

规格严格 功夫到家



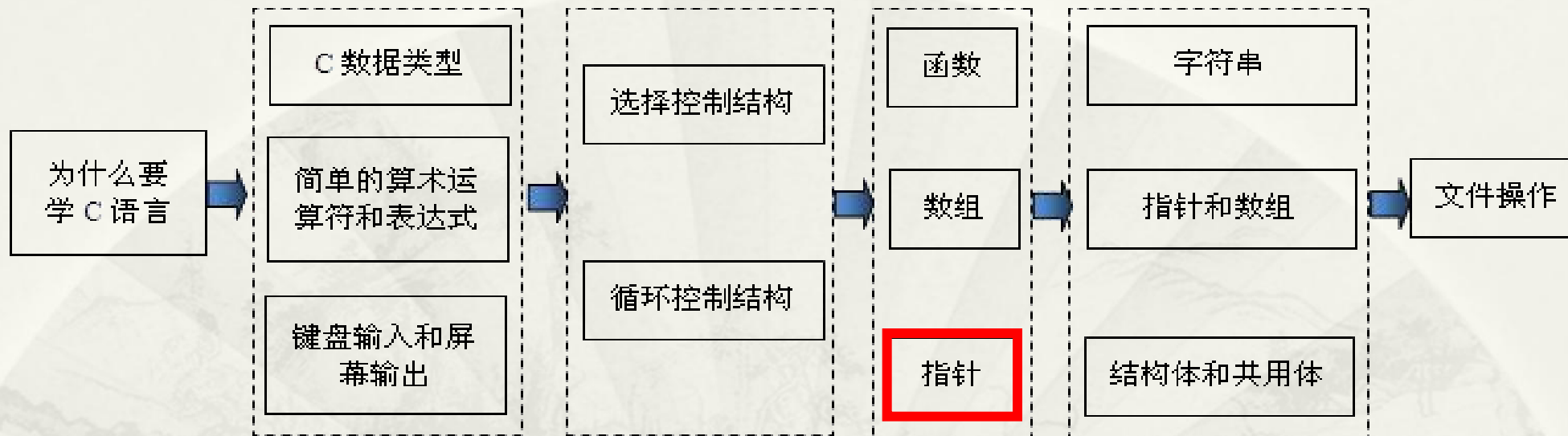
第9章 指针

哈尔滨工业大学（深圳）
计算机科学与技术学院
刘洋

Liu.yang@hit.edu.cn

课件.版权：哈尔滨工业大学，苏小红，sxh@hit.edu.cn





简单的数据结构





复杂的数据结构

第9章 学习内容

- 指针的概念和变量的地址
- 指针变量的定义和初始化
- 取地址和间接寻址运算符
- 指针变量作函数参数
- 函数指针



指针 (Pointer)

- 指针是“稀饭”最挚爱的武器
 - * 稀饭 == C Fans
- C的高效、高能主要来自于指针
- 很多“Mission Impossible”由指针完成
 - * 大多数语言都有无数的“不可能”
 - * 而C语言是
 - “一切皆有可能” —— 
 - “Impossible is Nothing” —— 



指针 (Pointer)

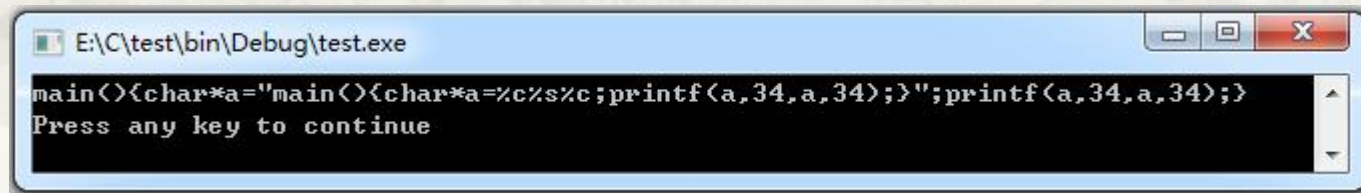
Mission Impossible

源程序

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

运行结果

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



指针 (Pointer)

强转与指针，并称C语言的两大神器
用好了可呼风唤雨，威力无比
用不好也会伤及自身



屠龙刀



倚天剑

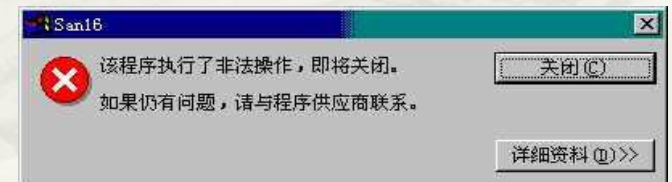


指针 (Pointer)

- 是谁惹的祸？
 - 几乎全是由指针和数组引起的非法内存访问导致的
- 黑客攻击服务器利用的bug绝大部分都是指针和数组造成的



- 理解指针要从变量的地址谈起



9.1变量的内存地址

内存中的每个字节都有唯一的编号（地址）

地址按字节编号，其字长一般与主机相同

32位机使用32位地址，最多支持 2^{32} 字节内存(4G)

变量的地址

```
int a=0;
```

&a

0x0037b000

0x0037b001

0x0037b002

0x0037b003

0

0

0

0

Contents

Contents

Contents

Contents

Contents

Contents

Contents

某存储区域

按变量名读取变量的值

a

变量名

地址是一个无符号整数，从0开始，依次递增。在表达和交流时，通常把地址写成十六进制数

内存

0x00000000

0x00000001

0x00000002

0xFFFFFFFF

4G内存

9.1 变量的内存地址

【例9.1】使用取地址运算符&取出变量的地址，然后将其显示在屏幕上。

```
1  #include <stdio.h>
2  int main()
3  {
4      int a = 0, b = 1;
5      char c = 'A';
6      printf("a is %d, &a is %p\n", a, &a);
7      printf("b is %d, &b is %p\n", b, &b);
8      printf("c is %c, &c is %p\n", c, &c);
9      return 0;
10 }
```

