14 结构和其它数据形式

内容提要

- 关键字　struct, union, typedef.
- 运算符　. ->
- 什么是 C 语言中的结构？如何创建结构模板和结构变量？
- 如何访问结构成员？如何编写处理结构的函数？
- C 的 typedef 工具.
- 联合及指向函数的指针.

示例问题：创建图书目录

1 示例问题：创建图书目录

- 打印关于每本图书的各种信息：书名，作者，出版商，版权日期，页数，册数及价格
  - 其中的一些项目（如，书名)可以存储再字符串数组，其他的项目需要一个int数组或float数组
  - 需要一种即能包含字符串又能包含数字的数据形式，而且还要保持各信息的独立
  - 创建多份列表：一份按书名排序、一份按作者排序、一份按价格排序等
    - 如果能把图书目录的信息都包含在一个数组里更好，其中每个元素包含一本书的相关信息
- 14.1　book.c
- 结构体必须掌握的3个技巧
  - 为结构建立一个格式或样式【自定义结构体类型】
  - 声明一个适合该样式的变量
  - 访问结构变量的各个部分

1 示例问题：创建图书目录

- 14.1 book.c
- 结构有3部分
  - 每个部分为成员（member）或字段（field）
- s_gets()函数去掉fgets()存储在字符串中的换行符

```c
//* book.c -- one-book inventory */
#include <stdio.h>
#include <string.h>
char * s_gets(char * st, int n);
#define MAXTITL  41  // maximum title+1
#define MAXAUTL  31  // maximum author's name + 1

struct book { // structure template: tag is book
    char title[MAXTITL];
    char author[MAXAUTL];
    float value;
}; /* end of structure template  */
```

```c
int main(void){
    struct book library;
    s_gets(library.title, MAXTITL);
    s_gets(library.author, MAXAUTL);
    scanf("%f", &library.value);
    printf("%s by %s: $%.2f\n ("%s: \"%s\" ($%.2f)\
",library.title, library.author, library.value),
library.author, library.title, library.value);
    return 0;
}
char * s_gets(char * st, int n){
    char * ret_val = fgets(st, n, stdin);
    if (ret_val) {
        char * find = strchr(st, '\n');
        if (find) *find = '\0';
        else while (getchar() != '\n') continue;
    }
    return ret_val;
}
```

建立结构声明

2 建立结构声明

- 结构声明（structure declaration)描述了一个结构的组织布局【定义类型模板，不是实体】
  - 关键字struct
    - 表明其后的是一个结构。其后可选为结构体名
  - 在结构声明中，用一对花括号括起来的是结构成员列表。每个成员都用自己的声明来描述

```
1. struct book {
2.     char title[MAXTITL];
3.     char author[MAXAUTL];
4.     float value;
5. };

6. struct book library;
```

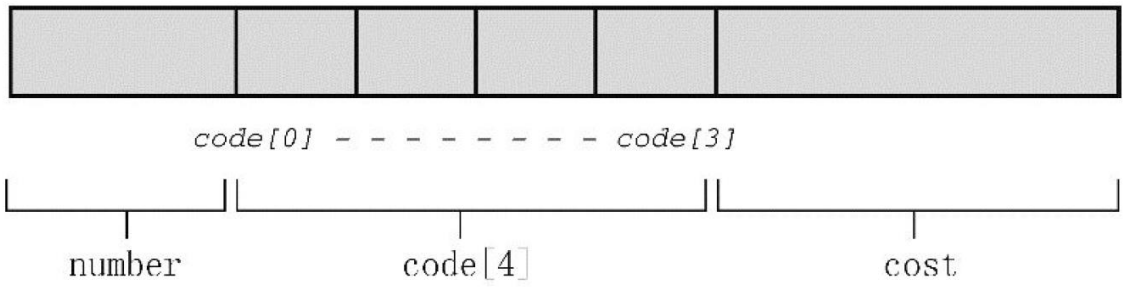
- struct book library;
  - 把library声明为一个使用book结构布局的结构变量

1 定义结构变量

3 定义结构变量

- struct book library;
- 在结构变量的声明中，struct book所起的作用相当于一般声明中的int或float
- 也将声明和变量定义合并在一起【不推荐】

```
struct stuff {
        int number;
        char code[4];
        float cost;
        };
```




```
1.  struct book {
2.      char title[MAXTITL];
3.      char author[AXAUTL];
4.      float value;
5.  } library; /* 声明的右花括号后跟变量名*/

6.  struct {//无结构标记。只适合于特定匿名结构体
7.      char title[MAXTITL];
8.      char author[AXAUTL];
9.      float value;
10. } library; /* 声明的右花括号后跟变量名*/
```

