计算机程序设计基础（1）

13 结构体

清华大学电子工程系

杨昉

E-mail: fangyang@tsinghua.edu.cn

课程回顾

- **动态内存的申请**
  - malloc(),calloc(),realloc(),free()函数的使用方法和注意事项
  - 动态一维数组、二维数组的生成及作为函数参数
- **字符串与指针**
  - 字符串指针与字符数组的异同点，字符串指针作为函数参数
  - 紧凑型字符串数组
- **函数与指针**
  - 用函数指针变量调用函数，函数指针数组
  - 函数指针变量作为函数参数，返回指针值的函数

课程回顾：动态内存

函数名	用法	说明
malloc()	(类型*) malloc(申请的字节数)	使用时需对申请后的内存进行强制类型转换，赋值失败返回NULL
calloc()	(类型 *) calloc(元素个数n，类型占据字节数size)	在内存的动态存储区中分配n个长度为size的连续空间，初始化为0，赋值失败返回NULL。能够申请比malloc()更大的动态内存空间
realloc()	指针名 = (数据类型*) realloc(指针名，新的内存长度)	在原先申请内存块的基础上，再重新申请一块更长的（或更短）的内存块，以实现内存块的动态增长，同时保留原数据
free()	free(指针名)	释放动态分配的内存，防止内存泄漏

课程回顾：字符串与指针

知识点	示例	说明
字符串的表示：字符指针与数组	char s[20]=“Hello world”； char *s = “Hello world”；	二者初始化方法相仿，都能够用来作为**函数参数**
字符指针自身可以修改	char *s = “Hello world”； s= s+4;	修改之后的s值为“o world”
字符指针所指向的字符串不能修改	char *s = “Hello world”； s[3]=‘f’； ✗	字符指针指向的字符串是**常量字符串**
字符数组名无法修改	char s[20]=“Hello world”； s = s+4; ✗	数组名是**常量指针**，无法进行修改
字符数组中的值可以进行修改	char s[20] = “Hello world”； s[1]= ‘a’；	修改后为“Hallo world”
不能通过输入函数让字符指针变量指向一个字符串	各种错误示例见对应章节课件	根本原因：对于通过键盘输入的字符串，系统**不分配存储空间**

课程回顾：函数指针

知识点	示例	说明
指向函数的指针定义	类型标识符 (*指针变量名)( );	最后的( )不能省略
函数指针的使用	(*函数指针变量名)(实参表) 或　函数指针变量名(实参表)	对函数指针变量运算没有意义的
函数指针数组	int (*p[5]) ( );	使函数指针数组每个元素分别指向不同的函数,减少代码冗余
函数指针变量作为函数参数	int root(double (*f)()){…}	函数指针指向不同函数入口地址时，在函数中可调用不同的函数
返回指针值的函数	int* f(…){…}	函数可以返回任何类型的指针值，指针函数通过函数名返回指针值
main函数的形参	main(int argc, char *argv[]){…}	argv是一个字符型指针数组，指向不同的字符串；argc的值是argv指向的字符串个数

第13讲课程目标

13.1 结构体变量与数组

- 结构体变量
- 结构体数组

13.2 结构体与指针

- 结构体指针变量
- 指针作为函数参数

13.3 链表

- 链表的概念
- 链表基本运算

13.4 联合体与枚举类型

- 联合体
- 枚举类型
- 自定义类型名

13.1 结构体变量与数组

- 结构体变量
  - ✓ 定义与引用
  - ✓ 结构体嵌套
  - ✓ 结构体变量初始化
  - ✓ 结构体与函数
- 结构体数组
  - ✓ 定义与引用
  - ✓ 作为函数参数

结构体变量与数组：结构体类型

- 回顾基本类型变量
  - 基本数据类型，C语言都已经定义，用户只要使用系统所定义的类型名(如int, double, char等)即可直接定义这些类型的变量
  - 数组是相同数据类型的集合，如果需要定义基本数据类型的数组，也只需要用这些基本数据类型名直接定义
- **结构体类型**
  - 是一种复合类型，多种数据作为整体处理。各种不同的结构体中的成员个数以及各成员的数据类型可能是各不相同的。

结构体变量与数组：结构体类型

结构体类型

- C编译系统提供了用户自己定义结构体类型的机制，以便用户用复合类型来描述现实世界中的复杂对象
- 具体的结构体类型由用户根据实际的需要自己定义
- 结构体类型名定义的是**类型名**，而**不是变量名**
  - 类似整型的类型名为int，双精度实型的类型名为double，字符型的类型名为char
  - 只不过这些类型是C编译系统定义的基本数据类型，而结构体是用户自己定义的

生活中的结构体

10寸披萨

10寸方型煎饼果子披萨

结构体变量与数组：结构体变量的定义


- 定义**结构体类型**
  - 形式

```
struct 结构体类型名
{成员表};
```

  - struct 是用于定义具体结构体类型的**关键字**
  - 在"成员表"中定义该类型中的**成员**及**数据类型**
- 例13-1：定义"日期"的结构体类型

```
1  struct date {
2    int year;
3    int month;
4    int day;
5  };
```

结构体变量与数组：结构体变量的定义


- 定义**结构体类型变量**
  - 原则：将定义结构体**类型**与定义结构体类型**变量**分开说明
  - 形式

```
struct 结构体类型名 变量表;
```

- 例13-2：定义结构体类型的变量

```
struct date birthday, x, y;
```

  - 需要说明，在定义结构体类型变量时不能只使用**结构体类型名**，而应使用结构体类型的全称，即**struct关键字不能省略**
  - 错误的写法有 date birthday, x, y;

结构体变量与数组：结构体变量的定义

- 定义**结构体类型变量**
  - 另一种定义形式

```
struct 结构体类型名
{成员表} 变量表;
```

  - 在定义**结构体类型**的同时又定义了**结构体类型变量**后，在程序中仍然可以定义此结构体类型的**其他变量**
- 定义**无名结构体类型**及变量
  - 直接定义结构体类型变量而**没有类型名**
  - 形式

```
struct
{成员表} 变量表;
```