%p表示输出变量a的地址值

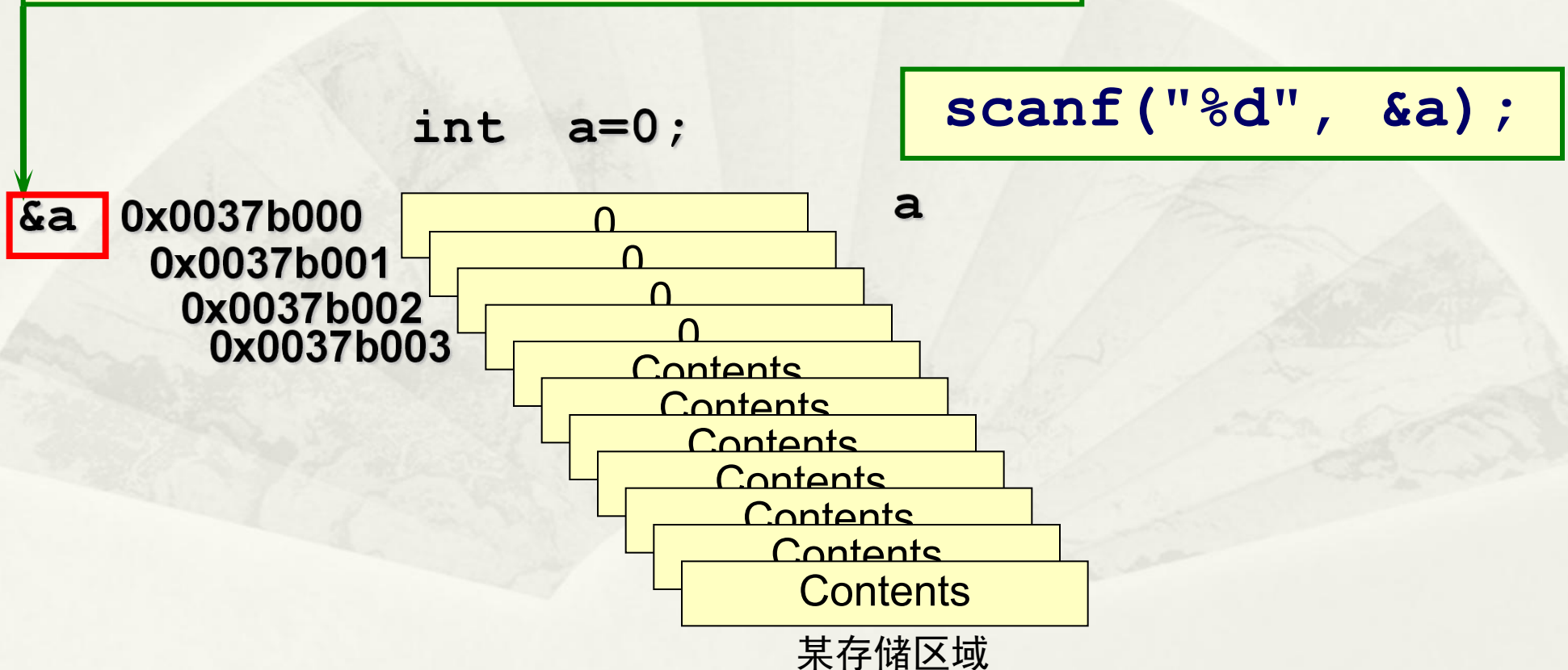
a is 0, &a is 0023FF74

b is 1, &b is 0023FF70

c is A, &c is 0023FF6F

9.1 变量的内存地址

直接寻址：按变量的地址直接访问



Errors



- `int i;`
`scanf("%d", i);`
/ 这样会如何? */*

`i`的值被当作地址。如`i`值为100，则输入的整数就会从地址100开始写入内存

- `char c;`
`scanf("%d", &c);`
/ 这样呢? */*

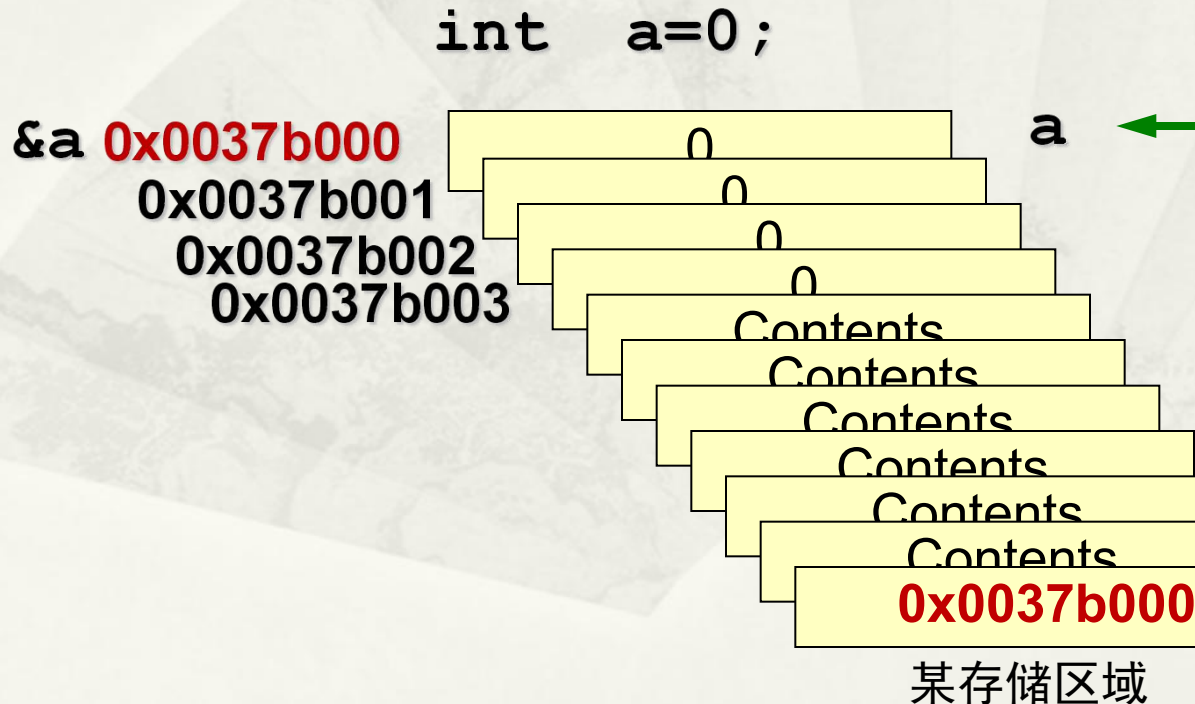
输入数据以`int`的二进制形式写到`c`所在的内存空间。

`c`所占内存不足以放下一个`int`，其后的空间也被覆盖



9.1变量的内存地址

间接寻址：通过存放变量地址的其他变量访问该变量



用什么类型的变量来存放地址呢？



9.2 指针变量的定义和初始化

- 用什么类型的变量来存放变量的地址呢？
 - * 指针（Pointer）类型
- 指针变量——具有指针类型的变量
- 变量的指针 \longleftrightarrow 变量的地址

