张晓平

武汉大学数学与统计学院

Homepage: xpzhang.me

C 语言程序设计

结构体及其他类型

示例

- 结构体将几个类型的数据结合在一起，将其当成一个类型来处理。

  例如图书馆管理书籍，一本书包含编号、书名、单价等内容，于是把编号、书名、单价等结合成一个类型。
- 这样的数据类型是自定义的数据类型。

注意：结构体是数据类型，而不是变量，必须在程序中定义了该类型后才能操作数据。

示例 I

```
1  // book.c:
2  #include<stdio.h>
3  #define MAXTITL 41
4  #define MAXAUTL 31
5  struct book
6  {
7    char title[MAXTITL];
8    char author[MAXAUTL];
9    float value;
10 };
```

示例 II

```
11
12 int main(void)
13 {
14   struct book library;
15
16   printf("Please enter the book title.\n");
17   gets(library.title);
18
19   printf("Please enter the author.\n");
20   gets(library.author);
21
```

示例 III

```
22    printf("Now enter the value.\n");
23    scanf("%f", &library.value);
24
25    printf("%s by %s: %.2f\n", library.title,
26       library.author, library.value);
27    printf("%s: *%s* (%.2f)\n", library.title,
28       library.author, library.value);
29
30    return 0;
31 }
```

示例

```
Please enter the book title.
C primer plus
Please enter the author.
Stephan Prata
Now enter the value.
80
C primer plus by Stephan Prata: 80.00
C primer plus: *Stephan Prata* (80.00)
Done.
```

示例

该程序创建的结构由 3 部分组成，每个部分称为成员（member）。

建立结构体声明

结构体声明 (structure declaration) 是描述结构如何组合的主要方法，形如

```
struct book
{
  char title[MAXTITL];
  char author[MAXAUTL];
  float value;
};
```

该声明描述了一个结构体，它由两个char 数组和一个float 变量组成。该声明并没有创建一个实际的数据对象，而是描述了组成这类对象的元素。

建立结构体声明

- 关键字struct 后是一个可选的标记book，是用于引用该结构体的快速标记。以下声明

```
struct book library;
```

把library 声明为一个使用book 结构体的结构体变量。

- 接下来是用一对花括号扩起来的结构体成员列表。每个成员变量都用它自己的声明来描述，用一个分号来结束描述。每个成员可以是任意 C 类型，甚至可以是其他结构体。
- 结束花括号后的分号表示结构体声明的结束。

建立结构体声明

结构体声明可以放在任何函数的外面，也可以放在一个函数内部。

- 如果是内部声明，则该结构体只能在该函数内部使用。
- 如果是外部声明，则它可以被本文件该声明后的所有函数使用。

结构体变量

有了结构体声明，就可以创建一个结构体变量，如

```
struct book library;
```

这样，编译器会使用book 模板为该变量分配空间：

- 一个长度为MAXTITL 的char 数组
- 一个长度为MAXAUTL 的char 数组
- 一个float 变量

这些变量是以一个名字library 结合在一起的。

结构体变量

- 在结构体变量的声明中，struct book 是一种新的数据类型，就如同int 或float 一样。
- 你可以创建一个struct book 类型的变量，也可以创建一个指向该结构体的指针。如

```
struct book lib1, lib2, * ptbook;
```

结构体变量

就计算机而言，声明

```
1 struct book library;
```

是以下声明的简化

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

也就是说，声明结构体的过程与定义结构体变量的过程可以被合并成一步。

结构体变量

将结构体声明与结构体变量定义合并在一起，是不需要使用标记的一种情况

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

然而，如果想多次使用一个结构体模板，就需要使用带标记的形式。

结构体变量：初始化结构体变量

要初始化一个结构体变量，可以这么做

```
1 struct book library = {
2   "C primer plus",
3   "Stephan Prata",
4   80
5 };
```

