器
- 强转与指针，并称C语言的两大神器

见证奇迹的时刻

- C的高效、高能主要来自于指针
  - 很多“Mission Impossible”由指针完成
  - C语言是一切皆有可能（Impossible is Nothing）

源程序（不要添加任何空格和换行）

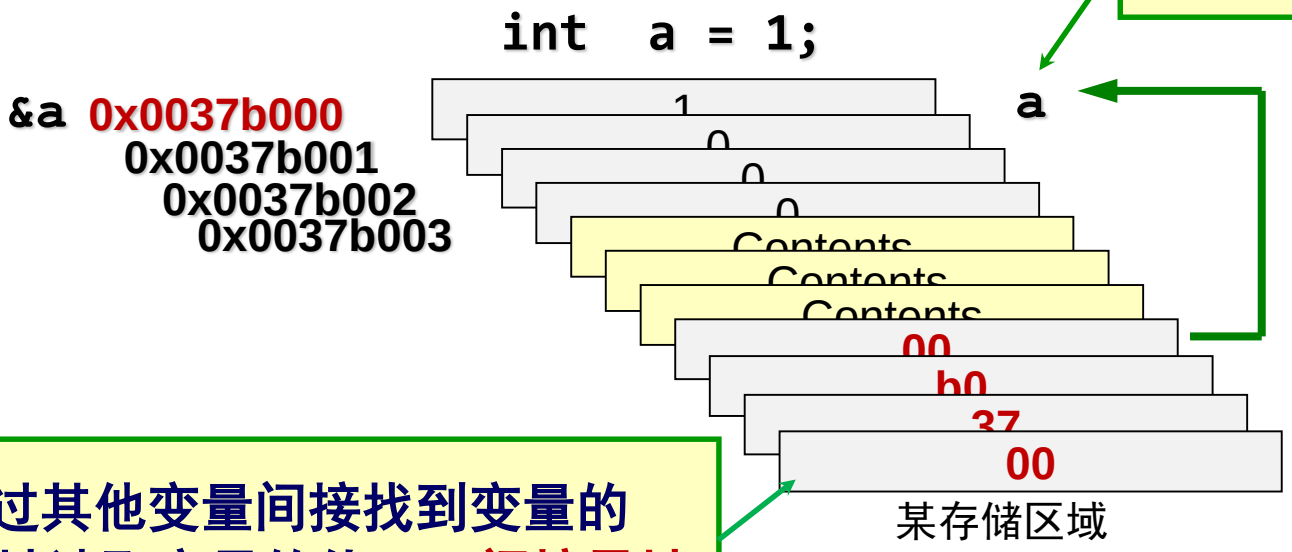
```
main(){char*a="main(){char*a=%c%s%c;printf(a,34,a,34);}"
;printf(a,34,a,34);}
```



如何对变量进行寻址？

```
printf("&a=%p\n", &a);
```

直接到变量名标识的存储单元中读取变量的值——直接寻址



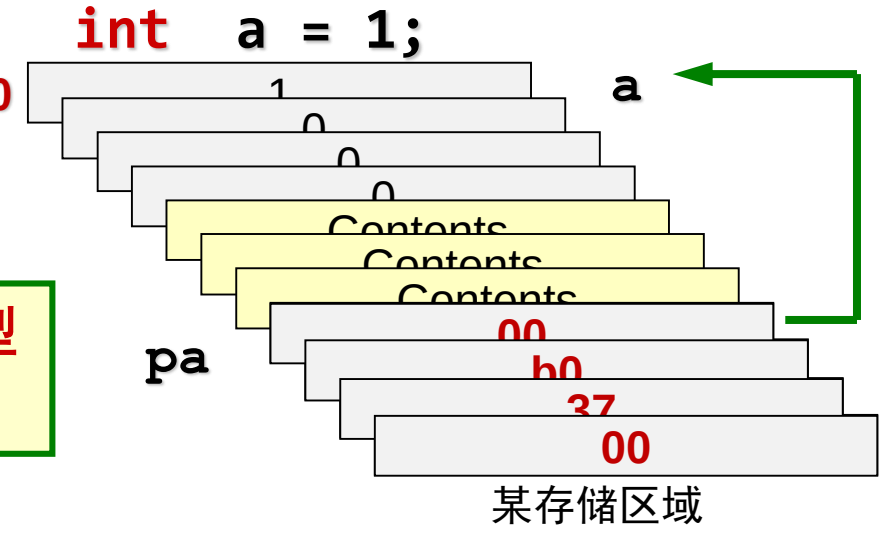
通过其他变量间接找到变量的地址读取变量的值——间接寻址

用什么类型的变量存放地址?