指针变量

变量的32位地址值

指向



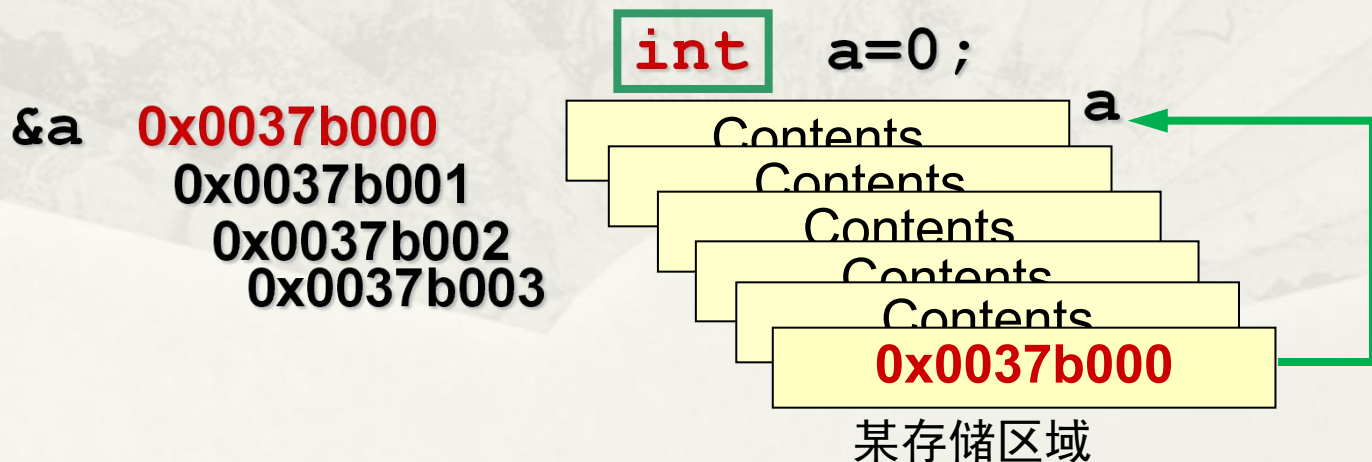
变量的地址

变量的值



9.2 指针变量的定义和初始化

- 保存32位地址值的指针变量占4个字节的内存，这4个字节中保存了一个地址
 - 只知道这些是不够的
- 从这个地址开始多少个字节内的数据是有效的呢？
- 用什么数据类型去理解这些数据呢？
 - 指针的基类型就是回答这个问题的



9.2 指针变量的定义和初始化

【例9-1】
 指针变量指向的数据类型, 称为**基类型** 变量的地址值

```

1  // 例9-1 指针变量的定义和初始化
2  int main()
3  {
4      int a = 0;
5      char c = 'A';
6      int *pa, *pb;          /* 定义了可以指向整型数据的指针变量 pa 和 pb */
7      char *pc;              /* 定义了可以指向字符型数据的指针变量 pc */
8      printf("a is %d, &a is %p, pa is %p\n", a, &a, pa);
9      printf("b is %d, &b is %p, pb is %p\n", b, &b, pb);
10     printf("c is %c, &c is %p, pc is %p\n", c, &c, pc);
11     return 0;
12 }
  
```

告诉编译器, **pa** 是一个指针变量, 占**4**字节内存, 需要用**一个int型变量的地址**给它赋值, 但**pa**并未具体指向某个**int型变量**

```

a is 0, &a is 0023FF74, pa is 0023FF78
b is 1, &b is 0023FF70, pb is 00401394
c is A, &c is 0023FF6F, pc is 77C04E42
  
```

9.2 指针变量的定义和初始化

【例9.2】使用指针变量在屏幕上显示变量的地址值

```
1  #include <stdio.h>
2  int main()
3  {
4      int a = 0, b = 1;
5      char c = 'A';
6      int *pa, *pb;          /* 定义了可以指向整型数据的指针变量 pa 和 pb */
7      char *pc;              /* 定义了可以指向字符型数据的指针变量 pc */
8      printf("a is %d, &a is %p, pa is %p\n", a, &a, pa);
9      printf("b is %d, &b is %p, pb is %p\n", b, &b, pb);
10     printf("c is %c, &c is %p, pc is %p\n", c, &c, pc);
11     return 0;
12 }
```

指针变量使用之前必须初始化
Never use uninitialized pointers

warning: local variable 'pa' used without having been initialized

warning: local variable 'pb' used without having been initialized

warning: local variable 'pc' used without having been initialized 16/53

9.2 指针变量的定义和初始化

【例9.2】使用指针变量在屏幕上显示变量的地址值

```
1  #include <stdio.h>
2  int main()
3  {
4      int a = 0;
5      char c = 'A';
6      int *pa = NULL, *pb = NULL; /* 定义指针变量并用 NULL 对其初始化 */
7      char *pc = NULL;           /* 定义指针变量并用 NULL 对其初始化 */
8      printf("a is %d, &a is %p, pa is %p\n", a, &a, pa);
9      printf("b is %d, &b is %p, pb is %p \n", b, &b, pb);
10     printf("c is %c, &c is %p, pc is %p \n", c, &c, pc);
11     return 0;
12 }
```

如果你不知把它指向哪里，那就指向**NULL**

a is 0, &a is 0013FF7C, pa is 00000000

b is 1, &b is 0013FF78, pb is 00000000

c is A, &c is 0013FF74, pc is 00000000

9.2 指针变量的定义和初始化

- 何谓空指针？
 - * 值为NULL的指针，即无效指针
- 既然0（NULL）用来表示空指针，那么空指针就是指向地址为0的单元的指针吗？
- 答案：不一定
 - * 每个C编译器都被允许用不同的方式来表示空指针
 - * 并非所有编译器都使用0地址
 - * 某些编译器为空指针使用不存在的内存地址
 - * 硬件会检查出这种试图通过空指针访问内存的方式



