



清华大学

腾讯会议 ID: 312 165 703

Tsinghua University

计算机程序设计基础

第11讲 指针

沈瑜 (010-62782951)

shenyu@tsinghua.edu.cn

清华大学电机系

2022.11.22





主要内容

- C程序设计举例
- 指针的概念
- 指针操作
- 指针的应用
 - 指针变量作为函数参数
 - 指针变量作为函数返回值

参考教材： 第8章、第9.3、9.4节





11.1 C程序设计举例

- 例1: 编写函数，交换两个字符的值

验证：输入参数为 'c', 'd', 输出为'd', 'c'



实现功能前先写验证程序

```
#include <stdio.h>
#include <assert.h>

#define DEBUG

int main()
{
#ifdef DEBUG
    char c='c', d='d';
    swap( c, d );
    assert( c=='d' );
#endif
    return 0;
}
```

编程好习惯

程序

```
#include <stdio.h>
#include <assert.h>

#define DEBUG

void swap( char c1, char c2 )
{
    char cTemp=c1;
    c1 = c2;
    c2 = cTemp;
}

int main()
{
    #ifdef DEBUG
        char c='c', d='d' ;
        swap( c, d );
        assert( c=='d' );
    #endif
    return 0;
}
```

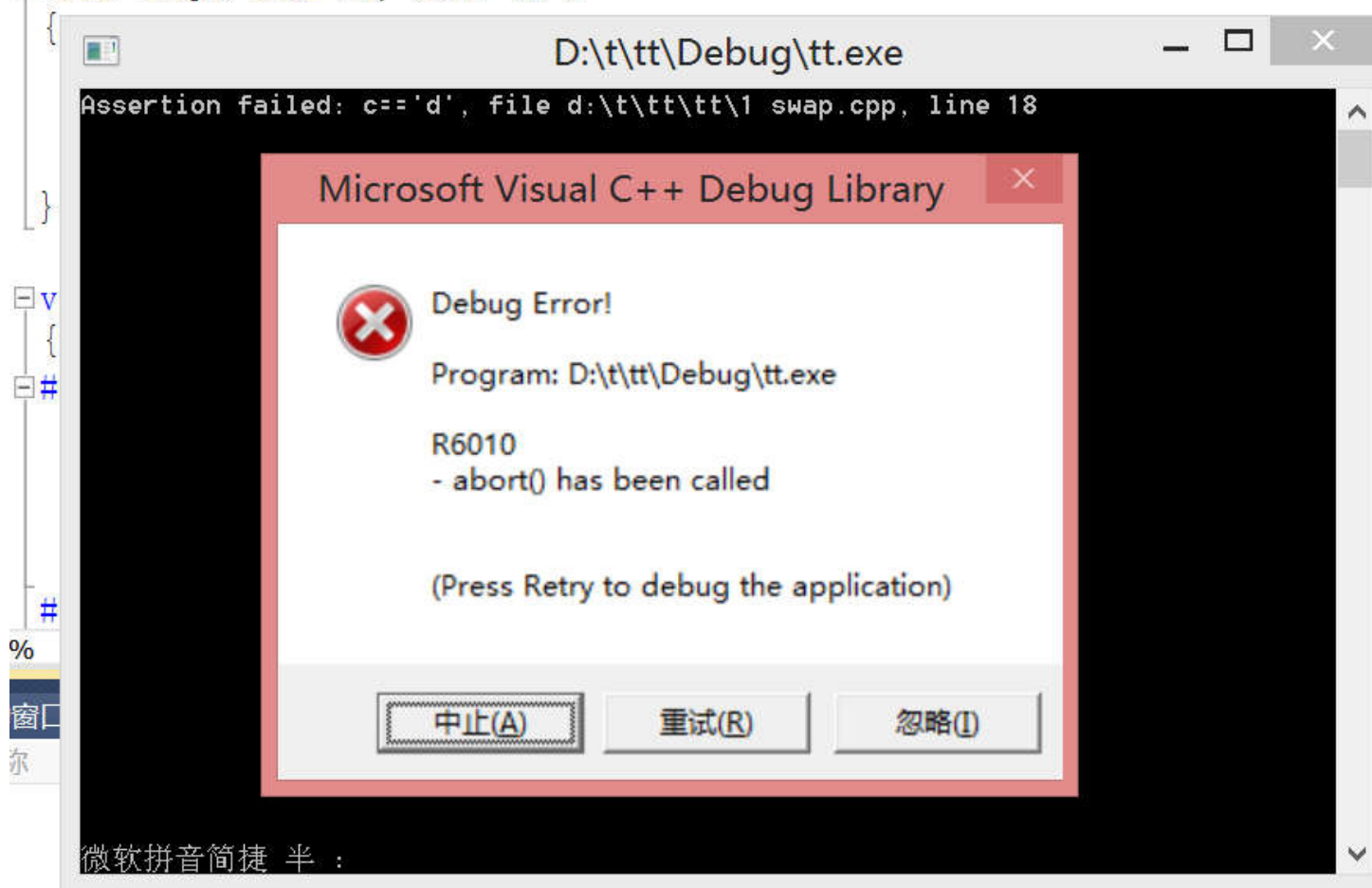
这样写，有问题吗？

执行结果

为什么会这样?

```
#define DEBUG
```

```
void swap( char c1, char c2 )
```



微软拼音简捷 半 :

原因分析

- 调用**swap**时，参数传递过程：
 - 将**c**赋给**c1**, **d**赋给**c2**
- 执行**swap**后：
 - **c1**与**c2**的值互相交换
- 问题：
 - 交换的是**c1**与**c2**的值，而不是**c**与**d**的值

```
#include <stdio.h>
#include <assert.h>

#define DEBUG

void swap( char c1, char c2 )
{
    char cTemp=c;
    c1 = c2;
    c2 = cTemp;
}

int main()
{
    #ifdef DEBUG
        char c='c', d='d';
        swap( c, d );
        assert( c=='d' );