- 指针（Pointer）类型的变量—指针变量
  - int *pa;　　//pa可以指向一个int型数据，但指向哪里不确定

```
pa = &a;
int *pa = &a;
```

指针变量指向的数据类型，称为指针的基类型
指针变量只能指向同一基类型的变量



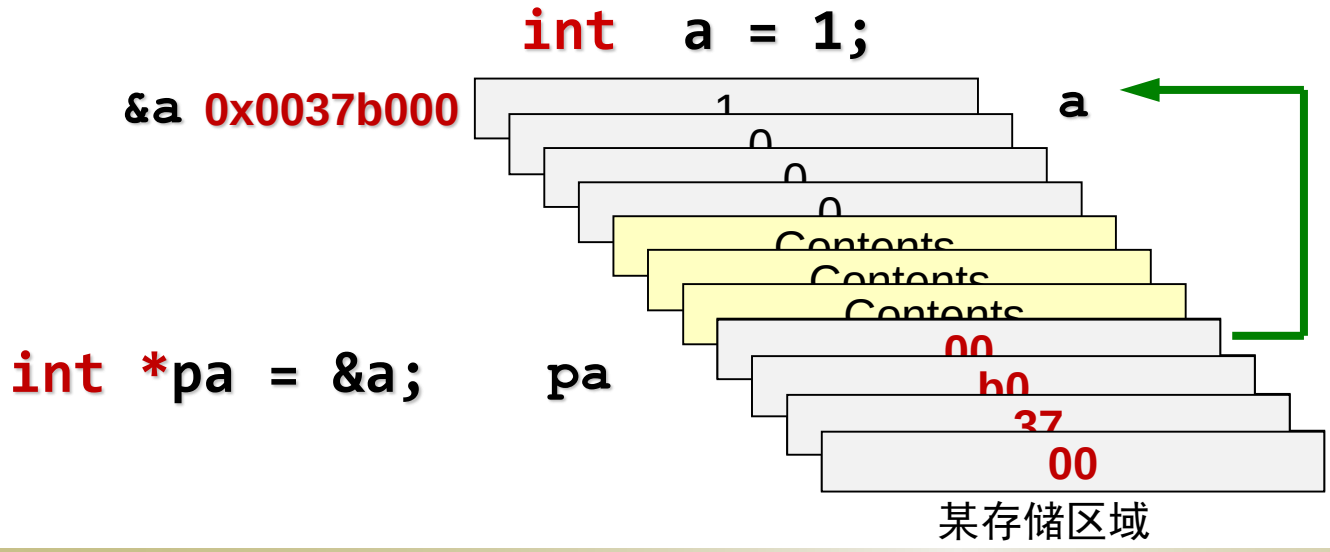
正因为指针可以保存一个地址，使得程序员直接访问内存成为可能

在32位机上保存32位地址值需要4个字节（无论指向什么类型）

指针占内存空间的字节数与指针的基类型无关

为什么要指定指针变量的基类型?

- 假如你是编译器
  - 你知道这4个字节中保存了一个地址，就够了吗?
  - 知道地址，就能正确解析指针吗?
  - 你会用什么类型去理解（访问）指针指向的内存数据?
  - 你知道从这个地址开始的多少个字节是有效数据吗?
  - 指针加1应该加几个字节呢?

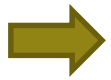


不指定指针指向哪里会怎样？

- 指针变量使用之前**必须初始化**
- 若你不知把它指向哪里，那就指向**NULL**

```
warning:'pa' is used uninitialized
```

```
int *pa;
```



```
int *pa = NULL;
```

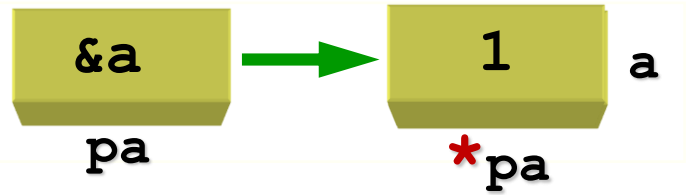
NULL是什么？

- 空指针——无效指针
  - 在stdio.h中定义为0
- 空指针就是指向地址为0的存储单元的指针吗？
  - 不一定
  - 并非所有编译器都使用0地址
  - 某些编译器使用不存在的内存地址
- p = 0 和 p = NULL 有什么区别？
  - p = NULL，明确说明p是指针变量

如何访问指针变量指向的数据？

- 通过间接寻址运算符访问（引用）指针变量指向的变量的值
- ——指针的解引用(Pointer Dereference)

```
#include <stdio.h>
int main()
{
   int  a = 0;
   int  *pa = &a;        此*非彼*
   *pa = 1;
   printf("a=%d\n", a);
   printf("*p=%d\n", *pa);
   return 0;
}
```



- *pa是a的别名
- 解引用空指针→程序崩溃

单选题

设置

下列说法错误的是（）。

A 指针变量指向的数据的类型，称为指针的基类型。指针变量只能指向同一基类型的变量。

B 指针变量占用的内存单元字节数就是它所指向的变量所占用的内存单元字节数。

C 指针变量使用之前必须初始化，使用未初始化的指针的结果将导致非法内存访问错误。

D 通过间接寻址运算符引用指针变量指向的变量的值，称为指针的解引用。

提交

你曾经写过的交换法排序

```
void ExchangeSort(int x[], int n)
{
   int i, j, temp;
   for (i=0; i<n-1; i++)
   {
        for (j=i+1; j<n; j++)
        {
           if (x[j] < x[i])//升序
           {
                temp = x[j];
                x[j] = x[i];
                x[i] = temp;
           }
        }
   }
}
```

```
void ExchangeSort(int x[], int n)
{
   int i, j, temp;
   for (i=0; i<n-1; i++)
   {
        for (j=i+1; j<n; j++)
        {
           if (x[j] < x[i])//升序
           {
              Swap(&x[j], &x[i]);
           }
        }
   }
}
```