结构体变量与数组：结构体变量作用域

- 例13-3：定义无名结构体

```
struct {int num; char name[10]; int age} a, b, st;
```

- 结构体类型的**作用域**
  - 需要指出的是，如果在**函数体外**定义了一个结构体类型，则从定义位置开始到整个程序文件结束之前的**所有函数中**均可用该结构体类型定义该类型的变量
  - 但在**函数体内**所定义的结构体类型，**只能在该函数体内**能用来定义该类型的变量。即结构体类型的定义与普通变量定义的**作用域是相同的**，有全局和局部之分

结构体变量与数组：结构体变量引用

- 结构体类型变量的**引用**
  - 形式　　结构体变量名.成员名
  - "."为结构体成员运算符，它的**优先级是C语言最高的**
  - 结构体变量中的每个成员**与普通变量一样**，可以**进行各种运算**
  - 如果结构体变量中的某成员是一个数组，则在为该成员赋值时，与普通数组一样，必须对该成员中的**逐个元素赋值**
- 例13-4：例13-3可以按照如下方式引用st变量

```
st.num=115; st.name[0]='M'; st.name[1]='a';
st.age=19;
```

结构体变量与数组：结构体变量初始化

- 结构体类型变量的**初始化**
  - 在定义结构体类型变量的同时也可以对结构体类型变量**赋初值**，其原理**与普通变量的初始化一样**
  - 形式：

```
struct 结构体名 变量名={变量表值};
```

  - 一个结构体类型变量往往包括多个成员，因此，在结构体类型变量初始化时，要用一对**花括号将所有的成员数据括起来**

```
1  struct student {
2    int num; char name[10];
3    char gender; int age; float score;
4  };
5  struct student st={101,"Zhang",'M',19,89.0};
```

结构体变量与数组：结构体变量

- 例13-5：定义“日期”结构体类型，初始化并引用内部成员

```
1  #include <stdio.h>                    //包含头文件"stdio.h"
2  struct date {
3    int year, month, day;
4  };
5  void main() {                         //定义main函数，这是程序的主体
6    struct date a={2017,12,8}; //定义date型变量a
7    printf("%02d/%02d/%d\n",a.month, a.day, a.year);
8  }
```

```
12/08/2017
请按任意键继续……
```

结构体变量与数组：结构体类型的嵌套

- 结构体类型的**嵌套**
  - 结构体类型的定义可以嵌套
- 例13-6：定义一个结构体"日期"
  - 其中含有一个嵌套的结构体"时间"
  - 在这个定义中，结构体"日期"类型date中有一个成员又属于结构体"时间"类型time

```
1  struct time {
2    int hour;
3    int minute;
4    int second;
5  };
```

嵌套

```
1  struct date {
2    int year;
3    int month;
4    int day;
5    struct time t;
6  };
```

结构体变量与数组：结构体类型的嵌套

- 例13-7：例13-6如果定义"日期"类型变量d，则可以按照如下方式引用d的内部成员
  - 注意"."的运算顺序为从左至右

```
1  struct date d;//定义struct型变量d
2  d.year          //结构体"日期"类型(即date)变量d的成员"年"
3  d.month         //结构体"日期"类型(即date)变量d的成员"月"
4  d.day           //结构体"日期"类型(即date)变量d的成员"日"
5  d.t.hour        //结构体"日期"类型(即date)变量d的成员结构体类型"时间"中的成员"时"
6  d.t.minute      //结构体"日期"类型(即date)变量d的成员结构体类型"时间"中的成员"分"
7  d.t.second      //结构体"日期"类型(即date)变量d的成员结构体类型"时间"中的成员"秒"
```

结构体变量与数组：结构体类型的嵌套

- 例13-8：定义学生结构体

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  struct student {
3    int num; char name[10]; char gender; int age; float score;
4  };
5  struct time {
6    int hour, minute, second;
7  };
8  struct date {
9    int year, month, day; struct time t;
10 };
11 void main() {
12   struct student st={101,"Zhang",'M',19,89.0}; //初始化student型变量st
13   struct date xy={2017,12,8,{10,34,55}};       //初始化date型变量xy
14   printf("st=%4d%6s%2c%3d%6.2f\n", st.num, st.name, st.gender, st.age, st.score);
15   printf("date=%d/%02d/%02d/%d:%d:%d\n", xy.year, xy.month, xy.day, xy.t.hour, xy.t.minute, xy.t.second);
16 }
```

```
st= 101 Zhang M 19 89.00
date=2017/12/08/10:34:55
请按任意键继续……
```

结构体变量与数组：结构体类型的嵌套


- 结构体嵌套类型变量的**初始化**
  - 若是结构体的嵌套类型，成员变量的初始化有两种方法
    - 内层结构体中的所有成员数据可以用一对花括号括起来
    - 也可以不用花括号括起来而直接赋值
- 例13-9：例13-8中对xy结构体变量的初始化等价于

```
1  struct date {
2    int year, month, day; struct time t;
3  };
4  struct date xy={2017,12,8,10,34,55};
```

结构体变量与数组：结构体与函数

- 结构体与函数
  - 在结构体类型变量中的**成员作为函数参数**的情况下：被调用函数中的形参是一般变量，而调用函数中的**实参是结构体类型变量中的一个成员**,但要求它们的类型应一致
  - 在结构体类型的**变量作为函数参数**的情况下：被调用函数中的形参是结构体类型的变量,调用函数中的**实参也是结构体类型的变量**,但要求它们属于同一个结构体类型
  - 结构体(或成员)做实参,函数**不能改变结构体(或成员)的值**
  - 函数**返回值**也可以是结构体类型

结构体变量与数组：结构体与函数

- 例13-10：结构体成员作为函数参数不改变成员的值

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  #include <string.h>   //包含头文件"string.h"
3  struct student {
4    int num; char name[10]; char gender; int age; float score;
5  };
6  void change(float t){
7    printf("score=%6.2f\n", t);
8    t=95.0;
9    printf("score=%6.2f\n", t);
10 };
11 void main(){
12   struct student st={101,"Zhang",'M',19,89.0}; //初始化student型变量st
13   printf("st=%4d%6s%2c%3d%6.2f\n",st.num, st.name, st.gender, st.age, st.score);
14   change(st.score);
15   printf("st=%4d%6s%2c%3d%6.2f\n",st.num, st.name, st.gender, st.age, st.score);
16 }
```

```
st= 101 Zhang M 19 89.00
score= 89.00
score= 95.00
st= 101 Zhang M 19 89.00
请按任意键继续……
```