    #endif
    return 0;
}
```

解决方案一：不使用子函数

- 不使用子函数，直接实现swap功能

```
#include <stdio.h>
#include <assert.h>

#define DEBUG

int main()
{
#ifdef DEBUG
    char c='c', d='d';
    char cTemp=c;
    c = d;
    d = cTemp;
    assert( c=='d' );
#endif
    return 0;
}
```



解决方案二：全局变量

- 将需要交换的变量定义为全局变量

```
#include <stdio.h>
#include <assert.h>

#define DEBUG

char c='c', d='d';

void swap( )
{
    char cTemp=c;
    c = d;
    d = cTemp;
}

int main()
{
    #ifdef DEBUG
        swap( );
        assert( c=='d' );
    #endif
    return 0;
}
```

解决方案三：？

- 方案一、方案二，可以吗？
- 可以。但是，swap函数是非常有用的功能。经常需要在程序中使用。
- 如果不将其封装成函数：
 - (1) 代码重复
 - (2) 程序可读性下降

那么，应该如何解决呢？



问题本质

- 希望:

在被调函数中，修改主调函数中变量的值



这是可能的吗？

- 支持理由：
 - 主调函数中变量生存期长
 - 在被调函数执行过程中，主调函数中变量是存在的
- 不支持理由：
 - 主调函数与被调函数各有各的作用域
 - 在被调函数内看不到主调函数中变量



解决方案三：

- 间接使用主调函数中变量
 - 方法：将变量地址传过去，通过地址间接使用
 - 因为：主调函数中变量生存期长



如何传变量地址？

- 涉及内容
 - 变量地址表示方法？
 - 指针变量
 - 如何获得变量地址？
 - 取地址
 - 如何将获得的变量地址赋值给指针变量？
 - 指针赋值
 - 如何作为函数参数进行传递？
 - 指针作为参数
 - 如何利用获得的地址对变量进行操作？
 - 间接寻址



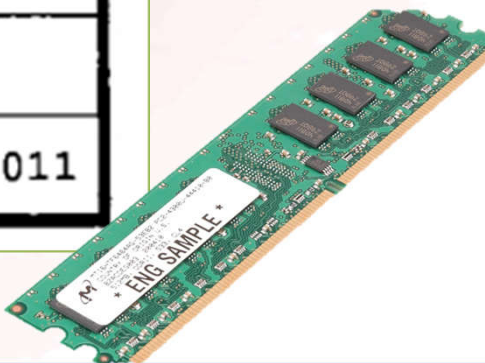


11.2 指针的概念

1. 回顾：内存

地址	内容
0	01010011
1	01110101
2	01110011
3	01100001
4	01101110
	⋮
n-1	01000011

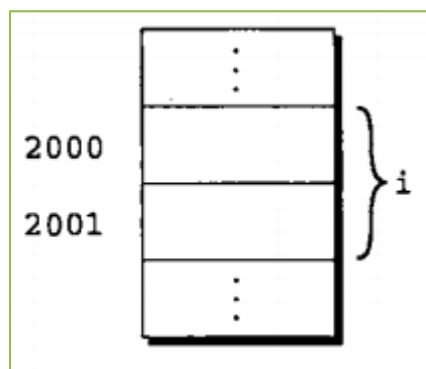
内存



- 程序中用到的变量和函数都转换成二进制位，存储在**内存芯片**中
- 内存芯片中空间的基本单位是字节（**8个bit位**）
 - **4G**芯片，相当与有**4G**个字节存储空间