你想修改谁，就传谁的地址

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

指针形参接收你想修改的变量的地址

Errors

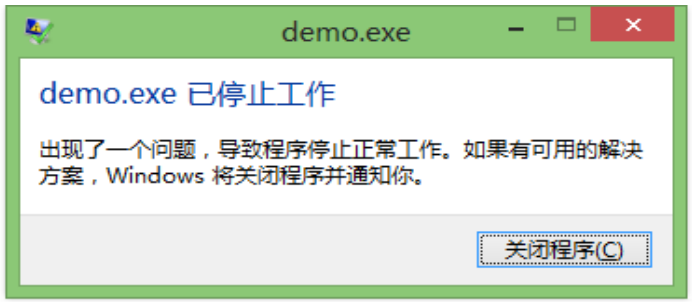
```
void SelectionSort(int x[],int n)
{
    int temp, i, k;
    for (i=0; i<n-1; i++)
    {
       k = FindMin(x, i, n);
       if (k != i)
       {
          Swap(x[k], x[i]);
       }
    }
}
```

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

某些编译器检查实参和形参的类型是否匹配，不匹配则给出警告

另一些编译器直接将实参的值当作地址值，产生非法内存访问，导致程序异常终止

```
scanf("%d", a);
```



单选题

这段代码的功能是（）

```
void Swap(int *x, int *y)
{
    int *pTemp;

    pTemp = x;
    x = y;
    y = pTemp;
}
```

A 交换指针x和y指向的内容

B 交换指针变量x和y的指向

C 什么都不能交换

D 指针变量pTemp未初始化，将导致程序崩溃

提交

设置

单选题

这段代码的功能是（）

```
void Swap(int *x, int *y)
{
    int *pTemp;

    *pTemp = *x;
    *x = *y;
    *y = *pTemp;
}
```

A 交换指针x和y指向的内容

B 交换指针变量x和y的指向

C 什么都不能交换

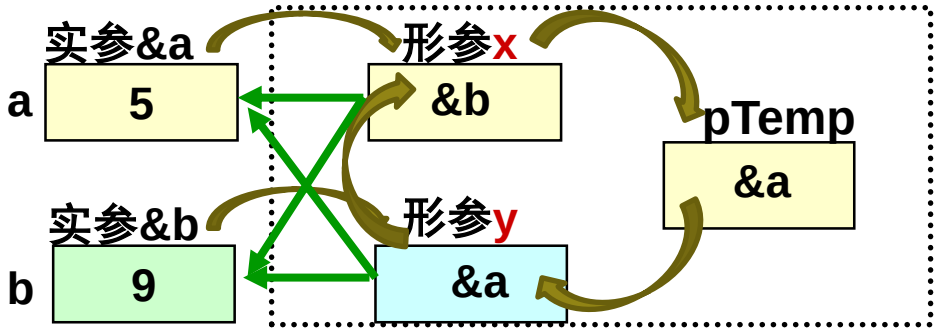
D 指针变量pTemp未初始化，将导致程序崩溃

提交

设置

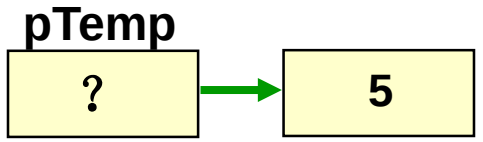
Errors

```
void Swap(int *x, int *y)
{
    int *pTemp;

    pTemp = x;
    x = y;
    y = pTemp;
}
```



交换的是地址值(即x和y的指向)
不是指针指向的内容

```
void Swap(int *x, int *y)
{
    int *pTemp;

    *pTemp = *x;
    *x = *y;
    *y = *pTemp;
}
```



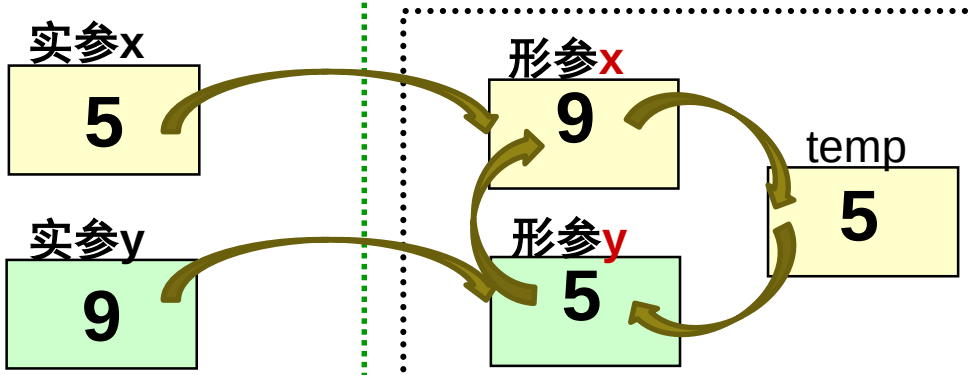
原因：指针变量未初始化
结论：不能借助一个未初始化的指针变量进行两数互换

Errors

```
int main()
{
  int x = 5, y = 9;
  Swap(x, y);
  printf("x=%d,y=%d", x, y);
  return 0;
}
```