3.1 初始化结构

- 使用在一对花括号中括起来的初始化列表进行初始化，各初始化项用逗号分隔
  - 每个成员的初始化项独占一行。提高代码的可读性
  - 对编译器而言，只需逗号分隔各成员即可

```
struct book library = {
    "The Pious Pirate and the Devious Damsel",
    "Renee Vivotte",
    1.95
};
```

- 如果初始化一个静态存储期的结构，初始化列表中的值必须是常量表达式。如果是自动存储期，初始化列表中的值可以不是常量

3.2 访问结构成员

- 用结构成员运算符点(.)访问结构中的各个成员
- 结构就像是一个 "超级数组"
  - 在本质上，.title，.author 和 .value 在 book 结构中扮演了下标的角色
  - 和新一代语言观点一致

```
1. struct book bill, newt;
2. s_gets(bill.title, MAXTITL);
3. s_gets(newt.title, MAXTITL);
```

3.3 结构的指定初始化项目

- C99和C11为结构提供了指定初始化器（designated initializer）
  - 其语法与数组的指定初始化器类似
  - 结构的指定初始化器使用点运算符和成员名（而不是方括号和下标）标识特定的元素。

```
1. struct book surprise = { .value = 10.99};

2. struct book gift = { .value = 25.99,
3.                      .author = "James Broadfool",
4.                      .title = "Rue for the Toad"};
```

结构数组

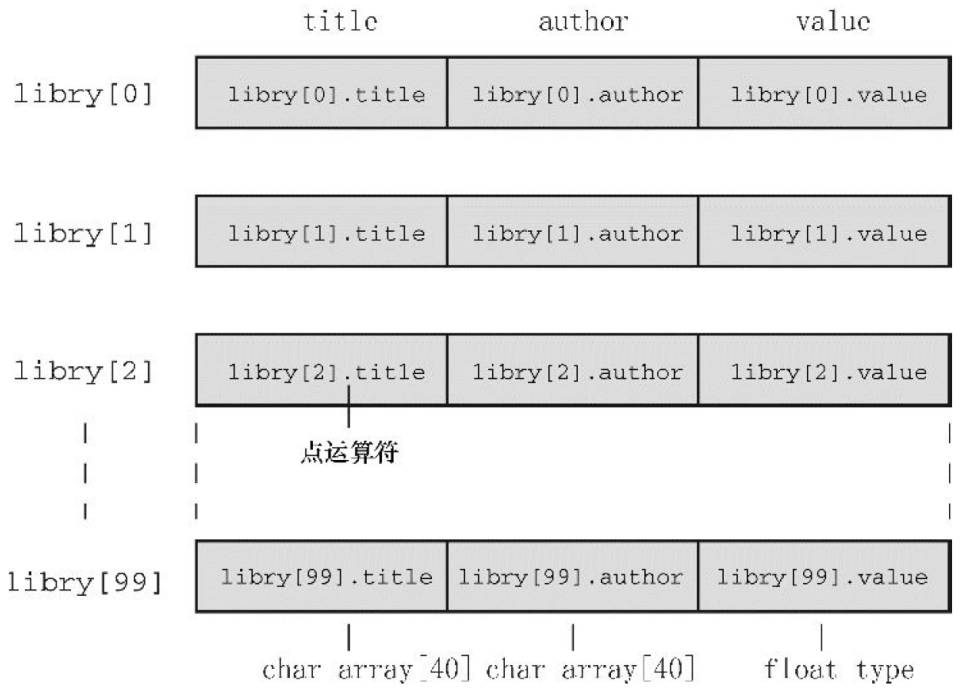
4 结构数组

使用结构数组来处理多本书,14.2 manybook.c

```
1.  char * s_gets(char * st, int n);
2.  #define MAXTITL   40
3.  #define MAXAUTL   40
4.  #define MAXBKS   100  // maximum number of books
5.  struct book {// set up book template
6.      char title[MAXTITL];
7.      char author[MAXAUTL];
8.      float value;
9.  };
10. int main(void){
11.     struct book library[MAXBKS]; // array of book
12.     int count = 0;
13.     int index;
14.     while (count < MAXBKS &&
    s_gets(library[count].title, MAXTITL) != NULL &&
    library[count].title[0] != '\0') {
15.         s_gets(library[count].author, MAXAUTL);
16.         scanf("%f", &library[count++].value);
17.         while (getchar() != '\n') continue;
18.         if (count < MAXBKS) printf("Next title.\n");
19.     }
20.     if (count > 0){
21.         for (index = 0; index < count; index++)
22.             printf("%s by %s: $%.2f\n",
    library[index].title,
    library[index].author, library[index].value);
23.     }
24.     else printf("No books? Too bad.\n");
25.     return 0;
26. }
```