结构体变量与数组：结构体与函数

- 例13-11：结构体作为函数参数不改变结构体的值

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  #include <string.h>   //包含头文件"string.h"
3  struct student {
4    int num; char name[10]; char gender; int age; float score;
5  };
6  void change(struct student t){
7    printf("t= %4d%6s%2c%3d%6.2f\n",t.num, t.name, t.gender, t.age, t.score);
8    t.score=95.0; strcpy(t.name, "Huang");
9    printf("t= %4d%6s%2c%3d%6.2f\n",t.num, t.name, t.gender, t.age, t.score);
10 };
11 void main(){
12   struct student st={101,"Zhang",'M',19,89.0}; //初始化student型变量st
13   printf("st=%4d%6s%2c%3d%6.2f\n",st.num, st.name, st.gender, st.age, st.score);
14   change(st);
15   printf("st=%4d%6s%2c%3d%6.2f\n",st.num, st.name, st.gender, st.age, st.score);
16 }
```

```
st= 101 Zhang M 19 89.00
t=  101 Zhang M 19 95.00
t=  101 Huang M 19 95.00
st= 101 Zhang M 19 89.00
请按任意键继续……
```

课堂练习1

- 结构体作为函数返回值

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  struct date {
3    int year, month, day;
4  };
5  struct date f() {
6    struct date t={2017,12,8};
7    return t;
8  };
9  void main() {
10   struct date xy;     //定义date型变量xy
11   xy = f();
12   printf("date=%d/%02d/%02d",xy.year,xy.month,xy.day);
13 }
```

```
date=2017/12/08
请按任意键继续……
```

请辨析上面三个例子

结构体变量与数组：结构体数组

- **结构体数组**定义与引用
  - 与整型数组、实型数组、字符型数组一样，在程序中也可以**定义结构体类型的数组**，并且同一个结构体数组中的元素应为**同一种结构体类型**
  - 引用方式与**普通的数组和结构体一样**

- 例13-12：给定学生成绩登记表。利用结构体数组计算课程成绩的平均成绩，最后输出该学生成绩登记表

结构体变量与数组：结构体数组

- 例13-12：学生成绩登记表

```
101 Zhang M 19 95.00 64.00 79.50
102 Wang  F 18 92.00 97.00 94.50
103 Zhao  M 19 85.00 78.00 81.50
请按任意键继续……
```

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  #define STU struct student
3  STU {
4    int num; char name[10]; char gender; int age; float score[3];
5  };
6  void main() {
7    int k;
8    STU stu[3]={ {101,"Zhang",'M',19,95.0,64.0}, {102,"Wang",'F',18,92.0,97.0},
9                 {103,"Zhao",'M',19,85.0,78.0} };
10   for (k=0; k<=2; k++){
11     stu[k].score[2]=(stu[k].score[0] + stu[k].score[1])/2;
12     printf("%-4d%-6s%-2c%-3d%-6.2f%-6.2f%-6.2f\n",
13             stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
14             stu[k].score[0], stu[k].score[1], stu[k].score[2]);
15   }
16 }
```

结构体变量与数组：结构体数组

- 结构体数组作为**函数参数**
  - 结构体类型数组也能作为函数参数，并且**形参与实参结合**的方式与基本数据类型的**数组完全一样**
  - 结构体类型形参数组与结构体类型实参数组是同一个存储空间，因此**传首地址**

- 例13-13：将例13-8中的平均值计算功能用函数p()实现，并且将**结构体数组作为函数参数**传入函数中

结构体变量与数组：结构体数组

- 例13-13：结构体数组举例

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  #define STU struct student
3  STU {
4    int num; char name[10]; char gender; int age; float score[3];
5  };
6  void p(STU t[], int n) {
7    int k;
8    for (k=0; k<n; k++)
9      t[k].score[2]=(t[k].score[0] + t[k].score[1])/2;
10 }
11 void main() {
12   int k;
13   STU stu[3]={{101,"Zhang",'M',19,95.0,64.0},{102,"Wang",'F',18,92.0,97.0},
14               {103,"Zhao",'M',19,85.0,78.0}};
```

结构体变量与数组：结构体数组

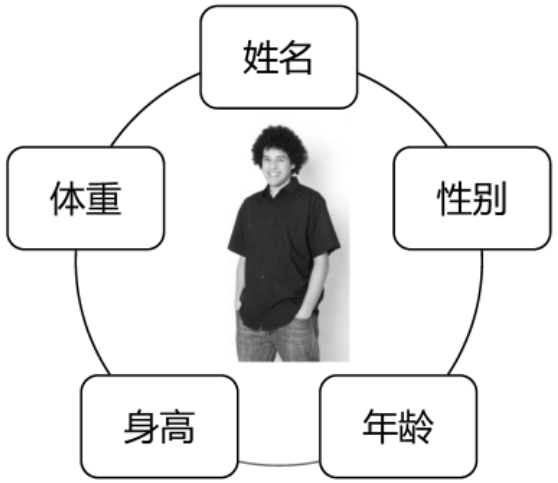
```
15    for (k=0; k<3; k++) {
16      printf("%-4d%-6s%-2c%-3d%-6.2f%-6.2f%-6.2f\n",
17             stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
18             stu[k].score[0], stu[k].score[1], stu[k].score[2]);
19    }
20    printf("--------------------------------\n");
21    p(stu,3);
22    for (k=0; k<3; k++) {
23      printf("%-4d%-6s%-2c%-3d%-6.2f%-6.2f%-6.2f\n",
24             stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
25             stu[k].score[0], stu[k].score[1], stu[k].score[2]);
26    }
27  }
```

```
101 Zhang M 19 95.00 64.00 0.00
102 Wang  F 18 92.00 97.00 0.00
103 Zhao  M 19 85.00 78.00 0.00
--------------------------------
101 Zhang M 19 95.00 64.00 79.50
102 Wang  F 18 92.00 97.00 94.50
103 Zhao  M 19 85.00 78.00 81.50
请按任意键继续……
```