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

Call by value



为什么形参和实参同名可以互不干扰？

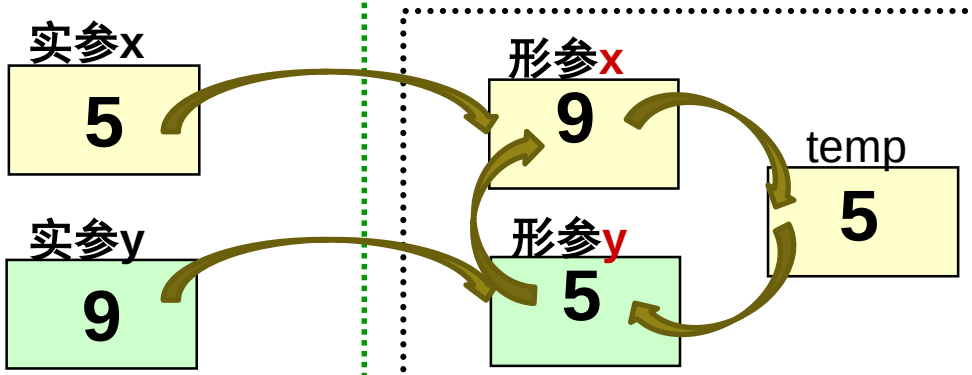
并列语句块内，作用域不同，所以形参值的改变不会影响实参

假如局部变量与全局变量同名？

```
int x = 5, y = 9;//全局变量
int main()
{
  Swap(x, y);
  printf("x=%d,y=%d", x, y);
  return 0;
}
```

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

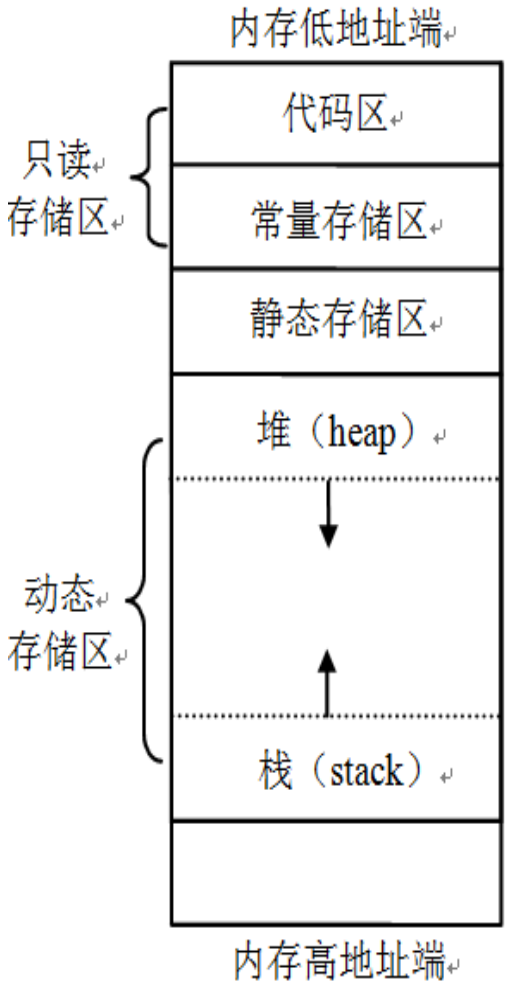
Call by value



- 作用域小的隐藏作用域大的

编译器如何区分不同作用域的同名变量？

* 通过将同名变量映射到不同的内存地址，来实现作用域的划分
  * 局部变量和全局变量可以同名
    * 被分配的内存区域不同
  * 形参和实参可以同名
    * 被分配的内存区域，但地址不同



n*n矩阵的转置

- 用二维数组作为函数参数，编程计算并输出n×n阶矩阵的转置矩阵。其中，n的值不超过10，n的值由用户从键盘输入。

```
#include <stdio.h>
#define N 10
void Transpose(int a[][N], int n);
void InputMatrix(int a[][N], int n);
void PrintMatrix(int a[][N], int n);
void Swap(int *x, int *y);
int main()
{
    int s[N][N], n;
    printf("Input n:");
    scanf("%d", &n);
    InputMatrix(s, n);
    Transpose(S, n);  //数组名做实参，传数组的首地址
    printf("The transposed matrix is:\n");
    PrintMatrix(s, n);
    return 0;
}
```

形参声明为二维数组时，第1维长度可省略，第2维长度不能省略，且必须是常量

```
// 计算n*n矩阵的转置矩阵
void Transpose(int a[][N], int n)
{
    int i, j;
    for (i=0; i<n; i++)
    {
        for (j=i+1; j<n; j++)
        {
            Swap(&a[i][j], &a[j][i]);
        }
    }
}
```