4.1 声明结构数组

- `struct book library[MAXBKS];`
- 声明一个结构数组和声明其他任何类型的数组一样
  - 数组的每个元素都是一个book类型的结构



4.2 标识结构数组的成员

- 在结构名后加一个点运算符，然后是成员名.

```
1. library // 一个book 结构的数组
2. library[2] // 一个数组元素，该元素是book结构
3. library[2].title // 一个char数组（library[2]的title成员）
4. library[2].title[4] // 数组中library[2]元素的title 成员的一个字符
```

4.3 程序讨论

- 相对于第一个程序的主要变化
  - 插入一个while循环读取多个项

- 14.2 manybook.c

```
1. while (count < MAXBKS &&
2. s_gets(library[count].title, MAXTITL) != NULL &&
3. library[count].title[0] != '\0')

4. while (getchar() != '\n')
5.     continue; /* 清理输入行 */
6. //弥补了scanf()函数遇到空格和换行符就结束读取的问题
```

5 嵌套结构

5 嵌套结构

- 嵌套结构，在一个结构中包含另一个结构
- 14.3 friend.c
- struct names handle; /* 嵌套结构 */

```c
#define LEN 20
const char * msgs[] = {"Thank you for the wonderful evening, ", "You certainly prove that a ", "is a special kind of guy. We must get together};
struct names {                          // first structure
    char first[LEN];
    char last[LEN];
};
struct guy {                            // second structure
    struct names handle;                // nested structure
    char favfood[LEN];
    char job[LEN];
    float income;
};
```

```c
int main(void)
{
    struct guy fellow = {   // initialize a variable
        { "Ewen", "Villard" },
        "grilled salmon
        68112.00
    };

    if (fellow.income > 150000.0)
        puts("!!");
    else if (fellow.income > 75000.0)
        puts("!");
    else
        puts(".");
    printf("\n%40s%s\n", " ", "See you soon,");
    printf("%40s%s\n", " ", "Shalala");
    return 0;
}
```

6 指向结构的指针

指向结构的指针

- 为什么指向结构的指针？
  - 就像指向数组的指针比数组本身更容易操控（如，排序问题）一样，指向结构的指针通常比结构本身更容易操控
  - 在一些早期的C实现中，结构不能作为参数传递给函数，但是可以传递指向结构的指针
  - 传递指针通常更有效率
  - 一些用于表示数据的结构中包含指向其他结构的指针（如链表）

6 指向结构的指针

➢ 14.4 friends.c

```c
#include <stdio.h>
#define LEN 20
struct names {
    char first[LEN];
    char last[LEN];
};
struct guy {
    struct names handle;
    char favfood[LEN];
    char job[LEN];
    float income;
};
```

```c
int main(void){
    struct guy fellow[2] = { {{ "Ewen", "Villard"},
"grilled salmon", "personality coach", 68112.00},
{{"Rodney", "Swillbelly"}, "tripe", "tabloid editor",
432400.00 }
    };
    struct guy * him;
    printf("address #1: %p #2: %p\n", &fellow[0],
&fellow[1]);
    him = &fellow[0];
    printf("pointer #1: %p #2: %p\n", him, him + 1);
    printf("him->income is $%.2f: (*him).income is
$%.2f\n", him->income, (*him).income);
    him++;  /* point to the next structure      */
    printf("him->favfood is %s:  him->handle.last is
%s\n", him->favfood, him->handle.last);
    return 0;
}
```

6.1 声明和初始化结构指针

- `struct guy * him; //`声明结构指针
- 和数组不同，一个结构的名字不是该结构的地址，必须使用 & 运算符.
- `him = &barney;`

- 在有些系统中，一个结构的大小可能大于它各成员大小之和
  - 系统对数据进行校准的过程中产生了一些“缝隙”。
  - 有些系统必须把每个成员都放在偶数地址上，或4的倍数的地址上【内存对齐】
    - 该系统中，结构的内部就存在未使用的“缝隙”

6.2 使用指针访问成员

- 第1种方法也是最常用的方法
  - 使用->运算符
- 第2种方法
  - 使用*

- him -> income
- fellow[0].income == (*him).income

7 向函数传递结构信息

7 向函数传递结构信息

- ANSI C允许把结构作为参数使用。所以程序员可以选择
  - 传递结构本身
  - 传递指向结构的指针
  - 如果只关心结构中的某一部分，也可以把结构的成员作为参数