归类思维——结构体

学会归纳数据和函数，把“大问题”变“小问题”

- 对于某类数据，及其可以执行的函数，可以**用结构体的形式归类**到一起



打包

```
struct Human
{
    string  m_strName;    // 姓名
    bool    m_bMale;      // 性别
    int     m_nAge;       // 年龄
    int     m_nHeight;    // 身高
    float   m_fWeight;    // 体重
};
```

- 可以更清晰地辨别不同类数据和它们的执行方式，**有利于组织编写更大规模**的程序

13.2 结构体与指针

- 结构体指针变量
  - ✓ 结构体指针变量定义
  - ✓ 结构体指针变量引用
- 指针作为函数参数

结构体与指针：结构体指针变量

- 定义与引用
  - 结构体类型的指针变量指向结构体类型变量或数组或数组元素的**起始地址**
  - 引用方式和**普通的指针一样**
- 例13-14：结构体指针定义与引用

```
1  struct student {
2    int num;
3    char name[10];
4    char gender;
5    int age;
6    float score;
7  };
8  struct student st1,st2,st[10],*p
```

令p=&st1

```
1  st1.num
2  (*p).num
3  p->num
4  p[0].num
```

令p=st

```
1  st[0].num
2  (*p).num
3  p->num
4  p[0].num
```

结构体与指针：结构体指针变量

- 定义与引用
  - 当结构体类型的指针变量p**指向一个结构体类型变量**a后,下列四种表示是等价的:

```
1  a.成员
2  (*p).成员
3  p->成员
4  p[0].成员
```

  - 其中，"->"被称为**指向运算符**
  - 必须注意,当p定义为指向结构体类型数据后,它不能指向某一成员：如p=&st1.num; 是**错误的**,因为这企图让结构体指针变量指向结构体变量st1中类型为int的成员num

结构体与指针：指针作为函数参数

- 结构体类型指针作为函数参数
  - 结构体类型指针可以指向**结构体类型的变量**
  - 当形参是结构体类型指针变量时,实参也可以是**结构体类型指针**(即地址)
  - **结构体类型形参指针**与**结构体类型实参指针**指向的是**同一个存储空间**
  - 调用函数改变了**结构体类型形参指针**所指向的地址中的值，实际上也改变了**结构体类型实参指针**所指向地址中的值

结构体与指针：指针作为函数参数


- 例13-15：结构体指针作为函数参数

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  #include <string.h>   //包含头文件"string.h"
3  struct student {
4    int num; char name[10]; char gender; int age; float score;
5  };
6  void change(struct student *t) {
7    printf(" t=%4d%6s%2c%3d%6.2f\n", t->num, t->name, t->gender, t->age, t->score);
8    t->score=95.0;
9    printf(" t=%4d%6s%2c%3d%6.2f\n", t->num, t->name, t->gender, t->age, t->score);
10 }
11 void main() {
12   struct student st={101,"Zhang",'M',19,89.0}; //定义student型变量st
13   printf("st=%4d%6s%2c%3d%6.2f\n",st.num, st.name, st.gender, st.age, st.score);
14   change(&st);
15   printf("st=%4d%6s%2c%3d%6.2f\n",st.num, st.name, st.gender, st.age, st.score);
16 }
```

```
st= 101 Zhang M 19 89.00
 t= 101 Zhang M 19 89.00
 t= 101 Zhang M 19 95.00
st= 101 Zhang M 19 95.00
请按任意键继续……
```

结构体与指针：指针作为函数参数

- 结构体类型指针作为函数参数
  - 结构体类型指针也可以指向**数组**或**数组元素**
  - 当形参是**结构体类型指针变量**时，实参也可以是**结构体类型数组名**或**数组元素的地址**
  - 在用**结构体类型数组名**作函数参数时，实际上也可以用**指向结构体类型数组**或**数组元素的指针**作为函数的参数

结构体与指针：指针作为函数参数

- 与标准数据类型的数组与指针一样，在结构体类型数组指针作函数参数时，也可以有以下四种情况
  - 实参与形参都用**结构体类型数组名**
  - 实参用**结构体类型数组名**，形参用**结构体类型指针变量**
  - 实参与形参都用**结构体类型指针变量**
  - 实参用**结构体类型指针变量**，形参用**结构体类型数组名**

结构体与指针：指针作为函数参数

- 例13-16：用结构体指针重写例13-13

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  #define STU struct student
3  STU {
4    int num; char name[10]; char gender; int age; float score[3];
5  };
6  void p(STU *t, int n) {
7    int k;
8    for (k=0; k<=n-1; k++)
9      t[k].score[2]=(t[k].score[0] + t[k].score[1])/2;
10 }
11 void main() {          //定义main函数，这是程序的主体
12   int k;
13   STU stu[3]={ {101,"Zhang",'M',19,95.0,64.0}, {102,"Wang",'F',18,92.0,97.0},
14                {103,"Zhao",'M',19,85.0,78.0} };
```

结构体与指针：指针作为函数参数

```
15      for (k=0; k<=2; k++){
16        printf("%-4d%-6s%-2c%-3d%-6.2f%-6.2f%-6.2f\n",
17              stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
18              stu[k].score[0], stu[k].score[1], stu[k].score[2]);
19    }
20    printf("--------------------------------\n");
21    int n = 3;
22    p(stu, n);
23    for (k=0; k<=2; k++){
24      printf("%-4d%-6s%-2c%-3d%-6.2f%-6.2f%-6.2f\n",
25            stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
26            stu[k].score[0], stu[k].score[1], stu[k].score[2]);
27    }
28  }
```

```
101 Zhang M 19 95.00 64.00 0.00
102 Wang  F 18 92.00 97.00 0.00
103 Zhao  M 19 85.00 78.00 0.00
--------------------------------
101 Zhang M 19 95.00 64.00 79.50
102 Wang  F 18 92.00 97.00 94.50
103 Zhao  M 19 85.00 78.00 81.50
请按任意键继续……
```