9.2 指针变量的定义和初始化

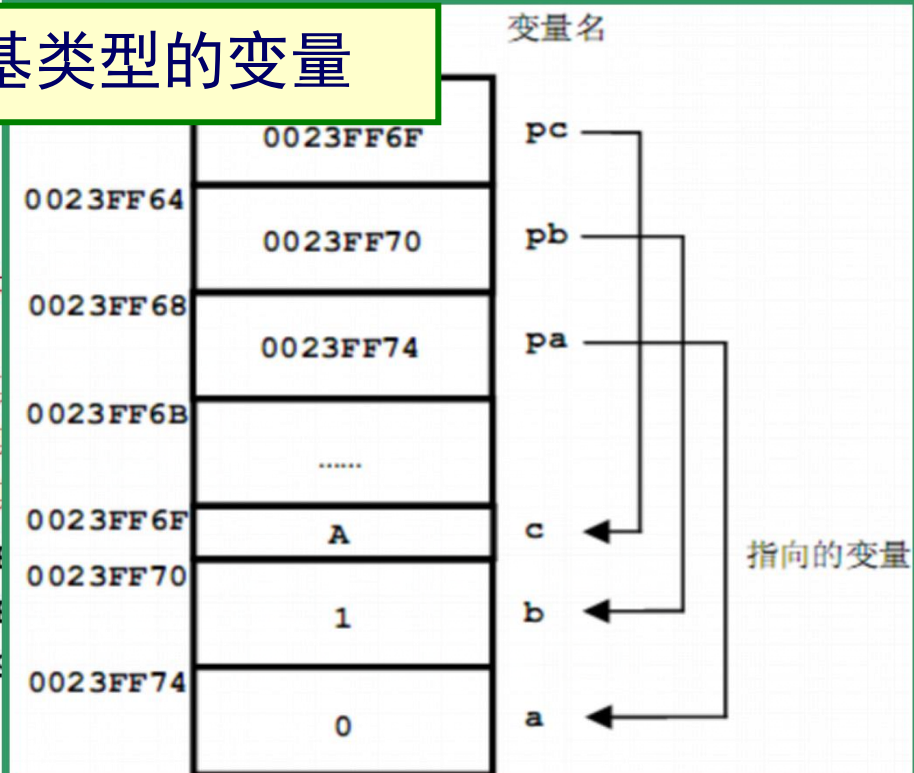
【例9.2】使用指针变量在屏幕上显示变量的地址值

```

1  #include <stdio.h>
2  int main()
3  {
4      int a = 0, b = 1;
5      char c = 'A';
6      int *pa, *pb;      /* 定义指针变量 pa 和 pb */
7      char *pc;          /* 定义指针变量 pc */
8      pa = &a;           /* 初始化指针变量 pa 使其指向变量 a */
9      pb = &b;           /* 初始化指针变量 pb 使其指向变量 b */
10     pc = &c;            /* 初始化指针变量 pc 使其指向变量 c */
11     printf("a is %d, &a is %p, pa is %p, &pa is %p\n", a, &a, pa, &pa);
12     printf("b is %d, &b is %p, pb is %p, &pb is %p\n", b, &b, pb, &pb);
13     printf("c is %c, &c is %p, pc is %p, &pc is %p\n", c, &c, pc, &pc);
14     return 0;
15 }

```

指针变量只能指向同一基类型的变量



a is 0, &a is 0023FF74, pa is 0023FF74, &pa is 0023FF68

b is 1, &b is 0023FF70, pb is 0023FF70, &pb is 0023FF64

c is A, &c is 0023FF6F, pc is 0023FF6F, &pc is 0023FF60

9.2 指针变量的定义和初始化

【例9.2】使用指针变量在屏幕上显示变量的地址值

```
1  #include <stdio.h>
2  int main()
3  {
4      int a = 0, b = 1;
5      char c = 'A';
6      int *pa, *pb;      /* 定义指针变量 pa 和 pb */
7      char *pc;          /* 定义指针变量 pc */
8      pa = &a;           /* 初始化指针变量 pa 使其指向 a */
9      pb = &b;           /* 初始化指针变量 pb 使其指向 b */
10     pc = &c;           /* 初始化指针变量 pc 使其指向 c */
11     printf("a is %d, &a is %p, pa is %p, &pa is %p\n", a, &a, pa, &pa);
12     printf("b is %d, &b is %p, pb is %p, &pb is %p\n", b, &b, pb, &pb);
13     printf("c is %c, &c is %p, pc is %p, &pc is %p\n", c, &c, pc, &pc);
14     return 0;
15 }
```

不能写成: `int* pa, pb;`

a is 0, &a is 0023FF74, pa is 0023FF74, &pa is 0023FF68

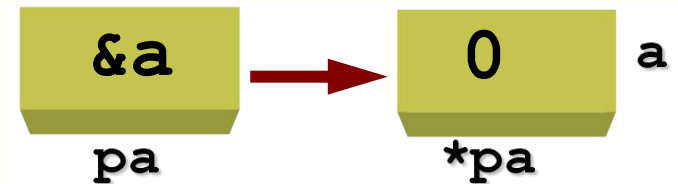
b is 1, &b is 0023FF70, pb is 0023FF70, &pb is 0023FF64

c is A, &c is 0023FF6F, pc is 0023FF6F, &pc is 0023FF60

9.3 间接寻址运算符

【例9.3】使用指针变量，通过**间接寻址**输出变量的值

```
1  #include <stdio.h>
2  int main()
3  {
4      int a = 0, b = 1;
5      char c = 'A';
6      int *pa = &a, *pb = &b; /* 在定义指针变量 pa 和 pb 的同时对其初始化 */
7      char *pc = &c;          /* 在定义指针变量 pc 的同时对其初始化 */
8      printf("a is %d, &a is %p, pa is %p, *pa is %d\n", a, &a, pa, *pa);
9      printf("b is %d, &b is %p, pb is %p, *pb is %d\n", b, &b, pb, *pb);
10     printf("c is %c, &c is %p, pc is %p, *pc is %c\n", c, &c, pc, *pc);
11     return 0;
12 }
```



a is 0, &a is 0023FF74, pa is 0023FF74, *pa is 0

b is 1, &b is 0023FF70, pb is 0023FF70, *pb is 1

c is A, &c is 0023FF6F, pc is 0023FF6F, *pc is A

9.3 间接寻址运算符

【例9.3】使用指针变量，通过间接寻址输出变量的值

```

1  #include <stdio.h>
2  int main()
3  {
4      int a = 0, b = 1;
5      char c = 'A';
6      int *pa = &a, *pb = &b; /* 在定义指针变量 pa 和 pb 的同时对其初始化 */
7      char *pc = &c;          /* 在定义指针变量 pc 的同时对其初始化 */
8      *pa = 9;                /* 修改指针变量 pa 所指向的变量的值 */
9      printf("a is %d, &a is %p, pa is %p, *pa is %d\n", a, &a, pa, *pa);
10     printf("b is %d, &b is %p, pb is %p, *pb is %d\n", b, &b, pb, *pb);
11     printf("c is %c, &c is %p, pc is %p, *pc is %c\n", c, &c, pc, *pc);
12
13

```

引用指针指向的变量的值，称为指针的解引用(Pointer Dereference)

a is 9, &a is 0023FF74, pa is 0023FF74, *pa is 9

b is 1, &b is 0023FF70, pb is 0023FF70, *pb is 1

c is A, &c is 0023FF6F, pc is 0023FF6F, *pc is A



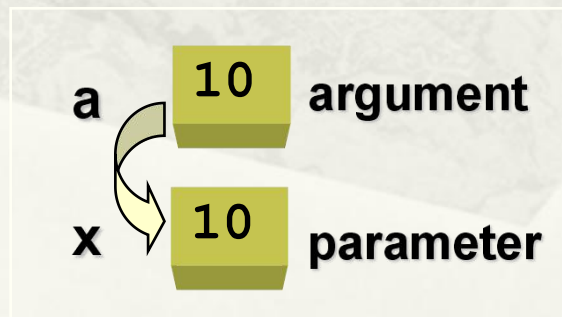
为什么
要用指
针?