7.1 传递结构成员

- 14.5 funds1.c
- 财务分析程序
- sum(stan.bankfund, stan.savefund)

```c
/* adds two double numbers */
double sum(double x, double y)
{
    return(x + y);
}
```

```c
#include <stdio.h>
#define FUNDLEN 50
struct funds {
    char   bank[FUNDLEN];
    double bankfund;
    char   save[FUNDLEN];
    double savefund;
};
double sum(double, double);
int main(void){
    struct funds stan = {"Garlic-Melon Bank", 4032.27, "Lucky's Savings and Loan", 8543.94 };
    printf("Stan has a total of $%.2f.\n",
            sum(stan.bankfund, stan.savefund) );
    return 0;
}
```

7.2 使用结构地址

- 14.6 funds2.c
- sum(&stan)
- double sum(const struct funds *); /* 参数是一个指针 */

```c
#include <stdio.h>
#define FUNDLEN 50
struct funds {
    char   bank[FUNDLEN];
    double bankfund;
    char   save[FUNDLEN];
    double savefund;
};

double sum(const struct funds *);
int main(void){
    struct funds stan = { "Garlic-Melon Bank",4032.27, "Lucky's Savings and Loan", 8543.94};
    printf("Stan has a total of $%.2f.\n", sum(&stan));
    return 0;
}
```

```c
double sum(const struct funds * money)
{
    return(money->bankfund + money->savefund);
}
```

7.3 把结构作为参数传递

- 14.7 funds3.c
- sum(stan)
- double sum(struct funds moolah); /* 参数是一个结构 */

```c
double sum(struct funds moolah)
{
    return(moolah.bankfund + moolah.savefund);
}
```

```c
#define FUNDLEN 50
struct funds {
    char   bank[FUNDLEN];
    double bankfund;
    char   save[FUNDLEN];
    double savefund;
};
double sum(struct funds moolah);

int main(void){
    struct funds stan = {"Garlic-Melon Bank",
4032.27,"Lucky's Savings and Loan", 8543.94};

    printf("Stan has a total of $%.2f.\n", sum(stan));

    return 0;
}
```

7.4 其他结构特性

- 一个结构可赋值给另一个结构，但数组不能直接复制！
  - o_data = n_data; //把一个结构赋值给另一个结构，
    - 把n_data 的每个成员的值都赋给o _data 的相应成员
- 14.8 names1.c
- 14.9 names2.c
  - 传递的返回结构

```
1.  #define NLEN 30
2.  struct namect {
3.      char fname[NLEN];
4.      char lname[NLEN];
5.      int letters;
6.  };
7.  void getinfo (struct namect * pst){}
8.  void makeinfo (struct namect * pst){}
9.  void showinfo (const struct namect * pst){}
10. char * s_gets(char * st, int n);
11. int main(void){
12.     struct namect person;
13.     getinfo(&person);
14.     makeinfo(&person);
15.     showinfo(&person);
16.     return 0;
17. }
```

7.5 结构，还是指向结构的指针

- 把指针作为参数
  - 它既工作在较早的 C 实现上，也工作在较新的 C 实现上，而且执行起来很快；只段传递一个单个地址.
  - 缺点是缺少对数据的保护。 被调函数中的一些操作可能不经意地影响到原来结构中的数据
    - ANSI C 新增的 const 限定词解决了这个问题
- 把结构作为参数传递
  - 优点
    - 函数处理的是原始数据的副本，这保护了原始数据。另外，代码风格也更清楚
  - 缺点
    - 较老版本的实现可能无法处理这样的代码，而且传递结构浪费时间和存储空间。尤其是把大型结构传递给函数，而它只使用结构中的一两个成员时特别浪费

7.6 在结构中使用字符数组还是字符指针

- 清单 14.3
- 如需要在结构中存储字符串，请使用字符数组成员！
  - 存储字符指针有它的用处，但有被严重误用的可能

```
1. #define LEN 20
2. struct names {
3.     char first[LEN];
4.     char last[LEN];
5. };

6. struct pnames {
7.     char * first;
8.     char * last;
9. };
```

7.7 结构，指针和malloc()