回顾：指针就是地址



- 程序中每个变量占一个到多个字节，其第一个字节的位置称为**变量的地址**
- 虽然用数表示地址，但其取值范围与整数不同，所以我们用新的类型——**指针类型**存储地址
- 用指针变量**p**存储整型变量**i**的地址时，我们称：
 - ◆ **p指向i**
 - ◆ **p为整型指针**



回顾：空指针

地址	内容
0	01010011
1	01110101
2	01110011
3	01100001
4	01101110
	⋮
n-1	01000011

- 值为0的指针称为空指针
 - 用**NULL**表示
 - 表示指针不指向内存中任何位置



2. 指针变量的定义

- 定义: 数据类型 *变量名

基本数据类型	对应指针类型	说明
int data1, data2;	int *pData1, *pData2;	整型指针
char data1, data2;	char *pData1, *pData2;	字符指针
float data1, data2;	float *pData1, *pData2;	实型指针
double data1, data2;	double *pData1, *pData2;	实型指针

data1, data2 为变量;
pData1, pData2为指针变量

指针变量定义有何特殊?

例1：使用辨析

```
void main()
{
    int* p1, p2;
}
```

相当于

```
void main()
{
    int *p1, p2;
}
```

或

```
void main()
{
    int (* p1), p2;
}
```

- p1 和 p2 都是指针吗？如果是，是什么类型的指针？

p1是整型指针

p2不是指针，是整数

*** 应与变量名写在一起**



例2：使用辨析2

```
void main()
{
    int *p1, data;
    double *p1, *9w;
    char *p$, ch, *c;
}
```

- 对吗？

- 错误1：*9w
 - 变量名不能以数字开头
- 错误2：*p\$
 - 变量名中只能包含数字、下划线与字母



例3：结构指针与文件指针

```
typedef struct
{
    double x, y;    //圆心
    double r;       //半径
} CIRCLE;

void main()
{
    CIRCLE c1={0, 0, 5};
    CIRCLE *p;
    FILE *f1;
}
```

- 对吗？
- p是指向什么类型的指针？
- f1呢？



11.3 指针操作

1. 取地址运算符

- 问题
 - `int a,b;`
 - 变量**a**, **b**在内存中存储地址是?
- 方法
 - 用**&**操作符
 - 在变量前加**&**符号, 求得变量地址
- 例
 - `int a,b;`
 - a的地址是: `&a`
 - b的地址是: `&b`



例1：取地址

```
void main() {  
    int    k = 9;  
  
    printf ("%d = %d\n", &k);  
    printf ("%x = %x\n", &k);  
    printf ("%X = %X\n", &k);  
    printf ("%o = %o\n", &k);  
    printf ("%p = %p\n", &k);  
}
```

```
%d = 5438440  
%x = 52fbe8  
%X = 52FBE8  
%o = 24575750  
%p = 0052FBE8
```

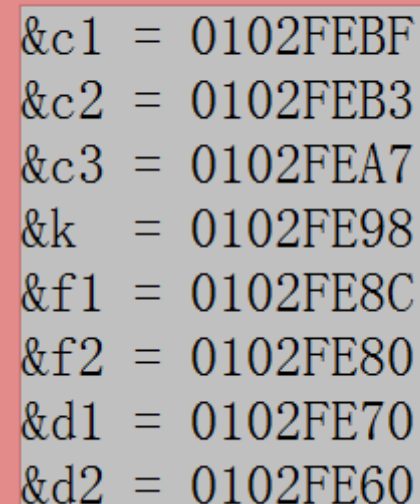
- &, 取变量地址
- %x, 以16进制输出整数(小写形式)
- %X, 以16进制输出整数(大写形式)
- %o, 8进制输出整数
- %p, 输出指针的值
- %, 输出一个%