9.4 按值调用与按地址调用

- 普通变量作函数参数—按值调用
 - * 实参的值不随形参值的改变而改变
- 形参 (**parameter**) ← 实参变量的值
- 指针做函数参数—按地址调用
 - * 为了在被调函数中修改其无法直接访问的实参的值
- 指针形参(**pointer parameter**) ← 实参变量的地址



【例9.4】演示按值调用

```

1  #include <stdio.h>
2  void Fun(int par);
3  int main()
4  {
5      int arg = 1;
6      printf("arg = %d\n", arg);
7      Fun(arg);
8      printf("arg = %d\n", arg);
9      return 0;
10 }
11 void Fun(int par)
12 {
13     printf("par = %d\n", par);
14     par = 2;
15 }

```

传变量的值

arg = 1
par = 1
arg = 1

形参值的改变
不影响对应的实参

【例9.5】演示按地址调用

```

1  #include <stdio.h>
2  void Fun(int *par);
3  int main()
4  {
5      int arg = 1;
6      printf("arg = %d\n", arg);
7      Fun(&arg);
8      printf("arg = %d\n", arg);
9      return 0;
10 }
11 void Fun(int *par)
12 {
13     printf("par = %d\n", *par);
14     *par = 2;
15 }

```

传变量的地址

arg = 1
par = 1
arg = 2

指针变量作函数形参可以
修改相应实参的值, 为函数
提供了修改变量值的手段

【例9.4】演示按值调用

```

1  #include <stdio.h>
2  void Fun(int par);
3  int main()
4  {
5      int arg = 1 ;
6      printf("arg = %d\n", arg);
7      Fun(arg);
8      printf("arg = %d\n", arg);
9      return 0;
10 }
11 void Fun(int par)
12 {
13     printf("par = %d\n", par);
14     par = 2;
15 }

```

arg = 1
par = 1
arg = 1

```

1  #include <stdio.h>
2  int Fun(int par);
3  int main()
4  {
5      int arg = 1 ;
6      printf("arg = %d\n", arg);
7      arg = Fun(arg);
8      printf("arg = %d\n", arg);
9      return 0;
10 }
11 int Fun(int par)
12 {
13     printf("par = %d\n", par);
14     par = 2;
15     return par;
16 }

```

return仅限于
从函数返回一个值

arg = 1
par = 1
arg = 2

例9.6：编写函数实现两数的互换



程序 1：简单变量作函数参数

程序 2：指针变量作函数参数

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

实参

```
int main()
{
    int a, b;
    a = 5;
    b = 9;
    Swap(&a, &b);
    printf("a=%d,b=%d", a, b);
    return 0;
}
```

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

形参

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

Not Work! Why?



例9.6：编写函数实现两数的互换

主调函数

```
int main()
{
    int a = 5, b = 9;

    Swap(a, b);

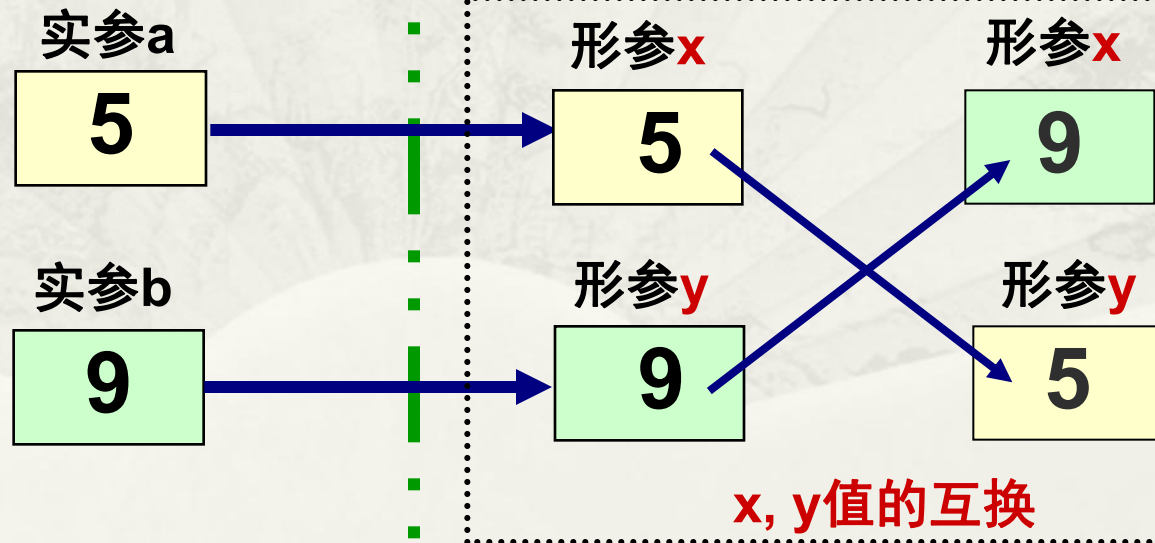
    printf("a=%d,b=%d", a, b);
    return 0;
}
```