- 14.10 names3.c

- 在结构中使用指针处理字符串的一个有意义的例子是使用 malloc() 分配内存，并用指针来存放地址. 这个方法的优点是可以请求 malloc() 分配刚好满足字符串需求数量的空间.

```
1.  // names3.c -- use pointers and malloc()
2.  #include <stdio.h>
3.  #include <string.h>   // for strcpy(), strlen()
4.  #include <stdlib.h>   // for malloc(), free()
5.  #define SLEN 81
6.  struct namect {
7.      char * fname;  // using pointers
8.      char * lname;
9.      int letters;
10. };

11. void getinfo(struct namect *);  // allocates memory
12. void makeinfo(struct namect *);
13. void showinfo(const struct namect *);
14. void cleanup(struct namect *); // free
15. char * s_gets(char * st, int n);
```

7.7 结构，指针和malloc()

- 14.10 names3.c
- 使用指针和 malloc() 函数

```c
#include <stdio.h>
#include <string.h>   // for strcpy(), strlen()
#include <stdlib.h>   // for malloc(), free()
#define SLEN 81
struct namect {
    char * fname;  // using pointers
    char * lname;
    int letters;
};
void getinfo(struct namect *);  // allocates memory
void makeinfo(struct namect *);
void showinfo(const struct namect *);
void cleanup(struct namect *); // free memory
char * s_gets(char * st, int n);
```

```c
int main(void){
    struct namect person;
    getinfo(&person);
    makeinfo(&person);
    showinfo(&person);
    cleanup(&person);
    return 0;
}

void getinfo (struct namect * pst){}
void makeinfo (struct namect * pst){}
void showinfo (const struct namect * pst){}
void cleanup(struct namect * pst){
    free(pst->fname);
    free(pst->lname);
}
```

7.8 复合文字和结构（C99）

- 14.11 complit.c
- ### VS2010 不支持此特性，Ubuntu gcc 支持

```
1.  #define MAXTITL  41
2.  #define MAXAUTL  31
3.  struct book {// structure template: tag is book
4.      char title[MAXTITL];
5.      char author[MAXAUTL];
6.      float value;
7.  };
8.  int main(void){
9.      struct book readfirst;
10.     int score;
11.     scanf("%d", &score);
12.     if(score >= 84)
13.         readfirst = (struct book) {"Crime and
    Punishment","Fyodor Dostoyevsky", 11.25};
14.     else
15.         readfirst = (struct book) {"Mr. Bouncy's Nice
    Hat", "Fred Winsome", 5.99};
16.     return 0;
17. }
```

7.9 伸缩型数组成员（C99，谨慎使用）

- 14.12 flexmemb.c
- VS 2010 不支持，gcc 支持

```
1. int count;
2. double average;
3. double scores[]; // 伸缩型数组成员
```

7.10 匿名结构（C11）

- 匿名结构是一个没有名称的结构成员

```
1.  struct names{
2.      char first[20];
3.      char last[20];
4.  };

5.  struct person{
6.      int id;
7.      struct names name; // 嵌套结构成员
8.  };

9.  struct person ted = {8483, {"Ted", "Grass"}};

10. struct person{
11.     int id;
12.     struct {
13.         char first[20]; char last[20];
14.     }; //匿名结构
15. };
```

7.11 使用结构数组的函数

```
1.  14.13  funds4.c
2.  #define FUNDLEN 50
3.  #define N 2
4.  struct funds {
5.      char   bank[FUNDLEN];
6.      double bankfund;
7.      char   save[FUNDLEN];
8.      double savefund;
9.  };

10. double sum(const struct funds money[], int n);

11. int main(void){
12.     struct funds jones[N] = {
13.         {"Garlic-Melon Bank", 4032.27, "Lucky's
    Savings and Loan", 8543.94},
14.         {"Honest Jack's Bank",3620.88, "Party Time
    Savings", 3802.91}
15.     };
16.     printf("The Joneses have a total of $%.2f.\n",
17.            sum(jones,N));
18.     return 0;
19. }

20. double sum(const struct funds money[], int n){
21.     double total;
22.     int i;
23.
24.     for (i = 0, total = 0; i < n; i++)
25.         total += money[i].bankfund +
    money[i].savefund;
26.
27.     return(total);
28. }
```

8 把结构内容保存到文件中

8 把结构内容保存到文件中

- 结构可以保存多种多样的信息，所以它是建立数据库的重要工具

- 数据库文件可以包含任意数量的此类数据对象。存储在一个结构中的整套信息被称为记录（record），单独的项被称为字段（field）

- 保存
  - fprintf (pbooks, "%s %s %.2f \n", primer.title, primer, author, primer,value);
  - 更好的方案是使用fread()和fwrite() 函数读写结构大小的单元。
    - fwrite (&primer, sizeof (struct book) , 1, pbooks) ;