例2：观察变量地址分布规律

- 有何规律？

```
#include <stdio.h>
void main()
{
    char    c1 = 'c', c2 = 'd', c3 = 'e';
    int     k = 9;
    float   f1 = 10.0, f2 = 0;
    double  d1 = 0.1, d2 = 0.2;

    printf("&c1 = %p\n", &c1);
    printf("&c2 = %p\n", &c2);
    printf("&c3 = %p\n", &c3);
    printf("&k  = %p\n", &k);
    printf("&f1 = %p\n", &f1);
    printf("&f2 = %p\n", &f2);
    printf("&d1 = %p\n", &d1);
    printf("&d2 = %p\n", &d2);
}
```



```
&c1 = 0102FEBF
&c2 = 0102FEB3
&c3 = 0102FEA7
&k  = 0102FE98
&f1 = 0102FE8C
&f2 = 0102FE80
&d1 = 0102FE70
&d2 = 0102FE60
```

地址分布规律

```
&c1 = 0102FEBF
&c2 = 0102FEB3
&c3 = 0102FEA7
&k  = 0102FE98
&f1 = 0102FE8C
&f2 = 0102FE80
&d1 = 0102FE70
&d2 = 0102FE60
```

```
&c1 = 010BFD2A
&c2 = 010BFD2B
&c3 = 010BFD29
&k  = 010BFD24
&f1 = 010BFD20
&f2 = 010BFD1C
&d1 = 010BFD14
&d2 = 010BFD0C
&c2 = 1
&c2 = -5
&c2 = 4
```

- 按变量定义顺序，由后往前连续分配空间；
- 空间最小分配单位：4个字节（8个字节）
 - 字符类型虽然只需要1个字节存储空间，但也被分配了12个
- Debug版本
- Release版本

例3：需要多大存储空间？

```
#include <stdio.h>
typedef struct
{
    char sex; // 'm', 'f'
    int age, height;
} PERSON;

void main()
{
    PERSON personA = {'m', 20, 180};
    PERSON personB = {'f', 16, 160};
    FILE file1;

    printf("&PersonA \t= %p\n",
           &personA);
    printf("&PersonB \t= %p\n",
           &personB);
    printf("PERSON size \t= %d\n",
           sizeof(PERSON) );
    printf("&file1 \t= %p\n", &file1);
    printf("FILE size \t= %d\n",
           sizeof(FILE) );
}
```

- **PERSON**需要多大的存储空间？
实际分配多大存储空间？
- 猜猜看，**FILE**结构需要多大存储空间？

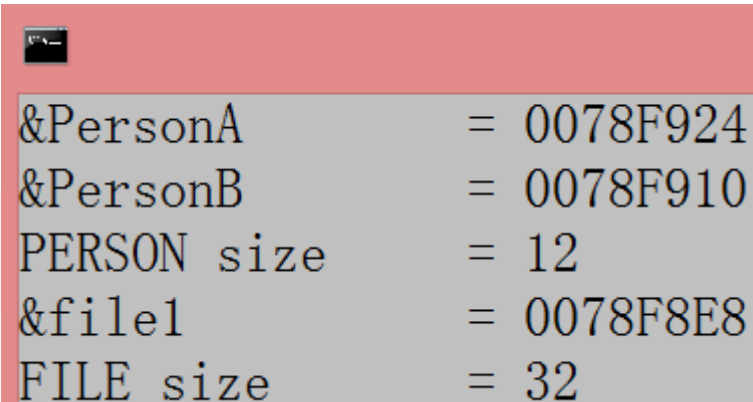


需要多大存储空间？

```
#include <stdio.h>
typedef struct
{
    char sex; // 'm', 'f'
    int age, height;
} PERSON;

void main()
{
    PERSON personA = {'m', 20, 180};
    PERSON personB = {'f', 16, 160};
    FILE file1;

    printf("&PersonA \t= %p\n",
           &personA);
    printf("&PersonB \t= %p\n",
           &personB);
    printf("PERSON size \t= %d\n",
           sizeof(PERSON) );
    printf("&file1 \t= %p\n", &file1);
    printf("FILE size \t= %d\n",
           sizeof(FILE) );
}
```



&PersonA	= 0078F924
&PersonB	= 0078F910
PERSON size	= 12
&file1	= 0078F8E8
FILE size	= 32