被调函数

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

函数体内

传值调用





例9.6：编写函数实现两数的互换

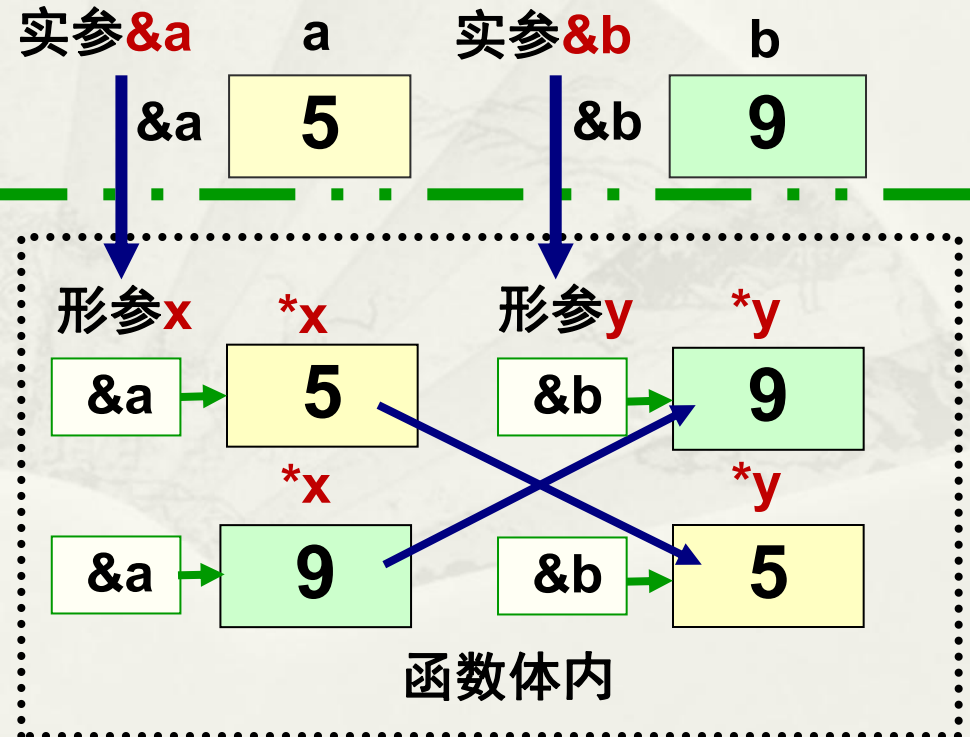
主调函数

```
int main()  
{  
    int a = 5, b = 9;  
  
    Swap(&a, &b);  
  
    printf("a=%d,b=%d", a, b);  
    return 0;  
}
```

传地址调用

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

被调函数



例9.6: 编写函数实现两数的互换



主调函数

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

某些编译器检查实参和形参的数据类型是否匹配，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

```
void Swap(int *x, int *y)
```

```
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

被调函数

```
warning: passing argument 1 of 'Swap' makes pointer from integer without a cast
note: expected 'int *' but argument is of type 'int'
warning: passing argument 2 of 'Swap' makes pointer from integer without a cast
note: expected 'int *' but argument is of type 'int'
```



例9.6: 编写函数实现两数的互换-数组实现



```
int main()  
{  
    int a[2] = {5, 9};  
  
    Swap(a);  
  
    printf("%d,%d", a[0], a[1]);  
    return 0;  
}
```

主调函数

```
void Swap(int p[])  
{  
    int temp;  
    temp = p[0];  
    p[0] = p[1];  
    p[1] = temp;  
}
```

被调函数

传地址调用

实参
数组名a

a[0]	a[1]
5	9

形参
数组名p

p[0]	p[1]
5	9

p[0]	p[1]
9	5

函数体内



Errors



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

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

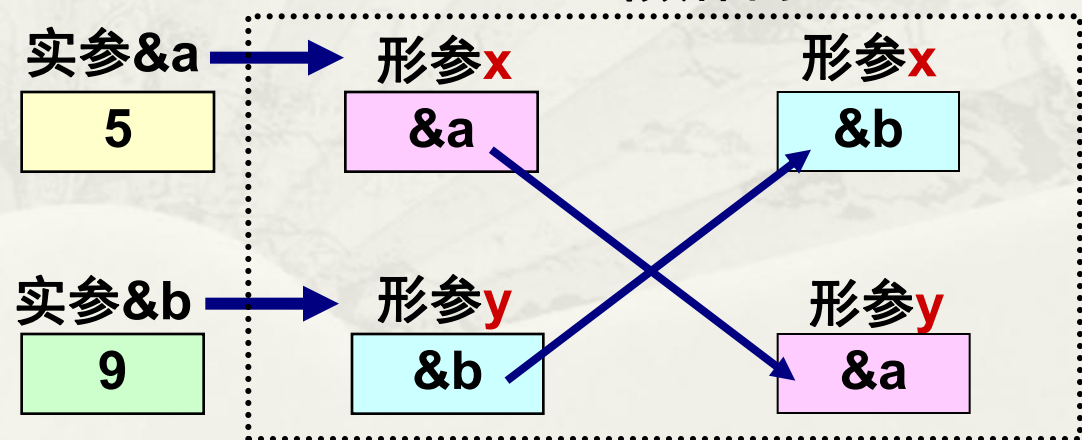
指针pTemp未初始化
指针pTemp指向哪里未知
对未知单元写操作是危险的
不能借助一个未初始化的
指针变量进行两数互换

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

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

借助指针pTemp
交换的是地址值
不是指针指向的内容

函数体内



只是x, y (地址值) 的互换 31/53

*9.5用指针变量作函数参数的程序实例

■ 【例9.7】 计算并输出最高分及相应学生的学号

```
4  int main()  
5  {  
6      int  score[N], maxScore;  
7      int   n, i;  
8      long  num[N], maxNum;  
9      printf("How many students?");  
10     scanf("%d", &n);  
11     printf("Input student's ID and score:\n");  
12     for (i=0; i<n; i++)  
13     {  
14         scanf("%ld%d", &num[i], &score[i]); /* 字母d前为字母l */  
15     }  
16     FindMax(score, num, n, maxScore, maxNum); /* 按值调用函数 */  
17     printf("maxScore = %d, maxNum = %ld\n", maxScore, maxNum);  
18     return 0;  
19 }
```

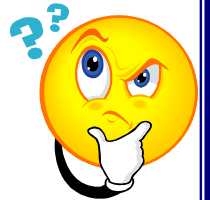
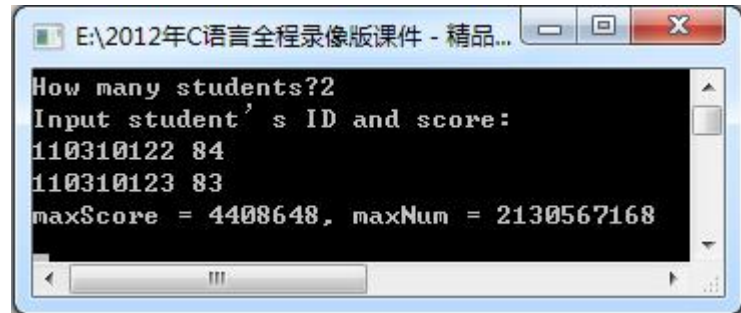


■ 【例9.7】 计算并输出最高分及相应学生的学号