即使用一个用花括号括起来的、用逗号隔开的初始化项目列表来进行初始化。

- 每个初始化项目必须与要初始化的结构体成员类型相匹配。
- 建议把每个成员的初始化项目写在单独的一行。

结构体变量：访问结构体成员

结构体像是一个“超级数组”，在这个超级数组内，一个元素可以是char类型，下一个元素可以是float 类型，再下一个可以是int 数组。

数组可以通过下标来访问每一个元素，那又如何访问结构体中的各个成员呢？

用结构体成员运算符(.)。

library.value 指的是library 的 value 成员，可以像使用任何其他float 变量那样使用library.value。

从本质上讲，.title、.author 和.value 在book 结构中扮演了下标的角色。

结构体变量：访问结构体成员

结构体像是一个“超级数组”，在这个超级数组内，一个元素可以是char类型，下一个元素可以是float 类型，再下一个可以是int 数组。

数组可以通过下标来访问每一个元素，那又如何访问结构体中的各个成员呢？

用结构体成员运算符(.)。

library.value 指的是library 的 value 成员，可以像使用任何其他float 变量那样使用library.value。

从本质上讲，.title、.author 和.value 在book 结构中扮演了下标的角色。

结构体变量：结构体的指定初始化项目

C99 支持结构体的指定初始化项目，使用点运算符和成员名来标识具体的元素。

- 只初始化book 结构体的成员value，可以这样做：

```
struct book surprise = {.value = 20.50};
```

- 可以按任意顺序使用指定初始化项目：

```
struct book gift = {
  .value = 40.50,
  .author = "Dennis M. Ritchie",
  .title = "The C programming language"
};
```

示例程序 I

```
1  // manybook.c:
2  #include<stdio.h>
3  #define MAXTITL 41
4  #define MAXAUTL 31
5  #define MAXBOOK 100
6  struct book
7  {
8    char title[MAXTITL];
9    char author[MAXAUTL];
10   float value;
```

示例程序 II

```
11 };
12
13 int main(void)
14 {
15   struct book library[MAXBOOK];
16   int count = 0;
17   int i;
18
19   printf("Enter the book title.\n");
20   printf("Press [enter] at the start of a line to
     stop.\n");
```

示例程序 III

```
21    while (count < MAXBOOK
22       && gets(library[count].title) != NULL
23       && library[count].title[0] != '\0') {
24      printf("Enter the author.\n");
25      gets(library[count].author);
26      printf("Enter the value.\n");
27      scanf("%f", &library[count++].value);
28      while(getchar() != '\n')
29        continue;
30      if (count < MAXBOOK)
31        printf("Enter the next title.\n");
```

示例程序 IV

```
32    }
33
34    if (count > 0) {
35      printf("Here is the list of your book:\n");
36      for (i = 0; i < count; i++)
37        printf("%s by %s: %.2f\n", library[i].title,
38            library[i].author, library[i].value);
39    } else {
40      printf("No book? Too bad!\n");
41    }
42
```

示例程序 V

```
43   return 0;
44 }
```

示例程序

```
Enter the book title.
Press [enter] at the start of a line to stop.
C primer plus[enter]
Enter the author.
Stephan Prata[enter]
Enter the value.
80[enter]
Enter the next title.
C programing language[enter]
Enter the author.
Dennis Ritchie[enter]
Enter the value
```

示例程序

```
Here is the list of your book:
C primer plus by Stephan Prata: 80.00
C programing language by Dennis Ritchie: 40.00
```

声明结构体数组

```
struct book library[MAXBOOK];
```

声明一个具有MAXBOOK 个元素的数组，每个元素都是一个book 类型的结构体。

注意：library 本身不是结构体变量名，它是元素类型为struct book 的数组名。

标识结构体数组的成员

为了标识结构体数组的成员，可这么做

```
library[0].value;       //第 1 个数组元素的 value 成员
library[4].title;       //第 5 个数组元素的 title 成员
library[2].title[4];    //第 3 个数组元素的 title 成员
                        // 的第 5 个字符
```

嵌套结构体

有时候，一个结构体中嵌套另一个结构体是很方便的。

如Shalala 创建一个有关他的朋友的信息的结构体。该结构体的一个成员是朋友的姓名，而姓名本身就可以标识为一个结构体，其中包含名和姓两个成员。

嵌套结构体 I

```
1  #include<stdio.h>
2  #define LEN 20
3
4  const char * msgs[5] =  {
5    "  Thank your for the wonderful evening, ",
6    "You certainly prove that a ",
7    "is a special kind of guy. We must get together",
8    "over a delicous ",
9    "and have a few laughs"
10 };
```

嵌套结构体 II

```
11
12 struct names {
13   char first[LEN];
14   char last[LEN];
15 };
16
17 struct guy {
18   struct names handle;
19   char favfood[LEN];
20   char job[LEN];
21   float income;
```

嵌套结构体 III

```
22 };
23
24 int main(void)
25 {
26   struct guy fellow = {
27     {"Ewen", "Villard"},
28     "grilled salmon",
29     "personality coach",
30     58812.0
31   };
32
```

嵌套结构体 IV

```
33    printf("Dear %s, \n\n", fellow.handle.first);
34    printf("%s%s.\n", msgs[0], fellow.handle.first);
35    printf("%s%s\n", msgs[1], fellow.job);
36    printf("%s\n", msgs[2]);
37    printf("%s%s%s", msgs[3], fellow.favfood, msgs[4]);
38    if (fellow.income > 150000.0)
39      puts("!!");
40    else if (fellow.income > 75000.0)
41      puts("!");
42    else
43      puts(".");
```