8.1 一个结构保存的实例

程序清单14.14 booksave.c

```c
#define MAXTITL  40
#define MAXAUTL  40
#define MAXBKS   10    /* maximum number of books */
char * s_gets(char * st, int n);
struct book {/* set up book template    */
    char title[MAXTITL];
    char author[MAXAUTL];
    float value;
};
int main(void){
    struct book library[MAXBKS];
    int count = 0;
    int index, filecount;
    FILE * pbooks;
    int size = sizeof (struct book);
    if ((pbooks = fopen("book.dat", "a+b")) == NULL)exit(1);
    rewind(pbooks); /* go to start of file */
    while (count < MAXBKS &&  fread(&library[count], size, 1, pbooks) == 1){
        if (count == 0)  puts("contents of book.dat:");
        printf("%s by %s: $%.2f\n",library[count].title,
               library[count].author, library[count].value);
        count++;
    }
    filecount = count;
    if (count == MAXBKS) exit(2);
    while (count < MAXBKS && s_gets(library[count].title, MAXTITL) != NULL && library[count].title[0] != '\0')  {
        s_gets(library[count].author, MAXAUTL);
        scanf("%f", &library[count++].value);
        while (getchar() != '\n') continue;
        if (count < MAXBKS)
            puts("Enter the next title.");
    }
    if (count > 0) {
        for (index = 0; index < count; index++)
            printf("%s by %s: $%.2f\n",library[index].title, library[index].author, library[index].value);
        fwrite(&library[filecount], size, count - filecount, pbooks);
    }
    else puts("No books? Too bad.\n");
    fclose(pbooks);
    return 0;
}
```

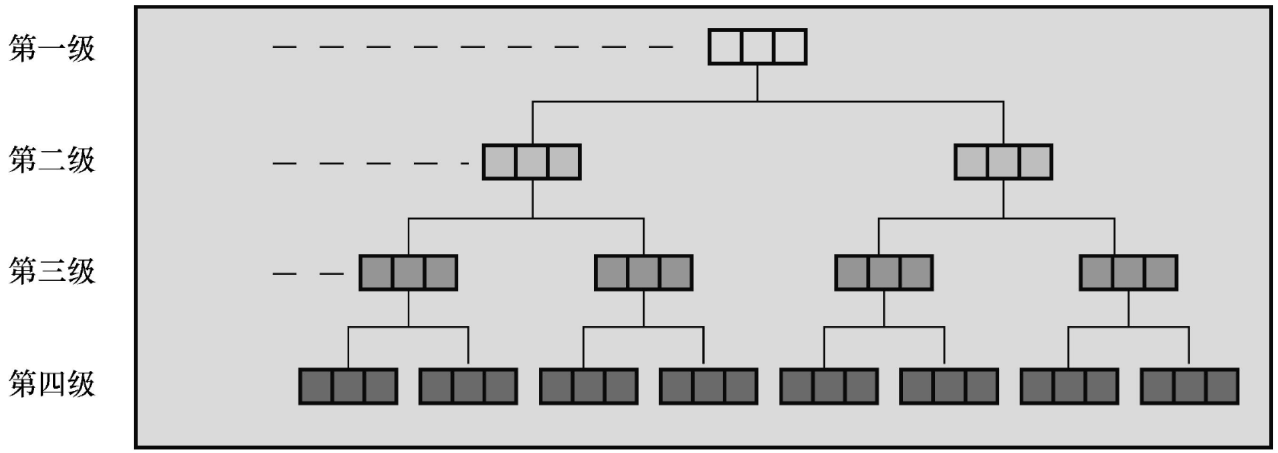
8.2 程序要点

- 使用 "a+b" 模式打开文件
  - a+部分允许程序读取整个文件并在文件的末尾添加内容
  - b是ANSI的一种标识方法，表明程序将使用二进制文件格式
    - 是value成员不是文本
- rewind() 确保文件位置指针处于文件开始部分，为开始读取做好准备

9 链式结构

9 链式结构

- 计算机用户已经开发出一些比我们提到过的数组和简单结构更适用于特定问题的数据形式．这些形式的名称有队列，二叉树，堆，哈希表和图．
- 许多这样的形式都由链式结构（linked structure）组成。通常，每个结构都包含一两个数据项和一两个指向其他同类型结构的指针。这些指针把一个结构和另一个结构链接起来，并提供一种路径能遍历整个彼此链接的结构。



10 联合简介

10 联合简介

- 联合（union）是一个能在同一个存储空间里（但不同时）存储不同类型数据的数据类型

- 一个典型的应用：一种表，设计它是用来以某种既没有规律，事先也未知的顺序保存混合类型数据
  - 使用联合类型的数组，可以创建相同大小单元的数组，每个单元都能存储多种类型的数据