```
void FindMax(int score[], long num[], int n, int pMaxScore, long pMaxNum)
{
    int i;
    pMaxScore = score[0];
    pMaxNum = num[0];
    for (i=1; i<n; i++)
    {
        if (score[i] > pMaxScore)
        {
            pMaxScore = score[i];
            pMaxNum = num[i];
        }
    }
}
```

真正原因：普通变量作函数参数按值调用
不能在调函数中改变相应实参的值

Not Work! Why?



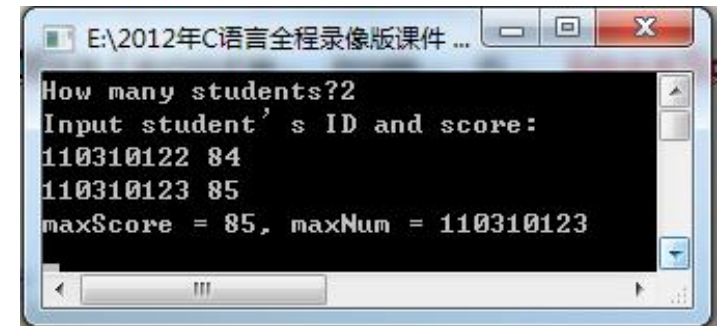
```
FindMax(score, num, n, maxScore, maxNum);
```

VC warning: local variable 'maxNum' used without having been initialized
warning: local variable 'maxScore' used without having been initialized

CB warning: 'maxScore' is used uninitialized in this function
warning: 'maxNum' is used uninitialized in this function

■ 【例9.7】 计算并输出最高分及相应学生的学号

```
void FindMax(int score[], long num[], int n, int *pMaxScore, long *pMaxNum)
{
    int i;
    *pMaxScore = score[0];
    *pMaxNum = num[0];
    for (i=1; i<n; i++)
    {
        if (score[i] > *pMaxScore)
        {
            *pMaxScore = score[i];
            *pMaxNum = num[i];
        }
    }
}
```



```
FindMax(score, num, n, &maxScore, &maxNum);
```

9.6 函数指针及其应用

- 函数指针(Function Pointer)就是指向函数的指针变量

数据类型 (*指针变量名) (形参列表);

- 例:

```
int (*f) (int a, int b);
```

- 函数指针f指向的函数原型为:

```
int 函数名 (int a, int b);
```

- 令 **f = Fun**, 就是让f指向函数fun()
- 编译器将不带()的函数名解释为该函数的入口地址
- 函数指针变量存储的是函数在内存中的入口地址

9.6 函数指针及其应用

```
int (*f)(int a, int b);
```

■ 常见错误:

- * 忘了写前一个()
——`int *f(int a, int b);`
 - 声明了一个函数名为`f`、返回值是整型指针类型的函数
- * 忘了写后一个()
——`int (*f);`
 - 定义了一个整型指针变量
- * 定义时的参数类型与指向的函数参数类型不匹配
`int (*f)(float a, float b);`
- * 不建议写成——`int (*f)();`

9.6 函数指针及其应用

例：用函数指针变量作函数参数，求最大值、最小值和两数之和

```
void Fun(int x, int y, int (*f)(int, int));  
  
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);  
}
```

```
int Max(int x, int y);  
int Min(int x, int y);  
int Add(int x, int y);  
  
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;  
}
```

9.6函数指针及其应用

- 函数指针的应用
 - * 编写通用性更强的函数
- 典型实例1
 - * 计算函数的定积分
- 典型实例2
 - * 通用的排序函数
 - * 既能按照升序排序，又能按照降序排序

为什么要舍
近求远呢？

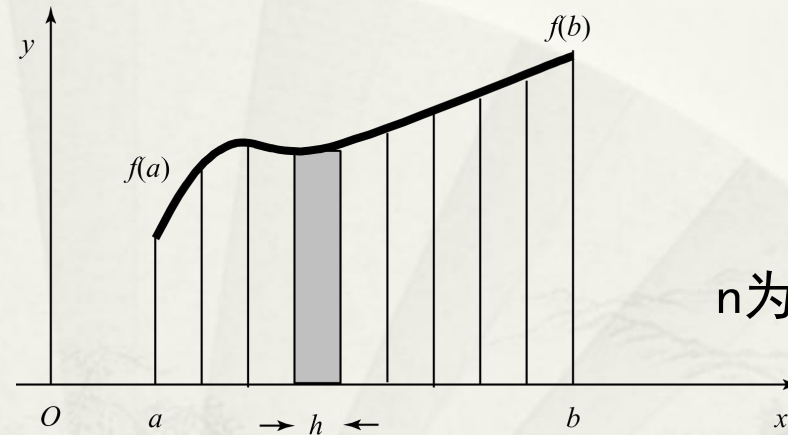


9.6 函数指针及其应用

【习题9.6】梯形法计算函数的定积分

$$y_1 = \int_0^1 (1 + x^2) dx$$

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



$$\begin{aligned} y &= \frac{h}{2}(f(a) + f(a+h)) + \frac{h}{2}(f(a+h) + f(a+2h)) + \dots + \frac{h}{2}(f(a+(n-1)h) + f(b)) \\ &= \frac{h}{2}(f(a) + 2f(a+h) + 2f(a+2h) + \dots + 2f(a+(n-1)h) + f(b)) \\ &= h\left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(a+i \cdot h)\right) \quad h = \frac{b-a}{n} \end{aligned}$$

9.6 函数指针及其应用

梯形法计算函数**F1**的定积分 $y = h(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+i \cdot h)) \quad h = \frac{b-a}{n}$

```
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 += F1(a + i * h);
    }
    return s * h;
}
```

$$y_1 = \int_a^b f_1(x) dx$$

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

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

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


9.6 函数指针及其应用

梯形法计算函数**F2**的定积分 $y = h(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+i \cdot h)) \quad h = \frac{b-a}{n}$

```
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 += F2(a + i * h);
    }
    return s * h;
}
```

$$y_2 = \int_a^b f_2(x) dx$$

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

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

```
y1 = IntegralF1(0.0, 1.0);
y2 = IntegralF2(0.0, 3.0);
```

9.6 函数指针及其应用

计算定积分的通用函数

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

```
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 += (*f)(a + i * h);
    }
    return s * h;
}
```

$$y = \int_a^b f(x) dx$$

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

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

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

9.6函数指针及其应用

【例9.8】修改例8.8中的排序函数，使其既能实现对学生成绩的升序排序，又能实现对学生成绩的降序排序

- 先不使用函数指针编程

```
1    #include <stdio.h>
2    #define N 40
3    int ReadScore(int score[]);          /* 成绩输入函数原型 */
4    void PrintScore(int score[], int n); /* 成绩输出函数原型 */
5    void AscendingSort(int a[], int n);  /* 升序排序原函数型 */
6    void DescendingSort(int a[], int n); /* 降序排序原函数型 */
```

9.6 函数指针及其应用