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

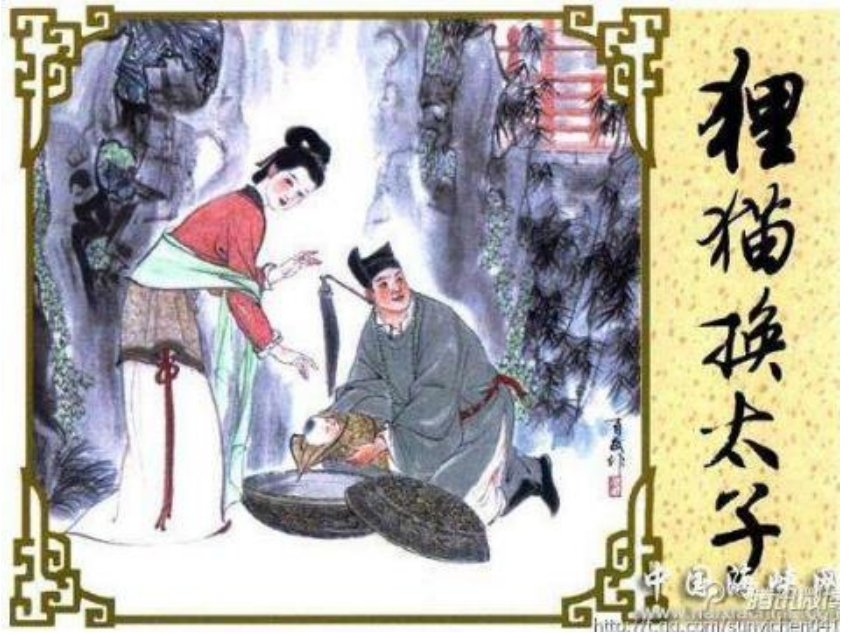
```
// 函数功能：输入n*n矩阵的值
void InputMatrix(int a[][N], int n)
{
    int i, j;
    printf("Input %d*%d matrix:\n", n, n);
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
}
// 函数功能：输出n*n矩阵的值
void PrintMatrix(int a[][N], int n)
{
    int i, j;
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
}
```

```
int main()
{
    int dog = 1, rabbit = 2, cat = 3, monkey = 4, deer = 5, flag;
    do{
        flag = 0;
        if (monkey > cat)
        {
            Swap(&monkey, &cat);
            flag++; //flag = 1;
        }
        if (dog < deer)
        {
            Swap(&dog, &deer);
            flag++; //flag = 1;
        }
        if (rabbit > monkey)
        {
            Swap(&rabbit, &monkey);
            flag++; //flag = 1;
        }
        if (rabbit < dog)
        {
            Swap(&rabbit, &dog);
            flag++; //flag = 1;
        }
    } while (flag);
    printf("%d,%d,%d,%d,%d\n", dog, rabbit, cat, monkey, deer);
    return 0;
}
```

利用swap()解决动物百米赛跑

狸猫换太子

- 一个关于发生在北宋年间变量替换的故事…
  - 真宗和李玉未出生的孩子：小太子
  - 刘娥的阴谋：用狸猫替换掉孩子
  - 筹备工作：设计两套替换行动方案



```c
#include <stdio.h>
#define CIVET 0  //定义狸猫值为0
#define PRINCE 1 //定义王子值为1
void Replace1(int baby)
{
    baby = CIVET;
}
void Replace2(int *baby)
{
    *baby = CIVET;
}
void Display(int who)
{
    if (who == CIVET)
        printf("狸猫");
    else if (who == PRINCE)
        printf("王子");
}
```

```c
int main()
{
    int baby = PRINCE;//baby代表孩子，刚出生时是王子

    printf("before change, baby is ");
    Display(baby);

    Replace1(baby);      //实施狸猫换太子第一次行动
    printf("\n");
    printf("after first action, baby is ");
    Display(baby);

    Replace2(&baby);     //实施狸猫换太子第二次行动
    printf("\n");
    printf("after second action, baby is ");
    Display(baby);
    return 0;
}
```

刘娥的两套行动方案的实施结果如何？

单选题 1分

设置

刘娥的两套行动方案的实施结果如何？

A 第1套失败，第2套成功

B 第1套成功，第2套失败

C 两套都成功

D 两套都失败

提交

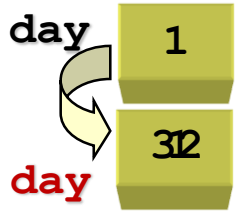
指针变量作函数形参，究竟有什么用？

```
//函数功能：对给定的某年某月某日，计算并返回它是这一年的第几天
void DayofYear(int year, int month, int day)
{
   int dayTab[2][13] = {{0,31,28,31,30,31,30,31,31,30,31,30,31},
                        {0,31,29,31,30,31,30,31,31,30,31,30,31}};
   int  i, leap;
   leap = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
   for (i=1; i<month; i++)
   {
       day = day + dayTab[leap][i];
   }
                  //若year为闰年，即leap值为1，则用第1行元素dayTab[1][i]计算；
}                 //否则leap值为0，用第0行dayTab[0][i]计算
```

DayofYear(year, month, day);

形参day值的改变能返回给实参day吗？

Call by value：基本类型的变量作形参



指针变量作函数形参，究竟有什么用？

```
//函数功能：对给定的某年某月某日，计算并返回它是这一年的第几天
int  DayofYear(int year, int month, int day)
{
   int dayTab[2][13] = {{0,31,28,31,30,31,30,31,31,30,31,30,31},
                        {0,31,29,31,30,31,30,31,31,30,31,30,31}};
   int  i, leap;
   leap = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
   for (i=1; i<month; i++)
   {
       day = day + dayTab[leap][i];
   }
   return day;  //return仅限于从函数返回一个值
}
```

```
DayofYear(year, month, day);
```