13.3 链表

- 链表的概念
- 链表基本运算
  - ✓ 链表的查找
  - ✓ 链表的插入/删除
  - ✓ 打印链表
  - ✓ 逆转链表

链表：链表的概念

- 链表一般结构
  - 由多个存储节点构成，每个存储结点包含**数据域**和**指针域**，分别存放**数据元素**和**下一结点元素的地址**
  - 可以用如下示意图表示

  - 逻辑结构

链表：链表的概念

- 链表一般结构
  - HEAD称为头指针，当HEAD=NULL(或0)时称为空表
  - 链表中最后一个结点的指针域为NULL(或0)，表示链表终止
  - 在链表中,各数据结点的存储位置是不连续的,并且各结点在存储空间中的位置关系与逻辑关系也不一致
- C语言链表结构
  - 定义链表结点

```
1  struct 结构体名 {
2    数据成员表;
3    struct 结构体名 *指针变量名;
4  };
```

  - 动态内存分配：(struct 结构体名 *)malloc(存储区字节数);
  - 动态内存释放：free(p);表示释放由p指向的动态存储空间

链表：链表的概念


- 例13-17：读入一个正整数序列建立链表，以非正整数结束

```
1  #include <stdio.h>     //包含头文件"stdio.h"
2  #include <stdlib.h>    //包含头文件"stdlib.h"
3  struct node {          //定义节点类型
4    int data; struct node *next;
5  };
6  void main() {
7    int x;
8    struct node *head, *p, *q;
9    head=NULL;  q=NULL; //置链表头指针为空，q为链表最后一个结点
10   scanf("%d", &x);       //输入正整数
11   while(x>0){            //输入值大于0
12     p=(struct node *)malloc(sizeof(struct node)); //申请动态内存
13     if (p==NULL){
14       printf("can't get memory!\n"); exit(1);
15     }
16     p->data=x;  p->next=NULL;          //置当前结点的数据域为输入的正整数x
```

链表：链表的概念

```
17      if(head==NULL)                  //若链表还为空，则将头指针指向当前结点p
18        head=p;
19      else q->next=p;                 //若不为空，则当前节点置于链表最后
20      q=p;                            //更新链表的最后
21      scanf("%d", &x);                //再输入一个正整数
22    }
23    p=head;
24    while(p!=NULL){                   //从头开始遍历链表
25      printf("%d ",p->data);          //输出结点的数值
26      q=p;   p=p->next;               //删除当前结点，并且向后滑动一个结点
27      free(q);                        //释放删除结点空间
28    }
29    printf("\n");
30  }
```

```
10 12 3 4 -1
10 12 3 4
请按任意键继续……
```

链表：链表基本操作

- 对链表的操作有很多种，可以分为：
  - **查找**指定元素
  - **插入**新的元素
  - **删除**某些元素
  - 链表的**打印**
  - 链表的**逆转**
- 不管是对链表进行插入或删除
  - 首先要找到插入或删除的位置，需要对链表进行**扫描查找**
  - 在链表中寻找包含指定元素值的**前一个结点**

通过示意图更好地理解链表的操作

链表：链表基本操作

- 查找元素
  - 通过结构体内的**指针域数据**依次向后遍历链表，直到链表尾为止
- 例13-18：描述链表的查找功能

```
1  struct node {
2    ET data; struct node *next;          //ET为数据元素类型名,下同
3  };
4  //在头指针为head的非空链表中寻找包含元素x的前一个结点p（结点p作为函数值返回）
5  struct node *lookst(struct node * head, ET x) {
6    struct node *p;
7    p=head;
8    while( (p->next!=NULL) && (p->next->data!=x) ){
9      p = p->next;
10   }
11   return p;
12 }
```

链表：链表基本操作

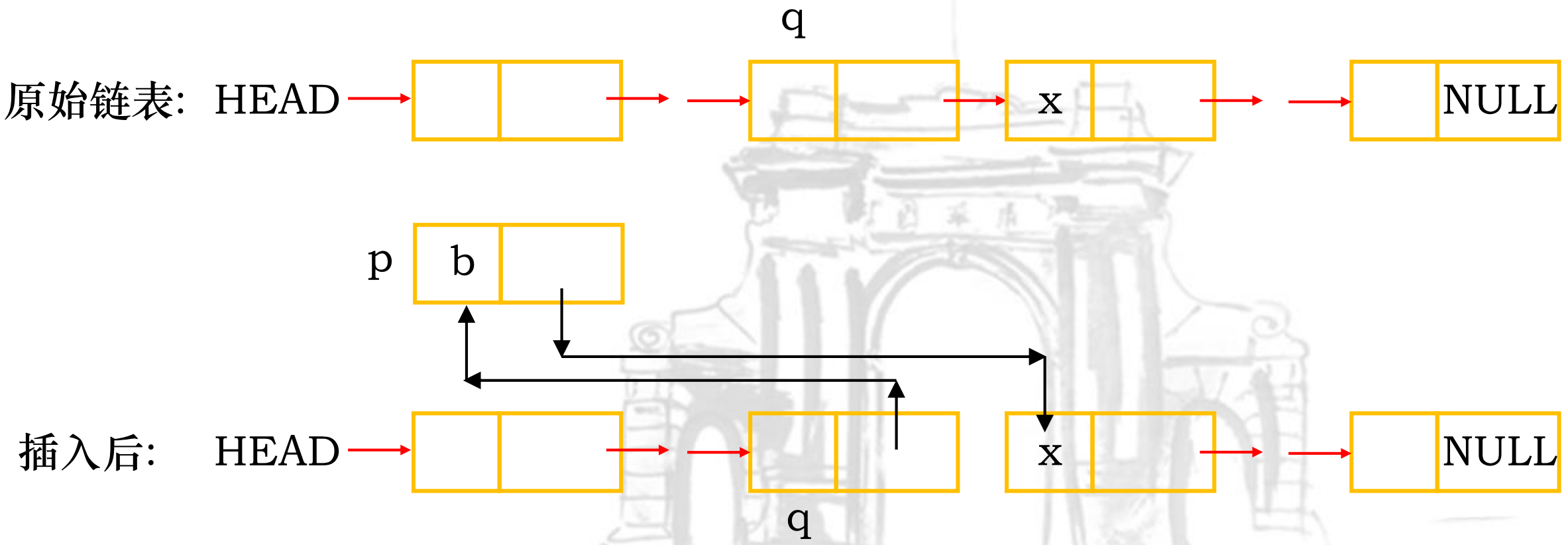
- 插入元素
  - 当找到包含指定元素的**前一个结点**后，就可以在该结点后**插入新结点**
  - 链表的插入**不需要移动数据**
- 链表插入元素举例
  - 在头指针为head的链表中包含元素x的结点之前插入新元素b
  - 用malloc( )函数申请取得新结点p，并置该结点的**数据域为b**，即令p->data=b

