第15讲

结构、联合和枚举

内容提要

- 结构概述、声明与初始化
- 对结构的操作
- 嵌套数组与结构
- 联合
- 枚举

第一节

结构概述、声明与初始化

数据类型

- C语言主要有两大类数据类型
  - 标量类型：只含有单个数据项的类型
  - 聚合类型：含有多个数据项的类型
- 聚合类型除了数组，还有结构、联合和枚举等类型。

数组与结构

- 数组：相同类型数据的有序集合
  - 数据项：元素，类型相同
  - 通过下标访问元素
  - 用途：存储一组同类数据，如一个班同学的C语言成绩
  - 类比，宿舍
- 结构：相关数据的有序集合
  - 数据项：成员，类型不尽相同
  - 起名字来访问成员
  - 用途：存储一组相关数据，一个对象的相关属性，如一个同学的学籍信息
  - 类比，家里房间

声明结构变量

- 仓库零件结构：

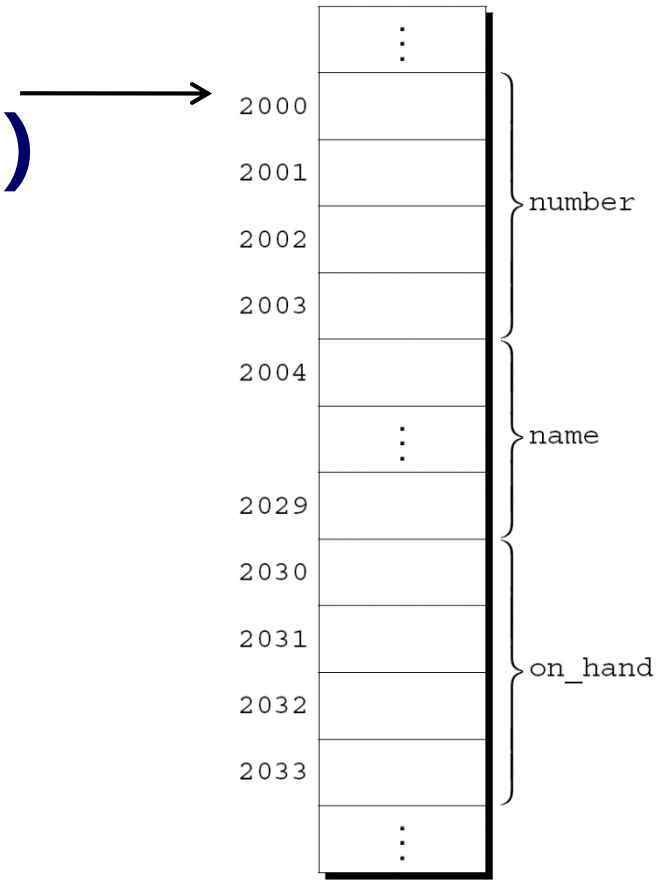
```
struct {
    int number;\\编号
    char name[NAME_LEN+1];\\名称
    int on_hand;\\数量
} part1, part2;
```

成员定义与变量类似，类型、名称

- struct {……}类型， part1,part2变量

结构变量的存储

- 结构成员按声明顺序存储
- part1存储（始地址2000）
  - number占4字节
  - NAME_LEN占25字节
  - on_hand占4字节
  - 成员之间没有空隙



结构名字空间

- 每个结构代表一个新的作用域，作用域内的标识符与外面的标识符不冲突

```
struct {
        int number;
        char name[NAME_LEN+1];
        int on_hand;
} part1, part2;
```

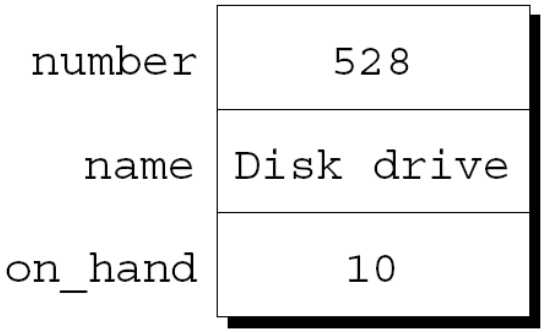
```
struct {
        int number;
        char name[NAME_LEN+1];
        char sex;
} employee1, employee2;
```

初始化结构变量

- 零件初始化:

```
struct {
    int number;
    char name[NAME_LEN+1];
    int on_hand;
} part1 = {528, "Disk
drive", 10},
  part2 = {914, "Printer
cable", 5};
```