Call by value：基本类型的变量作形参

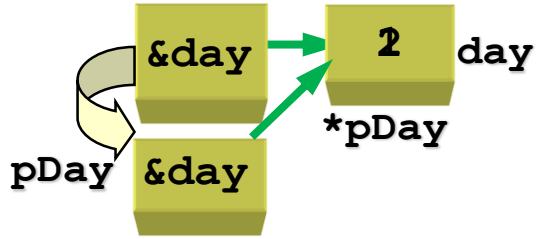
形参day值的改变不能返回给实参day

指针变量作函数形参，究竟有什么用？

```
//函数功能：对给定的某年某月某日，计算并返回它是这一年的第几天
void DayofYear(int year, int month, int *pDay)
{
   int dayTab[2][13] = {{0,31,28,31,30,31,30,31,31,30,31,30,31},
                        {0,31,29,31,30,31,30,31,31,30,31,30,31}};
   int  i, leap;
   leap = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
   for (i=1; i<month; i++)
   {
        *pDay = *pDay + dayTab[leap][i];
   }
}
```

指针形参接收你想修改的变量的地址

```
DayofYear(year, month, &day);
```



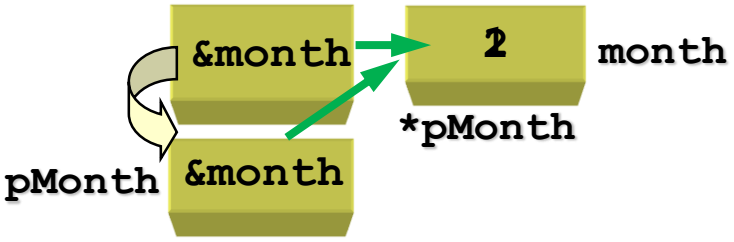
Simulating Call by reference：指针变量作形参

你想修改哪个变量，就传哪个变量的地址

指针变量作函数形参，究竟有什么用？

```
// 函数功能：对给定的某一年的第几天，计算并返回它是这一年的第几月第几日
void  MonthDay(int year, int yearDay, int *pMonth, int *pDay)
{
   int dayTab[2][13] = {{0,31,28,31,30,31,30,31,31,30,31,30,31},
                        {0,31,29,31,30,31,30,31,31,30,31,30,31}};
   int  i, leap;
   leap = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
   for (i=1; yearDay > dayTab[leap][i]; i++) //减到不够减了为止
   {
       yearDay = yearDay - dayTab[leap][i];
   }
   *pMonth = i;      //月份
   *pDay = yearDay; //日
}
```

指针变量作形参（出口参数）
为函数提供了修改实参值的手段



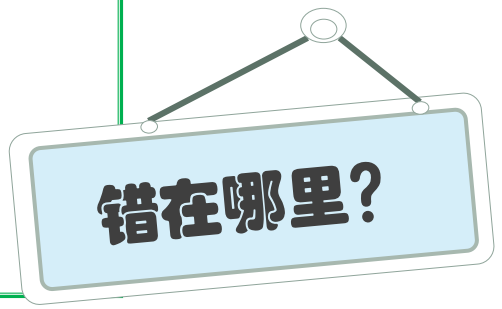

```
MonthDay(year, yearDay, &month, &day);
```

传你想修改的实参变量的地址

计算最小值及其下标位置

```
#include <stdio.h>
#define N 10
int FindMinPos(int a[], int m);
int main()
{
    int a[N], i, n;
    int minValue, minPos;
    printf("Input n:");
    scanf("%d", &n);
    printf("Input %d numbers:", n);
    for (i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
    minPos = FindMinPos(a, n); //不带下标的数组名做实参
    minValue = a[minPos];      //根据最小值下标计算最小值
    printf("min=%d,pos=%d\n", minValue, minPos);
    return 0;
}
```

```
int FindMinPos(int a[], int n)
{
    int  i, min, minPos;
    min = a[0];

    for (i=1; i<n; i++)
    {
        if (a[i] < min)
        {
            min = a[i];
            minPos = i;
        }
    }
    return minPos;//返回最小值下标位置
}
```



计算最小值及其下标位置

```c
#include <stdio.h>
#define N 10
int FindMinPos(int a[], int n);
int main()
{
    int a[N], i, n, minPos;
    scanf("%d", &n);
    printf("Input %d numbers:", n);
    for (i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
    minPos = FindMinPos(a, n);
    minValue = a[minPos];
    printf("min=%d,pos=%d\n",minValue,minPos);
    return 0;
}
```

```c
#include <stdio.h>
#define N 10
int FindMin(int a[], int n, int *pMinPos);
int main()
{
    ……
    minValue = FindMin(a, n, &minPos);
    printf("min=%d,pos=%d\n",minValue,minPos);
    return 0;
}
```

```c
int FindMinPos(int a[], int n)
{
   int  i, min, minPos;
   min = a[0];
   minPos = 0;//初始化为假设最小值a[0]的下标
   for (i=1; i<n; n++)
   {
      if (a[i] < min)
      {
          min = a[i];
          minPos = i; //记录最小值下标位置
      }
   }
   return minPos; //返回最小值下标位置
}
```

```c
int FindMin(int a[], int n, int *pMinPos)
{
   int i, min;
   min = a[0];
   *pMinPos = 0;//初始化为a[0]的下标
   for (i=1; i<n; i++)
   {
      if (a[i] < min)
      {
         min = a[i];
         *pMinPos = i;//记录最小值下标位置
      }
   }
   return min;//返回最小值
}
```