链表：链表基本操作

- 链表插入元素举例(续)
  - 在链表中寻找包含元素x的前一个结点，设该结点的存储地址为q
  - 最后将结点p插入到结点q之后。为了实现这一步,只要改变以下两个结点的指针域内容:
    - 使结点p指向包含元素x的结点（即结点q的后一结点），即令p->next = q->next
    - 使结点q的指针域内容改为指向结点p，即令q->next = p

链表：链表基本操作

- 链表插入元素示意图



链表：链表基本操作


- 例13-19：链表插入元素

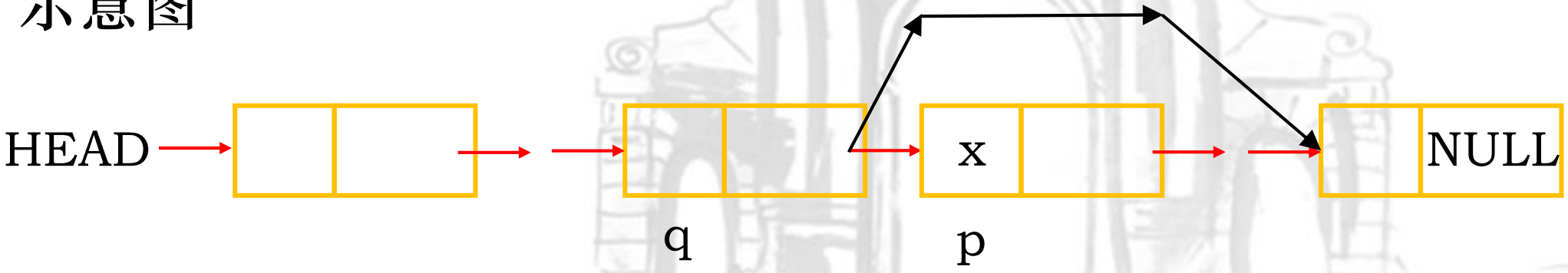
```
1  struct node {
2    ET data; struct node *next; };
3  void inslst(struct node **head, ET x, ET b) {  //需修改head值,因此传内存地址
4    struct node *p, *q;
5    p=(struct node *)malloc(sizeof(struct node)); //申请一个新结点
6    if ( p==NULL )
7      { printf("can't get memory!\n"); exit(1); }
8    p->data=b;                                    //置结点的数据域
9    if ( *head==NULL )                            //链表为空
10     { *head=p;  p->next=NULL;  return; }
11   if ( (*head)->data==x )                       //在第一个结点前插入
12     { p->next=*head;  *head=p;  return; }
13   q=lookst(*head,x);                            //寻找包含x的前一个结点q
14   p->next=q->next;  q->next=p;                  //结点p插入到q之后
15   return;
16 }
```

链表：链表基本操作

- 删除元素
  - 当找到包含指定元素的前一个结点后，就可以删除该结点后的一个结点
  - 链表的删除不需要移动数据
- 链表删除元素举例
  - 在头指针为head的链表中删除包含元素x的结点
  - 在链表中寻找包含元素x的前一个结点，设该结点地址为q。则包含元素x的结点地址p=q->next;

链表：链表基本操作

- 链表删除元素举例（续）
  - 将结点q后的结点p从链表中删除，即让结点q的指针指向包含元素x的结点p的后一个结点，令q->next=p->next;
  - 将包含元素x的结点p释放
  - 示意图



链表：链表基本操作

- 例13-20：链表删除元素

```
1  struct node {
2    ET data; struct node *next; };
3  void delst(struct node ** head, ET x) {
4    struct node *p,*q;
5    if ( *head==NULL )
6      { printf("This is a empty list!\n");  return; }
7    if ( (*head)->data==x ){                    //删除第一个结点
8        p=(*head)->next;  free(*head);          //释放删除结点
9        *head=p;  return;
10   }
11   q=lookst(*head,x);                          //寻找包含x的前一个结点q
12   if ( q->next==NULL )                         //没有包含元素x的结点
13     { printf("No this node in the list!\n");  return; }
14   p=q->next;  q->next=p->next;  free(p);      //删除结点p，释放删除结点p
15   return;
16 }
```

链表：链表基本操作

- 例13-21：链表打印
  - 遍历链表，并依次打印各结点的数据
- 从链表的第一个结点打印各结点的元素值

```
1  void printlst(struct node *head) {
2    struct node *p=head;
3    //从链表的第一个结点开始打印
4    while ( p!=NULL ){
5      printf("%d",p->data);
6      p=p->next;
7    }
8    printf("\n");
9  }
```