- PERSON:
 - 需要：9个字节
 - 实际分配：12
- FILE： 32个字节



2.赋值与初始化

- 操作符 =
等号两侧操作数类型要一致

- 赋值

```
int *p1, k, *p2;
```

```
p1 = &k;
```

```
p2 = p1;
```

- 初始化

```
int k;
```

```
int *p1 = NULL;
```

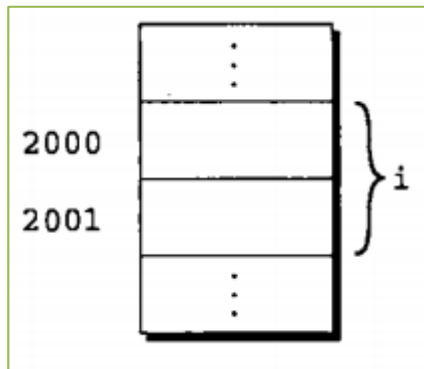
```
int *p2 = &k;
```

```
void main()  
{  
    int    k = 9;  
    char iAmH = 'h';  
  
    int *p1 = &k;  
    char *p2 = &iAmH;  
    char *p3 = p1; //不对!  
  
    printf("p1 = %p\n", p1);  
    printf("p2 = %p\n", p2);  
    printf("p3 = %p\n", p3);  
}
```

- 指针类型不同，不能相互赋值



3. 间接寻址



变量 \leftrightarrow 一小块内存

- 寻址：
 - 找到与变量对应的内存位置，以便进行相应存取操作
- 直接寻址
 - 通过变量名，找到这一小块内存
- 例：将10存到k代表空间
 - `int k;`
 - `k = 10;`



间接寻址

- 间接寻址
 - 通过运用 *****运算符 操作指针变量，找到这一小块内存
- 例： 将**10**存到**q**指向空间
 - `int k; int *p = &k;`
 - `int *q = p;`
 - `*q = 10;`
- 问题： `int *q=p; *q=10;` 这两个语句中*号作用有何不同？

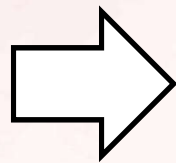


例：用 * 间接寻址

```
void main() {  
    int i;  
    int *p = &i;  
  
    // 1 存的操作  
    i = 10;  
    *p = 20;  
  
    // 2 取的操作  
    printf(" i = %d\n", i);  
    printf(" i = %d\n", *p);  
}
```

- 1、间接寻址
 - 通过p找到i，需要两步：先找到p，再根据p的内容来找到i
 - 所以是间接寻址
- 2、 i = 20

*p: 通过p中存储地址找到对应变量的地址



例：指针必须正常初始化

```
#include <stdio.h>

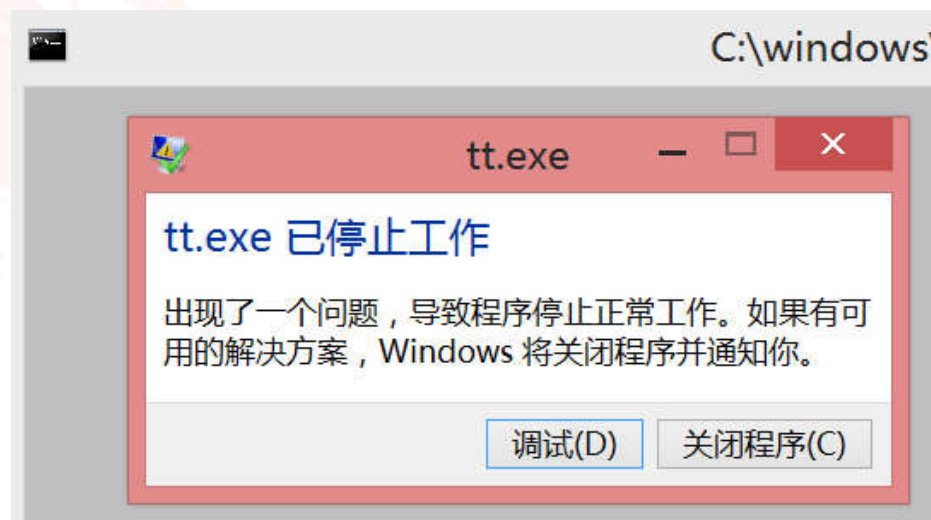
void main() {
    int i = 10, j = 20;
    int *p = NULL ;

    int k = i * j;
    *p = 30;
    int z = k * *p;

    printf(" k = %d\n", k);
    printf(" *p = %d\n", *p);
    printf(" z = %d\n", z);
}
```