- 联合只能存储一个值
  - 可以解释成其它的值，取决于用户

```
1.  union hold {
2.      int digit;
3.      double bigfl;
4.      char letter;
5.  };

6.  union hold valA;
7.  valA.letter = 'R';
8.  union hold valB = valA; // 用另一个联合来初始化
9.  union hold valC = {88}; // 初始化联合的digit 成员
10. union hold valD = {.bigfl = 118.2}; // 指定初始化器
```

10.1 使用联合

- 点运算符表示正在使用哪种数据类型
- 在联合中，一次只存储一个值
  - 即使有足够的空间，也不能同时存储一个char类型值和一个int类型值
- 用一个成员把值存储在一个联合中，然后用另一个成员查看内容，这种做法有时很有用

```
1. fit.digit = 23; //把 23 存储在 fit, 占2字节
2. fit.bigfl = 2.0; // 清除23, 存储 2.0, 占8字节
3. fit.letter = 'h'; // 清除2.0, 存储h, 占1字节

4. pu = &fit;
5. x = pu->digit; // 相当于 x = fit.digit
```

10.2 匿名联合（C11）

- 匿名联合是一个结构或联合的无名联合成员

```
1.  struct owner {
2.      char socsecurity[12];
3.      ...
4.  };
5.  struct leasecompany {
6.      char name[40];
7.      char headquarters[40];
8.      ...
9.  };
10. struct car_data {
11.     char make[15];
12.     int status; /* 私有为0, 租赁为1 */
13.     union {
14.             struct owner owncar;
15.             struct leasecompany leasecar;
16.     };
17.     ...
18. };
```

11 枚举类型

11 枚举类型

- 枚举类型（enumerated type）声明符号名称来表示整型常量
  - enum关键字，可以创建一个新“类型”并指定它可具有的值
  - 实际上，enum常量是int类型，只要能使用int类型的地方就可以使用枚举类型
  - 枚举类型的目的是提高程序的可读性
- 枚举符（enumerator）

```
1. enum spectrum {red, orange, yellow, green, blue,
   violet};
2. enum spectrum color;

3. int c;
4. color = blue;
5. if (color == yellow)
6.     ...;
7. for (color = red; color <= violet; color++)
8.     ...;
```

11.1 enum 常量

- blue 和 red 到底是什么？从技术上讲，它们是 int 类型的常量

- 在声明数组时，可以用枚举常量表示数组的大小
- 在switch语句中，可以把枚举常量作为标签

11.2 默认值

- 默认时，枚举列表中的常量被指定为 整数值 0，1，2 等等
  - `enum kids {nippy, slats, skippy, nina, 1iz };`

11.3 指定值

- `enum levels {low = 100, medium = 500, high = 2000};`
- 如果只给一个枚举常量赋值，没有对后面的枚举常量赋值，那么后面的常量会被赋予后续的值
- `enum feline {cat, lynx = 10, puma, tiger};`
  - 那么，cat的值是0（默认），lynx、puma和tiger的值分别是10、11、12。

11.4 enum用法

- 枚举类型是为了提高程序的可读性和可维护性
- 程序清单14.15 enum.c
- 当输入的字符串与color数组的成员指向的字符串相匹配时，for循环结束。如果循环找到匹配的颜色，程序就用枚举变量的值与作为case标签的枚举常量匹配

```
1. /* enum.c -- uses enumerated values */
2. #include <stdio.h>
3. #include <string.h>    // for strcmp(), strchr()
4. #include <stdbool.h>   // C99 feature
5. char * s_gets(char * st, int n);

6. enum spectrum {red, orange, yellow, green, blue,
   violet};
7. const char * colors[] = {"red", "orange", "yellow",
   "green", "blue", "violet"};
8. #define LEN 30
```

```
1.  int main(void){
2.      char choice[LEN];
3.      enum spectrum color;
4.      bool color_is_found = false;
5.      puts("Enter a color (empty line to quit):");
6.      while (s_gets(choice, LEN) != NULL && choice[0] !=
    '\0'){
7.          for (color = red; color <= violet; color++){
8.              if (strcmp(choice, colors[color]) == 0){
9.                  color_is_found = true;
10.                 break;
11.             }
12.         }
13.         if (color_is_found)
14.             switch(color){
15.             case red: puts("Roses are red.");
16.                 break;
17.         }
18.         else  printf("I don't know about %s.\n", choice);
19.         color_is_found = false;
20.         puts("Next color, please (empty line to quit):");
21.     }
22.     return 0;
23. }
```