- 初始化后的part1:



第二节

对结构的操作

对结构的操作

- 访问结构成员：
  - 句点. 运算符
  - 结构名. 成员名
- 结构成员可作变量使用，左值

对结构的操作-示例

- 输入\出part1成员值
  - scanf("%d", &part1.number);
  - printf("Part name: %s\n", part1.name);
  - printf("Quantity on hand: %d\n", part1.on_hand);

对结构的操作-示例

- 结构成员赋值及自增:
  - part1.number = 258;
  - part1.on_hand++;

对结构的操作

- 结构赋值，与数组之重大区别
  - part2 = part1;
- 等价于
  - part2.number=part1.numbe
  - part2.name=part1.name;//
  - part2.on_hand=part1.on_hand;

数组也被整体赋值可用之技

对结构的操作

- 前提：
  - 只能用于兼容类型的结构
  - 如 part1 和part2兼容
- 除了赋值，C不提供对整个结构的操作
  - ==和!=不能用于结构.

命名结构类型

- 前面定义结构变量的方法有时不太方便
  - 需要声明多个具有相同类型的结构变量
  - 在程序不同位置声明
- 先声明结构类型，再声明结构变
- 命名结构类型方法
  - 声明结构标记
  - typedef定义结构类型

```
struct {
        …
            …
}part1,
part2,…
```

声明结构标记

- 声明名为part的结构标记

```
struct part{
    int number;
    char name[NAME_LEN+1];
    int on_hand;
};//分号不可少
```

- 结构标记不是结构类型名称
  - struct+结构标记名，才是完整结构类型名

- 用part标记声明结构变量

```
struct part part1,
part2;
```

  - struct part结构类型名

声明（定义）结构类型

- typedef定义结构类型，如定义名为Part的结构类型

```
typedef struct{
    int number;
    char name[NAME_LEN+1];
    int on_hand;
}Part;
```

struct及大括号包含的全部成员声明就是完整的结构类型

typedef 类型名 别名;

- 定义Part类型变量：Part part1, part2;

结构作为参数和返回值

- 以结构变量作为参数的函数:

```
void print_part(struct part p){
   printf("Part number: %d\n", p.number);
   printf("Part name: %s\n", p.name);
   printf("Quantity on hand: %d\n",
p.on_hand);
}
```

- 调用函数: print_part(part1);

结构作为参数和返回值

- 返回part结构的函数:

```
struct part build_part(int number, const char
*name, int on_hand) {
  struct part p;
  p.number = number;
  strcpy(p.name, name);
  p.on_hand = on_hand;
  return p; }
```

- 函数调用: part1 = build_part(528, "Disk drive", 10);

第三节

嵌套数组与结构

嵌套数组和结构

- 结构和数组可以无约束地嵌套
  - 结构在外层
    - 以数组为成员\以结构为成员
  - 数组在外层
    - 以数组为元素\以结构为元素

嵌套结构

人名结构

```
struct person_name {
  char first[FIRST_NAME_LEN+1];
  char middle_initial;
  char last[LAST_NAME_LEN+1];
};
```

person_name嵌入student:

```
struct student {
  struct person_name name;
  int id, age;
  char sex;
} student1, student2;
```

嵌套成员访问：逐层访问

- 比如访问student1的姓：需要使用.操作符两次
- strcpy(student1.name.last, "Fred");

结构数组

- 以结构为元素构造数组：数组和结构最常见的组合之一。可以作为简单的数据库。
  - 如一栋楼（单元）同一房号的房子
- 存储100个part的结构数组:

```
struct part inventory[100];
struct student stu_list[MAX_STU_NUM];
```

结构数组操作

- 下标访问数组元素（结构）：

```
print_part(inventory[i]);
```

- 访问part结构成员需联合使用下标和成员选择:

```
inventory[i].number = 883;
```

- 访问part名里的单个字符需要使用下标、再选择、再下标:

```
inventory[i].name[0] = '\0';
```

指针操作结构

- 声明指针

```
struct part *p;
```

- 指向结构

```
p=&part1;
```

- 通过指针访问结构变量

```
(*p).number = 202;//括号
或p->number = 202;//常用
scanf("%s",p->name);
scanf("%c",&p->sex);
printf("%s",p->name);
```