```
7   int main()
8   {
9       int score[N], n;
10      int order;    /* 值为 1 表示升序排序, 值为 2 表示降序排序 */
11      n = ReadScore(score);
12      printf("Total students are %d\n", n);
13      printf("Enter 1 to sort in ascending order, \n");
14      printf("Enter 2 to sort in descending order:");
15      scanf("%d", &order);
16      printf("Data items in original order\n");
17      PrintScore(score, n);
18      if (order == 1)
19      {
20          AscendingSort(score, n); /* 按升序排序 */
21          printf("Data items in ascending order\n");
22      }
23      else
24      {
25          DescendingSort(score, n); /* 按降序排序 */
26          printf("Data items in descending order\n");
27      }
28      PrintScore(score, n);
29      return 0;
30  }
```

```
Input score:84 ✓
Input score:83 ✓
Input score:88 ✓
Input score:87 ✓
Input score:61 ✓
Input score:-1 ✓
Total students are 5
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order:1✓
Data items in original order
    84 83 88 87 61
Data items in ascending order
    61 83 84 87 88
```



```
52  /* 函数功能： 选择法实现数组a的升序排序 */
53  void AscendingSort(int a[], int n)
54  {
55      int i, j, k, temp;
56      for (i=0; i<n-1; i++)
57      {
58          k = i;
59          for (j=i+1; j<n; j++)
60          {
61              if (a[j] < a[k])
62              {
63                  k = j;
64              }
65          }
66          if (k != i)
67          {
68              temp = a[k];
69              a[k] = a[i];
70              a[i] = temp;
71          }
72      }
73  }
```

```
74  /* 函数功能： 选择法实现数组a的降序排序 */
75  void DescendingSort(int a[], int n)
76  {
77      int i, j, k, temp;
78      for (i=0; i<n-1; i++)
79      {
80          k = i;
81          for (j=i+1; j<n; j++)
82          {
83              if (a[j] > a[k])
84              {
85                  k = j;
86              }
87          }
88          if (k != i)
89          {
90              temp = a[k];
91              a[k] = a[i];
92              a[i] = temp;
93          }
94      }
95  }
```

9.6 函数指针及其应用

```
int ReadScore(int score[])
{
    int i = -1;
    do{
        i++;
        printf("Input score:");
        scanf("%d", &score[i]);
    } while (score[i] >= 0);
    return i;
}
```

```
void PrintScore(int score[], int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        printf("%4d", score[i]);
    }
    printf("\n");
}
```

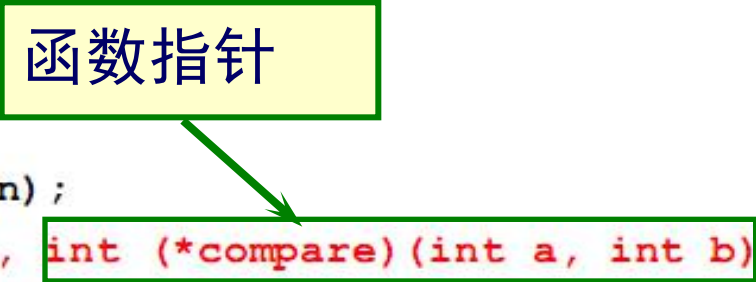
9.6函数指针及其应用

【例9.9】修改例9.8中的程序实例，用函数指针编程实现一个通用的排序函数，对学生成绩既能实现升序排序，又能实现降序排序

■ 使用函数指针编程

```
1  #include <stdio.h>
2  #define N 40
3  int ReadScore(int score[]);
4  void PrintScore(int score[], int n);
5  void SelectionSort(int a[], int n, int (*compare)(int a, int b));
6  int Ascending( int a, int b );
7  int Descending( int a, int b );
```

函数指针



9.6 函数指针及其应用

```
8   int main()
9   {
10      int score[N], n;
11      int order;          /* 值为 1 表示升序排序, 值为 2 表示降序排序 */
12      n = ReadScore(score); /* 输入成绩, 返回学生人数 */
13      printf("Total students are %d\n", n);
14      printf("Enter 1 to sort in ascending order, \n");
15      printf("Enter 2 to sort in descending order:");
16      scanf("%d", &order);
17      printf("Data items in original order\n");
18      PrintScore(score, n); /* 输出排序前的成绩 */
19      if (order == 1)
20      {
21          SelectionSort(score, n, Ascending); /* 函数指针指向 Ascending() */
22          printf("Data items in ascending order\n");
23      }
24      else
25      {
26          SelectionSort(score, n, Descending); /* 函数指针指向 Descending() */
27          printf("Data items in descending order\n");
28      }
29      PrintScore(score, n); /* 输出排序后的成绩 */
30      return 0;
31  }
```


9.6 函数指针及其应用

```

53  /* 函数功能: 通用的交换法排序 */
54  void SelectionSort(int a[], int n, int (*compare)(int a, int b))
55  {
56      int i, j, k, temp;
57      for (i=0; i<n-1; i++)
58      {
59          k = i;
60          for (j=i+1; j<n; j++)
61          {
62              if ((*compare)(a[j], a[k])) /*调用函数指针compare指向的函数*/
63              {
64                  k = j;
65              }
66          }
67          if (k != i)
68          {
69              temp = a[k];
70              a[k] = a[i];
71              a[i] = temp;
72          }
73      }
74  }

```

```
int Ascending(int a, int b)
```

```
{
```

```
    return a < b; //为真,则按升序排序
```

```
}
```

```
if (a[j] < a[k])
```

```
int Descending(int a, int b)
```

```
{
```

```
    return a > b; //为真,则按降序排序
```

```
}
```

```
if (a[j] > a[k])
```

```
Swap(&a[i], &a[k]);
```


小结



■ 指针变量

- 指针类型的变量，保存地址型数据

■ 指针变量与其他类型变量的共性

- 在内存中占据一定大小的存储单元（通常4个字节）
- 先定义，后使用

■ 特殊性

- 指针变量中保存的内容只能是地址（变量或函数的地址）
- 必须初始化后才能使用，否则指向不确定的存储单元
- 只能指向同一基类型的变量
- 可参与的运算：加、减整数，自增、自减、关系、赋值

小结



- 指针变量
 - 定义不同基类型的指针，是明确指针指向单元的内容是什么
 - 初始化的目的，是明确指针指向了哪里
- 使用指针变量的基本原则
 - 明确指针指向了哪里
 - 明确指针指向单元的内容是什么
 - 永远不要使用未初始化的指针变量
 - 一个（xx型）的指针指向一个（xx型）的变量

小结



- 指针的一个重要应用之一
 - 作函数参数，向函数传递变量或函数的地址
- 指向变量的指针，作函数参数
 - 按地址调用，传递变量在内存中的地址（用取地址运算符）
 - 被调函数根据该地址读写它不能直接访问的变量的值（用间接寻址运算符，指针的解引用）
- 指向函数的指针，作函数参数
 - 按地址调用，传递函数在内存中的入口地址（函数名）
 - 被调函数根据传入的不同地址调用不同的函数