指针可以指向代码吗？

```
#include <stdio.h>
void Fun(int x, int y, int (*f)(int,
int));
int Max(int x, int y);
int Min(int x, int y);
int Add(int x, int y);
int main()
{    int a, b;
     scanf("%d,%d", &a, &b);
     Fun(a, b, Max);
     Fun(a, b, Min);
     Fun(a, b, Add);
     return 0;
}
void Fun(int x,int y, int (*f)(int,int))
{    int result;
     result = (*f)(x, y) ;
     printf("%d\n", result);
}
```

```
5,9↙
max=9
min=5
sum=1
```

```
int Max(int x, int y)
{
     printf("max=");
     return x>y ? x : y;
}
```

```
int Min(int x, int y)
{
     printf("min=");
     return x<y ? x : y;
}
```

```
int Add(int x, int y)
{
     printf("sum=");
     return  x + y;
}
```

指向函数的指针变量——函数指针

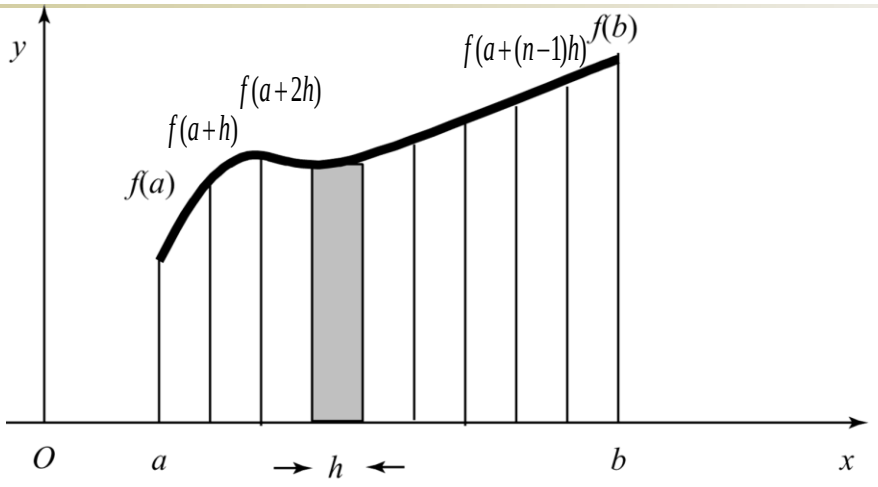
编译器将不带()的函数名解释为该函数在内存中的入口地址

梯形法计算函数的定积分

- 如果不用**函数指针**编程…

$$y_1 = \int_0^1 (1+x^2)\mathrm{d}x$$

$$y_2 = \int_0^3 \frac{x}{1+x^2}\mathrm{d}x$$



n为区间等分数 $h = \frac{b-a}{n}$

$$y = \frac{h}{2}(f(a)+f(a+h)) + \frac{h}{2}(f(a+h)+f(a+2h)) + \ldots + \frac{h}{2}(f(a+(n-1)h)+f(b))$$

$$= \frac{h}{2}(f(a)+2f(a+h)+2f(a+2h)+\ldots+2f(a+(n-1)h)+f(b))$$

$$= h(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+i\cdot h))$$

$$y_1 = \int_0^1 (1+x^2)dx \qquad y = h(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+i\cdot h)) \qquad h = \frac{b-a}{n} \qquad y_2 = \int_0^3 \frac{x}{1+x^2}dx$$

```
y1 = IntegralF1(0.0, 1.0);
```

```
y1 = IntegralF2(0.0, 3.0);
```

```
float IntegralF1(float a,float b)
{
    float s, h;
    int n = 100, i;
    s = (F1(a) + F1(b)) / 2;
    h = (b - a) / n;
    for (i=1; i<n; i++)
    {
        s =  s + F1(a + i * h);
    }
    return s * h;
}
```

```
float IntegralF2(float a,float b)
{
    float s, h;
    int n = 100, i;
    s = (F2(a) + F2(b)) / 2;
    h = (b - a) / n;
    for (i=1; i<n; i++)
    {
        s =  s + F2(a + i * h);
    }
    return s * h;
}
```

```
float F1(float x)
{
    return 1+x*x;
}
```

$$f_1(x) = 1+x^2$$

```
float F2(float x)
{
    return x/(1+x*x);
}
```

$$f_2(x) = \frac{x}{1+x^2}$$

$$y = \int_a^b f(x)\mathrm{d}x$$

```
y1 = IntegralF1(0.0, 1.0);
```

```
float IntegralF1(float a,float b)
{
    float s, h;
    int n = 100, i;
    s = (F1(a) + F1(b)) / 2;
    h = (b - a) / n;
    for (i=1; i<n; i++)
    {
        s =  s + F1(a + i * h);
    }
    return s * h;
}
```

```
float F1(float x)
{
    return 1+x*x;
}
```