程序设计: 维护零件数据库

- 程序跟踪、管理存储在仓库中的零件
- 零件的信息存储在一个结构数组中.
- 每个结构的内容:
  - 零件号
  - 名称
  - 数量

程序设计: 维护零件数据库

- 程序支持的操作:
  - 添加新零件：零件号，零件名，和数量
  - 打印零件：给定零件号，打印出零件名和数量
  - 修改零件数量：给定零件号，改变数量
  - 显示所有零件：显示数据库中的所有信息
  - 终止程序的执行

温故而知新——结构

- 数组——同类型数据集合
  - 一个班同学名单：

```
char name[69][NLEN+1];
```

  - 一个班同学C成绩：

```
int scores[69];
```

- 结构——密切关联数据的集合
  - 一个学生信息：
    - 姓名、性别、学号、出身年月
  - 一个零件信息：
    - 零件号、零件名、数量

温故而知新——结构定义

- 类型声明——结构标记:

```
struct student{
    char name[NLEN+1];
    char sex;
    int age;
    char number[SLEN+1];
};
```

- 变量声明:

```
struct student
stu1,stu2;
```

- 类型声明——typedef

```
typedef struct{
    char name[NLEN+1];
    char sex;
    int age;
    char number[SLEN+1];
}Student;
```

- 变量声明:

```
Student stu1, stu2;
```

温故而知新——结构操作

- 句点

```
part1.name;
student1.name.last
inventory[i].number
```

- 指针->

```
p=&part1;
(*p).number = 202;//括号
或p->number = 202;//常用
```

程序设计: 维护零件数据库

- 操作（代码）：
  - i：插入
  - s：查找
  - u：更新
  - p：打印
  - q：退出
- 与程序的一个会话，如右图

```
Enter operation code: i
Enter part number: 528
Enter part name: Disk drive
Enter quantity on hand: 10

Enter operation code: s
Enter part number: 528
Part name: Disk drive
Quantity on hand: 10

Enter operation code: p
Part Number   Part Name          Quantity on Hand
          528     Disk drive                8
          914     Printer cable             5

Enter operation code: q
```

程序设计: 维护零件数据库

- 每个零件信息存储在一个struct part结构里.
- 以struct part为元素构建inventory结构数组

```
struct part inventory[MAX_PARTS];
```

- 变量num_parts跟踪当前零件数量(inventory长度)

```
int num_parts = 0;
```

程序设计: 维护零件数据库

- 程序的主循环概要:

```
for (;;) {
        提示用户输入操作码;
        读入操作码;
        switch (操作码) {
          case ‘i’ : 执行插入操作; break;
          case ‘s’ : 执行查找操作; break;
          case ‘u’ : 执行更新操作; break;
          case ‘p’ : 执行打印操作; break;
          case ‘q’ : 终止程序;
          default:  打印错误消息;
        }
}
```

程序设计: 维护零件数据库

- 插入、查找、更新和打印操作单独函数实现
- inventory、num_parts外部变量
- 程序分为三个文件:
  - inventory.c（程序的主体）
  - readline.h（含有 read_line函数的原型）
  - readline.c（含有read_line函数的定义）

inventory.c——main()

```
/* Maintains a parts database (array version) */
 #include <stdio.h>
#include "readline.h"
#define NAME_LEN 25
#define MAX_PARTS 100

struct part {
   int number;
   char name[NAME_LEN+1];
   int on_hand;
} inventory[MAX_PARTS];
int num_parts = 0;   /* number of parts currently stored */

int find_part(int number);
void insert(void);
void search(void);
void update(void);
void print(void);
```

inventory.c——main()

```
int main(void)
{
  char code;//操作码
  for (;;) {
        printf("Enter operation code: ");
        scanf(" %c", &code);
        while (getchar() != '\n')  ;//取走操作码后多余回车
        switch (code) {
              case 'i': insert(); break;
              case 's': search(); break;
              case 'u': update(); break;
              case 'p': print(); break;
              case 'q': return 0;
              default:  printf("Illegal code\n");
        }
        printf("\n");
  }
}
```

inventory.c—— find_part()

```
/**********************************************************
 * find_part: Looks up a part number in the inventory     *
 *            array. Returns the array index if the part  *
 *            number is found; otherwise, returns -1.     *
 **********************************************************/
int find_part(int number)
{
  int i;

  for (i = 0; i < num_parts; i++)
        if (inventory[i].number == number)
              return i;
  return -1;
}
```