• 区别：

- 间接寻址运算符：
单目运算符，后面
必须跟**指针变量**
- 乘法运算符：双目
运算符





11.4 应用

1. 应用一：指针变量作为函数参数

- 作用：在被调函数中修改主调函数中变量的值

例1：scanf

```
int n;  
printf("您要从1累加到几? ");  
scanf("%d", &n);
```

- scanf第2个参数传的是n的地址
- 为什么要这样做？
 - 需要在scanf中对n的值进行修改
 - 即：在被调函数中修改主调函数中变量的值



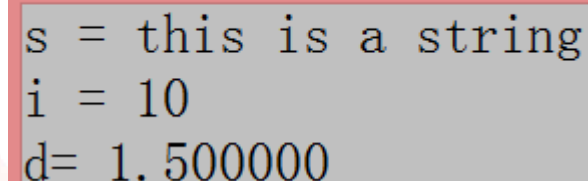
例：文件指针作为参数

```
#include <stdio.h>
#include <process.h>

void main( void )
{
    FILE *fp;
    int    i = 10;
    double d = 1.5;
    char   s[] = "this is a string";
    char   c = '\n';

    fp = fopen( "d:\\fprintf.out", "w" );
    fprintf( fp, "s = %s%c", s, c );
    fprintf( fp, "i = %d\n", i );
    fprintf( fp, "d= %f\n", d );
    fclose( fp );

    system( "type d:\\fprintf.out" );
}
```



```
s = this is a string
i = 10
d= 1.500000
```

- 作用
 - 在被调函数中修改主调函数中变量的值
 - 传递参数时，需要的内存空间更小

思考：如何让函数返回多个值？

- 一个函数只能有**0**个或**1**个返回值
 - 例： `int add(int a, int b);`
- 如果需要同时返回多个结果，该如何办？
 - 例：求一个实数的整数部分和小数部分
 - $x = k + y$
- `void decompose(double x, int *k, double *y);`



```
void decompose(double x, int *k, double *y) {  
    *k = (int)x;  
    *y = x - *k;  
}
```

```
void main() {  
    double x = 10.3, y;  
    int k;  
    decompose(x, &k, &y);  
    printf("x = %8.3f, k=%6d, y=%8.3f",x,k,y);  
}
```



```
int m=1,n=2,*p=&m,*q=&n,*r;  
r=p;p=q;q=r;  
printf("%d,%d,%d,%d\n",m,n,*p,*q);  
输出是?
```

- ☐ A 1,2,1,2
- ☒ B 1,2,2,1
- ☐ C 2,1,2,1
- ☐ D 2,1,1,2

提交



2. 应用二：指针变量作为函数返回值

- 使用方法：与其他类型变量作为返回值一样

例1：文件指针作为返回值

```
FILE *fopen( const char *filename, const char  
*mode );
```



例：辨析

```
#include <stdio.h>
#include <assert.h>

#define DEBUG

int * accumulate(int nFrom, int nTo)
{
    int sum = 0;
    for(int i=nFrom; i<= nTo; i++)
        sum += i;
    return &sum;
}

void main( void )
{
    #ifdef DEBUG
        int *p = accumulate(1, 100);
        assert( *p == 5050 );
    #endif
}
```

- 能通过编译吗？
 - 有**warning**
- 运行正确吗？
 - **yes**

• 通过指针，主调函数间接使用了被调函数中局部变量

- 但是，被调函数中局部变量生存期短
- 当被调函数执行结束时，为局部变量分配空间即被操作系统收回
- 即指针指向的是已被收回的空间。可能会引发莫名其妙的错误！

结论： 不应使用指针返回局部变量地址

小心！ 不要乱指哦



交换函数swap的实现

```
#include <stdio.h>
#include <assert.h>

#define DEBUG

void swap( char *p1, char* p2 )
{
    char *cTemp=p1;
    p1 = p2;
    p2 = cTemp;
}

int main()
{
#ifdef DEBUG
    char c='c', d='d';
    swap( &c, &d );
    assert( c=='d' && d=='c' );
#endif
    return 0;
}
```

对吗？

- 这样写对吗？
 - 不对
 - 交换的是p1, p2
 - 而不是p1, p2所指向的变量
 - 这样操作：对c, d还是没有影响

交换函数swap的实现

正解

```
#include <stdio.h>
#include <assert.h>

#define DEBUG

void swap( char *p1, char* p2 )
{
    char cTemp=*p1;
    *p1 = *p2;
    *p2 = cTemp;
}

int main()
{
    #ifdef DEBUG
        char c='c', d='d';
        swap( &c, &d );
        assert( c=='d' && d=='c' );
    #endif
    return 0;
}
```



小结

小结1：指针变量定义与基本操作

- 指针变量定义
- 指针基本操作
&, =, *





小结

小结2:

指针应用：指针作为参数与返回值

- 在被调函数中可以操作主调函数中变量
 - 有时必须这么做
 - 且：只需单向传递数据
 - 使用方便，效率高
 - 且：传递指针往往比传递数据需要的内存小
- 不应通过指针来返回被调函数中局部变量
 - 若一意孤行，