11.5 共享的名字空间

- C使用术语名字空间（namespace）来表示识别一个名字的程序部分
  - 即通过名称来识别。作用域是名称空间概念的一部分：两个不同作用域的同名变量不冲突；两个相同作用域的同名变量冲突
    - `struct rect { double x; double y; };`
    - `int rect; / /` 在C 中不会产生冲突

12 typedef简介

12 typedef简介

- typedef 工具是一种高级数据特性，它使你能够为某一类型创建你自己的名字
- 与 #define 不同
  - 与 #define 不同，typedef 给出的符号名称仅限于对类型，而不是对值
  - typedef 的解释由编译器，而不是预处理器执行.
  - 虽然它的范围有限，但在其受限范围内，typedef 比 #define更灵活
- typedef常用于给复杂的类型命名

```
1. typedef unsigned char BYTE;

2. BYTE x, y[10], * z;

3. typedef struct {double x; double y;} rect;
4. rect r1 = {3.0, 6.0};
5. rect r2;

6. //把FRPTC声明为一个函数类型，该函数返回一个指针，该指针指向内含5个char类型元素的数组
7. typedef char (* FRPTC ()) [5];
```

13 其他复杂的声明

13 其他复杂的声明

- 当进行一个声明时，可以添加一个修饰符来修饰名称（或标识符）
- 表 14.1声明时可以使用的修饰符

Modifier	Significance
*	Indicates a pointer
( )	Indicates a function
[ ]	Indicates an array

```
1. int board[8][8]; //声明一个内含int数组的数组
2. int ** ptr; //声明一个指向指针的指针，被指向的指针指向int
3. int * risks[10]; //声明一个内含10个元素的数组，每个元素都是一个指向int的指针
4. int (* rusks)[10]; //声明一个指向数组的指针，该数组内含10个int类型的值
5. int * oof[3][4]; //声明一个3×4 的二维数组，每个元素都是指向int的指针
6. int (* uuf)[3][4]; //声明指向3×4二维数组的指针，该数组中内含int类型值
7. int (* uof[3])[4]; //声明内含3个指针元素的数组，每个指针指向内含4个int类型元素的数组
8. char * fump(int); //返回字符指针的函数
9. char (* frump)(int); //指向函数的指针，该函数的返回类型为char
10. char (* flump[3])(int); //内含3个指针的数组，每个指针指向返回类型为char的函数
11. typedef int arr5[5];
12. typedef arr5 * p_arr5;
13. typedef p_arr5 arrp10[10];
14. arr5 togs; // togs是内含5个int类型值的数组
15. p_arr5 p2; // p2是指向数组的指针，该数组内含5个int类型的值
16. arrp10 ap; // ap是内含10个指针的数组，每个指针都指向一个内含5个int类型值的数组
```

14 函数和指针

14 函数和指针

- 一个函数指针可以作为另一个函数的参数，告诉第二个函数使用哪一个函数
  - 如，选择合适的排序函数
- 函数也有地址
  - 因为函数的机器语言实现由载入内存的代码组成。指向函数的指针中存储着函数代码的起始处的地址
- 声明一个函数指针时，必须声明指针指向的函数类型
  - 为了指明函数类型，要指明函数签名，即函数的返回类型和形参类型

```
1.  void (*pf) (char *);   /* pf 是一个指向函数的指针 */
2.  (*pf) (mis);     /* 把 ToUpper 作用于 mis (语法1) */
3.  pf (mis);        /* 把 ToLower 作用于 mis (语法2) */

4.  void ToUpper(char *);
5.  void ToLower(char *);
6.  int round(double);
7.  void (*pf)(char *);
8.  pf = ToUpper; // 有效，ToUpper是该类型函数的地址
9.  pf = ToLower; //有效，ToLower是该类型函数的地址
10. pf = round; // 无效，round与指针类型不匹配
11. pf = ToLower(); // 无效，ToLower()不是地址
```

代码

- 程序清单14.16　func_ptr.c

关键概念

15　关键概念

- C结构可以把这些信息都放在一个单元内。在组织程序时这很重要，因为这样可以把相关的信息都存储在一处，而不是分散存储在多个变量中
  - 如果要在结构中增加一个成员，只需重写函数，不必改写函数调用。这在修改结构时很方便
- 联合声明与结构声明类似。但是，联合的成员共享相同的存储空间，而且在联合中同一时间内只能有一个成员

- enum工具提供一种定义符号常量的方法，typedef工具提供一种为基本或派生类型创建新标识符的方法
- 指向函数的指针提供一种告诉函数应使用哪一个函数的方法