inventory.c—— insert()

```
void insert(void)
{
  int part_number;

  if (num_parts == MAX_PARTS) {
        printf("Database is full; can't add more parts.\n");
        return;
  }
  printf("Enter part number: ");
  scanf("%d", &part_number);
  if (find_part(part_number) >= 0) {
        printf("Part already exists.\n");
        return;
  }//零件已在
  inventory[num_parts].number = part_number;
  printf("Enter part name: ");
  read_line(inventory[num_parts].name, NAME_LEN);
  printf("Enter quantity on hand: ");
  scanf("%d", &inventory[num_parts].on_hand);
  num_parts++;
}
```

inventory.c—— search()

```
/**********************************************************
 * search: Prompts the user to enter a part number, then  *
 *         looks up the part in the database. If the part *
 *         exists, prints the name and quantity on hand;  *
 *         if not, prints an error message.               *
 **********************************************************/
void search(void)
{
  int i, number;

  printf("Enter part number: ");
  scanf("%d", &number);
  i = find_part(number);
  if (i >= 0) {//成功找到
          printf("Part name: %s\n", inventory[i].name);
          printf("Quantity on hand: %d\n", inventory[i].on_hand);
  } else
          printf("Part not found.\n");
}
```

inventory.c—— update()

```
/**********************************************************
 * update: Prompts the user to enter a part number.      *
 *         Prints an error message if the part doesn't   *
 *         exist; otherwise, prompts the user to enter   *
 *         change in quantity on hand and updates the    *
 *         database.                                     *
 **********************************************************/
void update(void)
{
  int i, number, change;
  printf("Enter part number: ");
  scanf("%d", &number);
  i = find_part(number);
  if (i >= 0) {
        printf("Enter change in quantity on hand: ");
              scanf("%d", &change);
        inventory[i].on_hand += change;//增加存量
  } else
        printf("Part not found.\n");
}
```

inventory.c—— print()

```
/*************************************************************
 * print: Prints a listing of all parts in the database,  *
 *        showing the part number, part name, and         *
 *        quantity on hand. Parts are printed in the      *
 *        order in which they were entered into the       *
 *        database.                                       *
 *************************************************************/
void print(void)
{
  int i;
   printf("Part Number   Part Name                  "
          "Quantity on Hand\n");
  for (i = 0; i < num_parts; i++)
          printf("%7d      %-25s%11d\n", inventory[i].number,
                 inventory[i].name, inventory[i].on_hand);
}
```

第四节

联合

联合

- 由一个或多个类型不尽相同，不同时存在的成员组成，类似结构
  - 结构：各成员同时存在
    - 联系方式：tel;mail;QQ;
  - 联合：各成员不同时存在，一山不容二虎
    - 联系方式（任一）：
- 编译器只为最大的成员分配足够的空间，各个成员共享空间（相互覆盖）
- 为一个成员赋一个新值会改变其他成员的值.

联合与结构示例

联合示例

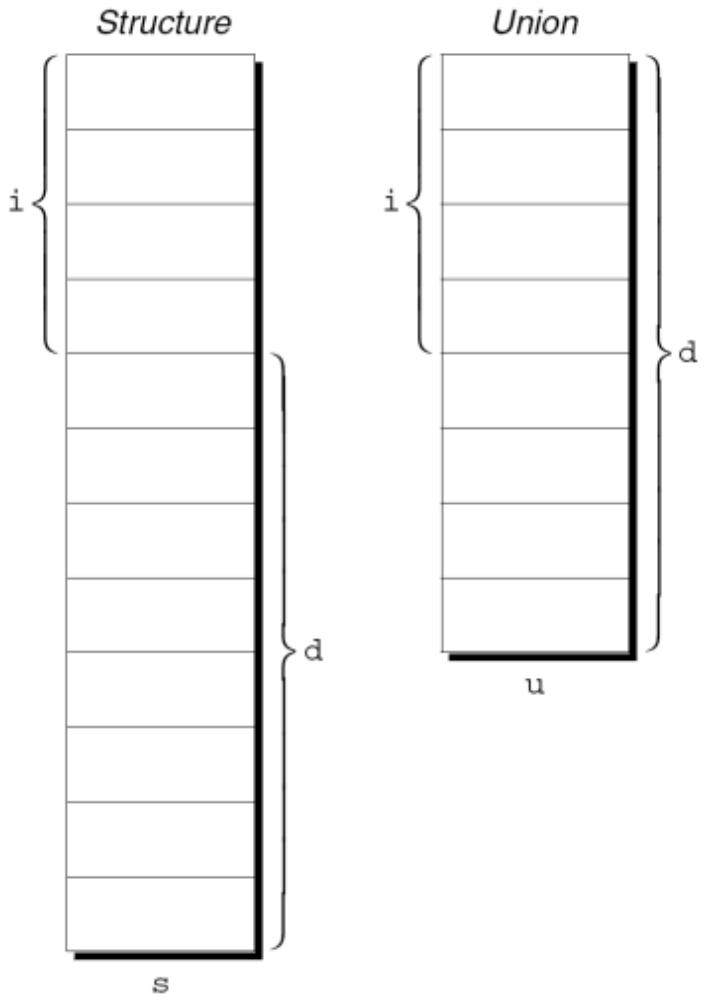
```
union {
        int i;
        double d;
} u;
```