*指针的妙用

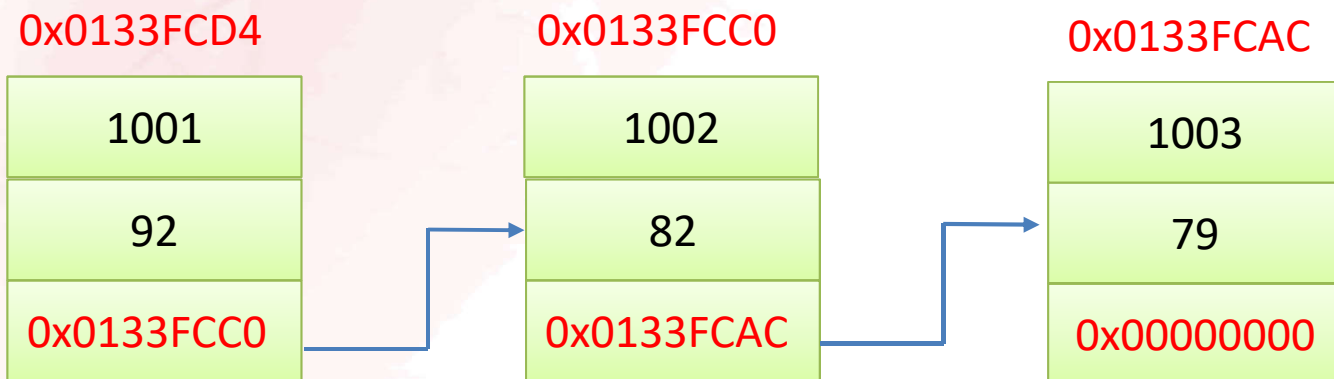
• 链表（教材9.4节）

```
struct student
{
    int num;
    int score;
    struct student* next;
};
```

```
int main()
{
    struct student stu1={1001, 92}, stu2={1002, 82}, stu3={1003, 79};
    struct student *pHead;

    pHead=&stu1;
    stu1.next=&stu2;
    stu2.next=&stu3;
    stu3.next=NULL;

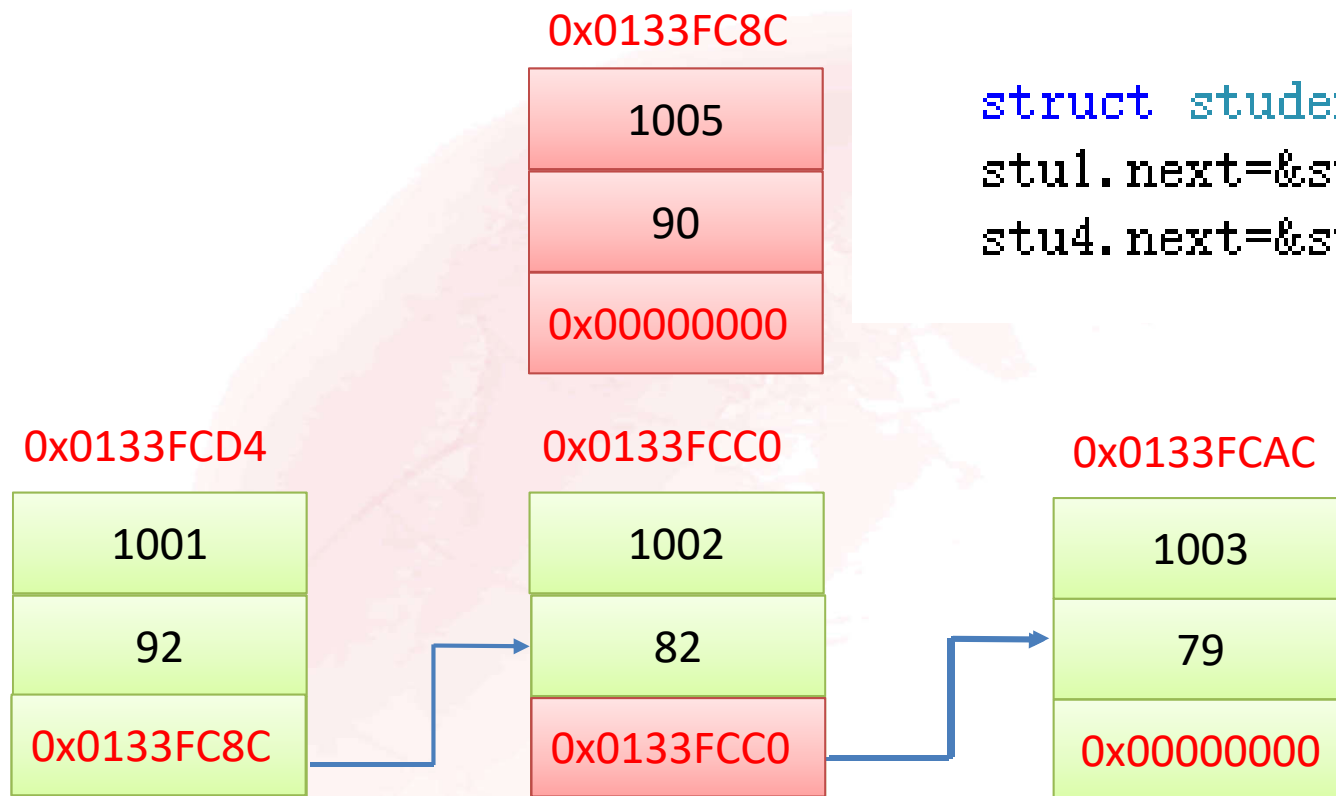
    return 0;
}
```