实现1

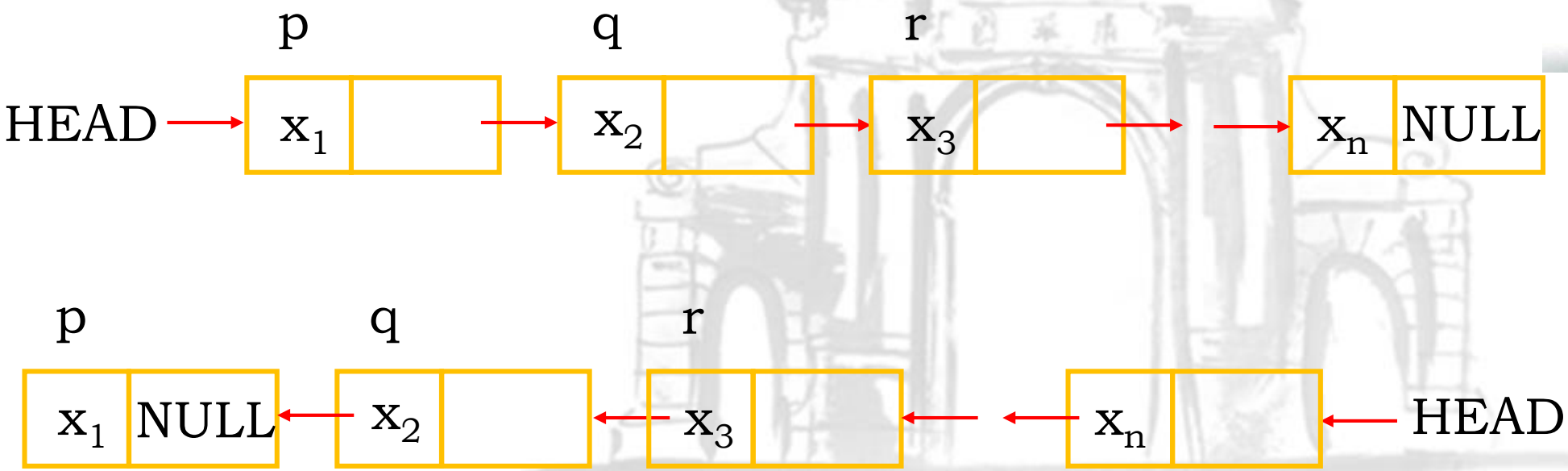
```
1  void printlst(struct node *head) {
2    //从链表的第一个结点开始打印
3    while ( head!=NULL ){
4      //打印当前结点中的数据
5      printf("%d",head->data);
6      head=head->next;
7    }
8    printf("\n");
9  }
```

实现2

课堂练习2

- 链表逆转
  - 依次遍历并逆转
- 逆转链表示意图

请思考如何实现逆转？



课堂练习2

- 逆转链表

```
1  void reverselst(struct node **head){
2    struct node *p,*q,*r;   //依次为本结点、下结点、下下结点
3    p=*head;
4    if (p==NULL) return;    //空链表直接结束
5    q=p->next;              //下一结点
6    p->next=NULL;           //头结点改为尾结点
7    while (q!=NULL){
8      r=q->next;            //r为下下结点
9      q->next=p;            //设置p为q的下一节点
10     p=q;  q=r;            //重新赋值，进行下次迭代
11   }
12   *head=p;                //更新头结点
13   return;
14 }
```

13.4 联合体与枚举类型

- 联合体
- 枚举类型
- 自定义类型名

联合体与枚举类型：联合体

- 定义：又称**共用体**，各种不同数据**共用同一段存储空间**
- 形式

```
union 联合体名
{成员表};
```

- 与结构体的异同
  - 结构体：按照定义中各个成员所需要的**存储空间的总和**来分配存储单元，其中各成员的存储**位置是不同**的
  - 联合体：按定义中需要**存储空间最大**的成员来分配存储单元，其他成员也使用该空间，它们的**首地址是相同**的

联合体与枚举类型：联合体特征

- 在定义联合体类型变量时
  - 可以将**类型的定义**与**变量的定义分开**
  - 也可以在定义联合体类型的**同时定义**该类型的变量
  - 或者**直接定义**联合体类型变量
- 由于一个联合体变量中的各成员**共用**一段**存储空间**，因此，在**任一时刻**，只能有**一种类型的数据**存放在该变量中
- 在引用联合体变量中的成员时，必须**保证数据的一致**
  - 如果最近一次存入到联合体变量中的是整型成员的数据，则在下一次取数时，也只能取该变量中整型成员中的数据

联合体与枚举类型：联合体特性


- 在定义联合体变量时**不能为其初始化**
- 联合体变量**不能作为函数参数**
- 联合体类型与结构体类型**可以互相嵌套**
  - 联合体类型可以作为结构体类型的成员
  - 结构体类型也可以作为联合体类型的成员
- 不能直接引用联合体变量本身,而只能引用联合体变量中的各成员：

  联合体变量名.成员名

- 结构体和联合体都可以用**无名**的方式定义

```
1  union EXAMPLE {
2    struct{
3        int x, y;
4    }in;
5    int a,b;
6  }e;
```

联合体与枚举类型：联合体应用

- 例13-22：联合体的使用

```
1  #include <stdio.h>     //包含头文件"stdio.h"
2  union EXAMPLE {
3    struct{
4      int x, y;
5    }in;
6    int a,b;
7  }e;
8  void main() {
9    e.a=1;
10   e.b=2;
11   e.in.x=e.a*e.b;
12   e.in.y=e.a+e.b;
13   printf("%d,%d\n",e.in.x, e.in.y);
14 }
```

```
4,8
请按任意键继续……
```

联合体与枚举类型：联合体应用

- 例13-22中联合体的使用
  - 对于联合体e，实际上是a、b、in共享内存
  - 进一步，因为in中有x和y，实际上是a、b、in.x**共享内存**
  - in.y的内存单元在in.x之后
  - 在执行e.a=1; e.b=2;后，实际上**a和b的值都是2**
  - 执行e.in.x=e.a*e.b; 之后，**a和b和in.x的值都变成4**

```
1  union EXAMPLE {
2    struct{
3        int x, y;
4    }in;
5    int a,b;
6  }e;
```

请仔细思考

联合体与枚举类型：枚举类型

- 定义：将变量的值一一列出来，变量的值**只限于列举出来的值的范围**内，有三种定义方法
  - 方法一：先定义枚举类型名,然后定义该枚举类型的变量

    enum 枚举类型名{枚举元素列表};

    - 在枚举元素列表中依次列出了该类型中所有的元素

      此时定义枚举变量为：enum 枚举类型名 变量表;
  - 方法二：在定义枚举类型的同时定义该枚举类型的变量

    enum 枚举类型名{枚举元素列表}变量表;