结构示例

```
struct {
        int i;
        double d;
} s;
```

联合

- 结构s和联合u的区别：
  - s的成员在内存中有各自的存储空间
  - u的成员共享存储空间
- 在一个时刻联合的成员只有一个有效
  - 联合成员i和d只有一个有效
  - 结构成员i和d同时有效



联合的初始化与操作

联合初始化

```
union {
        int i;
        double d;
} u={0};
```

联合操作

```
u.i = 82;
u.d = 74.8;
```

使用联合节省空间

- 设计礼品目录
- 每件商品有一个货号和价格
  - stock_number、 price
- 以及其他一些依赖于商品类型的信息
  - Books: title, author, num_pages
  - Mugs: design(图案)
  - Shirts: design, colors, sizes

使用联合节省空间

- 礼品统一声明为结构：浪费大量内存空间

```
struct catalog_item {
      int stock_number;
      double price;
      char title[TITLE_LEN+1];
      char author[AUTHOR_LEN+1];
      int num_pages;
      char design[DESIGN_LEN+1];
      char design[DESIGN_LEN+1];
      int colors;
      int sizes;
};
```

使用联合节省空间

```
struct catalog_item {
    int stock_number;
    double price;
    int item_type;
    union {
        struct {
            char title[TITLE_LEN+1];
            char author[AUTHOR_LEN+1];
            int num_pages;
        } book;
        struct {
            char design[DESIGN_LEN+1];
        } mug;
        struct {
            char design[DESIGN_LEN+1];
            int colors;
            int sizes;
        } shirt;
    } item;
};
```

第五节

枚举

枚举

- 很多程序需要取值范围为一个很小集合的变量，比如只有几种可能的取值
  - poker花色：“梅花”，“方块”，“红桃”和“黑桃”
    - {CLUBS, DIAMONDS, HEARTS, SPADES}
  - 院系：通信、电工、计算机、数学、外语……
    - {COM,EE,CS,MS,FL……}
- 可以用整型或宏定义
  - 整型：不能表达范围
  - 宏：麻烦，无法体现同类型，且调试时不可见
- 用枚举类型表示
  - 值由程序员列举（“枚举”），变量值只限于列举的值的范围内.
- 每个值必须有个名字(一个枚举常量)

枚举标签和类型名

- 声明枚举标记

```
enum suit {CLUBS, DIAMONDS, HEARTS, SPADES};
enum suit s1, s2;
```

- typedef创建一个Suit类型名

```
typedef enum {CLUBS, DIAMONDS, HEARTS, SPADES}
Suit;
Suit s1, s2;
```

- 枚举常量与变量
  - 列举：常量，eg. DIAMONDS
  - 声明：变量，eg. s1,s2

作为整型的枚举

- C把枚举变量和常量当作整型处理.
  - 编译器默认将整数0，1，2，… 赋给枚举中的常量.
  - 在suit枚举中，CLUBS，DIAMONDS，HEARTS，和SPADES 分别表示0，1，2 和3.
- 枚举实质（个人理解）：
  - 取值范围受限、明确的整数

作为整型的枚举

- 可以为枚举常量选择不同的值:

```
enum suit {CLUBS = 1, DIAMONDS=2, HEARTS
= 3, SPADES = 4};
```

- 枚举常量值可为任意整数,没有特别的顺序:

```
enum dept {RESEARCH = 20,
PRODUCTION = 10, SALES = 25};
```

- 两个或多个枚举常量具有相同的值也合法

作为整型的枚举

- 枚举值与普通整数可以混合：

```
int i;
enum {CLUBS, DIAMONDS, HEARTS, SPADES} s;
i = DIAMONDS;
```

- s被当作某种整型变量

```
s = 0;
s++;
i = s + 2;
```