*指针的妙用

- 插入新节点



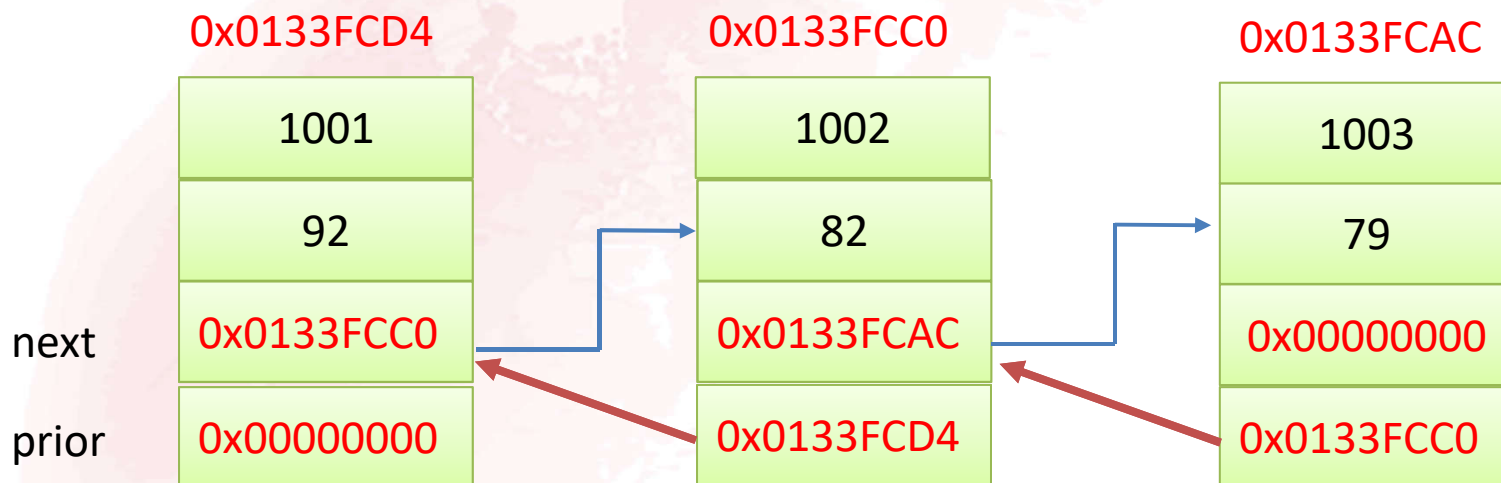
```
struct student stu4={1005, 90};  
stu1.next=&stu4;  
stu4.next=&stu2;
```



*指针的妙用

- 双向链表

```
.....  
struct student  
{  
    int num;  
    int score;  
    struct student* next;  
    struct student* prior;  
};
```



方便双向遍历





几点建议

- 读书读一本，反复看。不要看很多书，而是一本书看很多遍；
- 做题，反复做。不要做很多习题，而是一道题做很多遍。

贪多嚼不烂

此前各周的题目，请自己重新做！




```
void fun(int *a, int * b)
{
    int *w;*w=*a; *a=*a+*a;*a=*b; *b=*w;
}
int main()
{
    int x=9,y=5,*px=&x,*py=&y;
    fun(px,py);printf("%d,%d\n",x,y);
}输出为
```

- ☐ A 18,5
 ☒ B 报错
 ☐ C 5,9
 ☐ D 5,18

提交