$$f_1(x) = 1 + x^2$$

```
y1 = Integral(F1, 0.0, 1.0);
y2 = Integral(F2, 0.0, 3.0);
```

```
float Integral(float (*f)(float),
                    float a, float b)
{
    float s, h;
    int n = 100, i;
    s = ((*f)(a) + (*f)(b)) / 2;
    h = (b - a) / n;
    for (i=1; i<n; i++)
    {
        s = s + (*f)(a + i * h);
    }
    return s * h;
}
```

```
float F2(float x)
{
    return x/(1+x*x);
}
```

$$f_2(x) = \frac{x}{1+x^2}$$

还有什么时候需要指针指向代码？

```
//选择法升序排序
void AscendingSort(int a[], int n)
{
   int i, j, k;
   for (i=0; i<n-1; i++)
   {
      k = i;
      for (j=i+1; j<n; j++)
      {
        if (a[j] < a[k])  k = j;
      }
      if (k != i)
      {
         Swap(&a[k], &a[i]);
      }
   }
}
```

```
//选择法降序排序
void DescendingSort(int a[], int n)
{
   int i, j, k;
   for (i=0; i<n-1; i++)
   {
      k = i;
      for (j=i+1; j<n; j++)
      {
         if (a[j] > a[k])  k = j;
      }
      if (k != i)
      {
          Swap(&a[k], &a[i]);
      }
   }
}
```

使用函数指针编写一个通用的排序函数

```
//选择法升序排序
void AscendingSort(int a[], int n)
{
   int i, j, k;
   for (i=0; i<n-1; i++)
   {
      k = i;
      for (j=i+1; j<n; j++)
      {
        if (a[j] < a[k])  k = j;
      }
      if (k != i)
      {
         Swap(&a[k], &a[i]);
      }
   }
}
```

```
//选择法降序排序
void DescendingSort(int a[], int n)
{
   int i, j, k;
   for (i=0; i<n-1; i++)
   {
      k = i;
      for (j=i+1; j<n; j++)
      {
         if (a[j] > a[k])  k = j;
      }
      if (k != i)
      {
          Swap(&a[k], &a[i]);
      }
   }
}
```

```
SelectionSort(a, n, Ascending);
SelectionSort(a, n, Descending);
```

```
//通用的选择法排序函数
void SelectionSort(int a[], int n,
                   int (*compare)(int,int))
{
   int i, j, k;
   for (i=0; i<n-1; i++)
   {
      k = i;
      for (j=i+1; j<n; j++)
      {
         if ((*compare)(a[j], a[k])) k = j;
      }
      if (k != i)
      {
          Swap(&a[k], &a[i]);
      }
   }
}
```

```
//选择法升序排序
void AscendingSort(int a[], int n)
{
    int i, j, k;
    for (i=0; i<n-1; i++)
    {
        k = i;
        for (j=i+1; j<n; j++)
        {
          if (a[j] < a[k])  k = j;
        }
        if (k != i)
        {
            Swap(&a[k], &a[i]);
        }
    }
}
```

```
//选择法降序排序
void DescendingSort(int a[], int n)
{
    int i, j, k;
    for (i=0; i<n-1; i++)
    {
        k = i;
        for (j=i+1; j<n; j++)
        {
           if (a[j] > a[k])  k = j;
        }
        if (k != i)
        {
            Swap(&a[k], &a[i]);
        }
    }
}
```

■ 使用函数指针编写一个通用的排序函数

```
SelectionSort(a, n, Ascending);
SelectionSort(a, n, Descending);
```

```
//通用的选择法排序函数
void SelectionSort(int a[], int n,
                   int (*compare)(int,int))
{
    i
    f
    {
        if ((*compare)(a[j], a[k])) k = j;
    }
    if (k != i)
    {
    }
}
```

```
if (Ascending(a[j], a[k])) k = j;
int Ascending(int a, int b)
{
    return a < b ? 1 : 0;//为真则升序
}
```

```
if (Descending(a[j], a[k])) k = j;
int Descending(int a, int b)
{
    return a > b ? 1 : 0;//为真则降序
}
```

多选题 1分

设置

下列说法正确的是（）

A 指针是一种特殊的可以保存地址值的数据类型

B 指针就是地址，因为指针变量的值是一个地址

C 指针保存了哪个存储单元的地址，就表示指针指向了哪个存储单元

D 指针既可以保存变量的起始地址，也可保存函数的入口地址

E 被调函数可以根据指针形参接收的地址访问它不能直接访问的变量的值，因此形参指针指向的变量的值有可能被被调函数所修改

F 函数指针做形参，使得被调函数可以根据传入的不同地址调用不同的函数

提交

小结

- 正确理解指针的概念
  - 指针是一种特殊的可以保存地址值的数据类型
  - 指针类型的变量，称为指针变量
  - 指针不是地址，指针变量的值是一个地址
  - 想让指针变量指向哪个存储单元，就让其保存哪个单元的地址
    - 保存一个变量的地址
    - 保存一个数组的首地址
    - 保存一个字符串的首地址
    - 保存一个函数的入口地址
  - 永远清楚指针指向了哪里