    - 例如：enum week{sun,mon,tue,wed,thu,fri,sat}a,b;

联合体与枚举类型：枚举类型


- 方法三：直接定义枚举类型变量

```
enum {枚举元素列表}变量表;
```

  - 例如：enum {sun,mon,tue,wed,thu,fri,sat}a,b;

- 无论哪种方式定义，在定义后都可以给枚举量a，b**赋值**
  - 如：a=mon; b=fri;
- 也可将一个整型值经**强制类型转换**后赋给枚举类型变量
  - 如：a=(enum week)1; b=(enum week)6;

联合体与枚举类型：枚举类型

- 注意事项
  - **不能对枚举元素赋值**，因为枚举元素本身就是常量(即枚举常量)
    - 在上面的定义中，下列赋值语句是错误的：mon=1
  - 虽然在程序中不能对枚举元素赋值，但实际上每个枚举元素都有**一个确定的整型值**
  - 如果在定义枚举类型时**没有显式**地给出各枚举元素的值，则这些元素的值按列出的顺序**依次取值为0，1，2，…**

联合体与枚举类型：枚举类型

- 例13-23：根据键盘输入的一周中的星期几（整数值）,输出其英文名称

```
input n:2
Tuesday
请按任意键继续……
```

```
1  #include <stdio.h>          //包含头文件"stdio.h"
2  void main() {
3    int n;
4    enum week{sun ,mon, tue, wed, thu, fri, sat} weekday;
5    printf("input n: "); scanf("%d",&n);
6    if ((n>=0)&&(n<=6)){
7       weekday = (enum week)n;
8       switch (weekday) {
9         case sun: printf("Sunday\n");    break;
10        case mon: printf("Monday\n");    break;
11        case tue: printf("Tuesday\n");   break;
12        case wed: printf("Wednesday\n"); break;
13        case thu: printf("Thursday\n");  break;
14        case fri: printf("Friday\n");    break;
15        case sat: printf("Saturday\n");  break;
16      }
17   }
18   else printf("ERR\n");
19 }
```

联合体与枚举类型：自定义类型名

- 定义
  - 用typedef声明新的类型名来代表**已有的类型名**
  - 形式：

```
typedef 原类型名 新类型名;
```

- 注意
  - 利用typedef 声明只是对已经存在的类型**增加了一个类型别名**,而**没有定义新的类型**
  - 在用typedef 指定新类型名时,习惯上将新类型名**用大写字母表示**，以便与系统提供的标准类型标识符相区别

联合体与枚举类型：自定义类型名

- 例13-24：typedef用法
  - typedef 只是为了用户书写程序的方便，使得程序简洁易读
  - 当我们定义struct student后，可以用该结构体定义变量或者申请动态内存

```
1  struct student *p, a;
2  p=(struct student*)malloc(sizeof(struct student));
```

  - 若有typedef struct student STU;则上面语句可以改为

```
1  STU *p, a;
2  p=(STU*)malloc(sizeof(STU));
```

本课总结

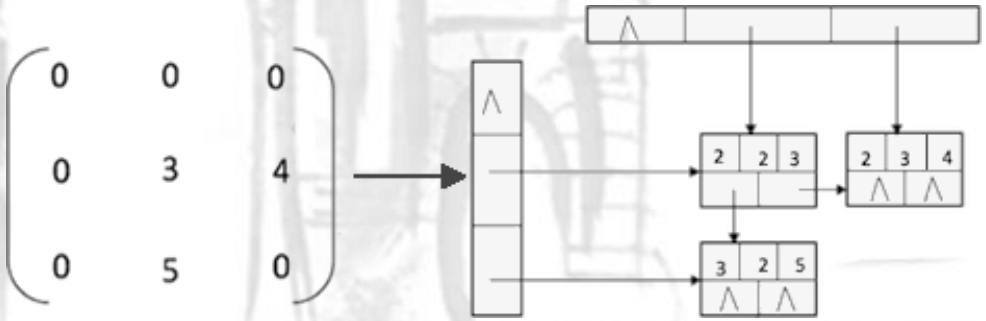
- **结构体变量与数组**
  - 结构体变量：定义、嵌套、初始化方法、作为函数参数
  - 结构体数组：定义与引用、作为函数参数
- **结构体与指针**
  - 结构体指针变量与指针数组：理解指针的含义、函数参数、引用
- **链表**
  - 概念和基本运算：查找、元素插入/删除、打印链表、逆转链表
- **联合体与枚举类型**
  - 联合体与结构体的辨析、枚举类型定义和使用

本课作业

- 第十三讲作业
  - 教材p.315习题4，5，6
  - 探究题：用环形链表解约瑟夫问题
    - 所谓环形链表，就是链表尾指针又指向了链表头，形成环状
    - 所谓约瑟夫问题，就是，m个人每人一个编号：1, 2, 3, …, m，排成一个圆圈，由第1个人开始报数，每报数到第n人则该人就出列，然后再由下一个重新报数，直到剩下最后1个人，并打印出其编号
  - 完成后将**word文档**或**拍照**提交到网络学堂

附加作业

- 结构体内存对齐
  - 分别观察结构体struct {short a1; short a2; short a3;} A;和结构体struct{long a1; short a2;} B;占用的内存，是否和你预想的相同？请分析出现这种结果的原因
  - 进一步，思考为什么定义结构体的时候，要把空间占用更小的变量写在靠前的位置？
- 有序链表合并
  - 对于两个有序整数链表，我们希望将其进行合并，生成一个更大的有序整数链表。而最直接的合并方式，是通过两个链表创建一个新的链表，但是这会消耗大量空间。
  - 请用C语言实现上述功能，要求不创建新的链表，而在原有链表上完成
- 稀疏矩阵存储
  - 链表的变化类型多种多样，常见的一类变种是十字链表，这是一种二维链表，每个节点可以指向横纵两个方向。十字链表在保存诸如稀疏矩阵等结构时具有明显优势，现要求通过C语言实现十字链表




THANKS