嵌套结构体 V

```
44
45    printf("\n%40s%s\n", " ", "See you soon, ");
46    printf("%40s%s\n", " ", "Shalala");
47
48    return 0;
49 }
```

嵌套结构体

```
Dear Ewen,

  Thank your for the wonderful evening, Ewen.
You certainly prove that a personality coach
is a special kind of guy. We must get together
over a delicous grilled salmonand have a few laughs.

                                See you soon,
                                Shalala
```

嵌套结构体

- 对嵌套结构的成员进行访问，只需使用两次点运算符：

```
fellow.handle.first
```

指向结构的指针

为什么要使用指向结构的指针，有以下三个原因：

1. 如同指向数组的指针比数组本身更容易操作一样，指向结构的指针通常比结构本身更容易操作；
2. 在一些早期的 C 实现中，结构不能作为参数被传递给函数，但指向结构的指针可以；
3. 许多奇妙的数据表示都使用了包含指向其他结构的指针的结构。

声明和初始化结构指针

声明如下：

```
struct guy * him;
```

该声明意味着指针him 现在可以指向任意guy 类型的结构。如果barney 是一个guy 类型的结构，可以这样做

```
him = &barney;
```

请注意：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。

声明和初始化结构指针

fellow 是一个结构数组，亦即fellow[0] 是一个结构，以下代码让him 指向fellow[0]：

```
him = &fellow[0];
```

使用指针访问结构成员

假设him 现在指向fellow[0]，有两种方式来访问它的成员：

1. 使用运算符：->。

```
him->income is fellow[0].income
if him == &fellow[0]
```

务必注意him 是个指针，而him->income 是him 所指向结构的一个成员。

2. 使用点运算符。

```
fellow[0].income  == (*him).income
```

必须使用圆括号，因点运算符的优先级比* 更高。

使用指针访问结构成员

假设him 现在指向fellow[0]，则以下表达式等价

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

向函数传递结构信息

向函数传递结构信息，有三种方式：

1. 将结构作为参数传递
2. 将指向结构的指针作为参数传递
3. 将结构成员作为参数传递

传递结构成员

若结构成员为基本类型（即int，char，float，double 或指针），就可将结构成员作为参数传递给函数。

传递结构成员

例

编制程序，把客户的银行账户加到他的储蓄与贷款账户中。

传递结构成员 I

```
1  // funds1.c:
2  #include<stdio.h>
3  #define LEN 50
4  struct funds {
5    char bank[LEN];
6    double bankfund;
7    char save[LEN];
8    double savefund;
9  };
10
```

传递结构成员 II

```
11 double sum(double, double);
12
13 int main(void)
14 {
15   struct funds stan = {
16     "Hankou Bank",
17     3024.32,
18     "Lucky's Savings and Loan",
19     9237.11
20   };
21
```

传递结构成员 III

```
22   printf("Stan has a total of %.2f.\n",
23       sum(stan.bankfund, stan.savefund));
24 }
25
26 double sum(double x, double y)
27 {
28   return (x + y);
29 }
```

传递结构成员

- sum() 既不知道也不关心实参是不是结构成员，它值要求参数是double 类型。
- 若想让被调函数影响调用函数中的成员值，可以传递成员地址：

```
modify(&stan.bankfund);
```

使用结构地址 I

```
1  #include<stdio.h>
2  #define LEN 50
3  struct funds {
4    char bank[LEN];
5    double bankfund;
6    char save[LEN];
7    double savefund;
8  };
9
10 double sum(const struct funds *);
```

使用结构地址 II

```
11
12 int main(void)
13 {
14   struct funds stan = {
15     "Hankou Bank",
16     3024.32,
17     "Lucky's Savings and Loan",
18     9237.11
19   };
20
21   printf("Stan has a total of %.2f.\n",
```

使用结构地址 III

```
22      sum(&stan));
23 }
24
25 double sum(const struct funds * money)
26 {
27   return (money->bankfund + money->savefund);
28 }
```

把结构作为参数传递 I

```
1  // funds3.c
2  #include<stdio.h>
3  #define LEN 50
4  struct funds {
5    char bank[LEN];
6    double bankfund;
7    char save[LEN];
8    double savefund;
9  };
10
```

把结构作为参数传递 II

```
11 double sum(const struct funds moolah);
12
13 int main(void)
14 {
15   struct funds stan = {
16     "Hankou Bank",
17     3024.32,
18     "Lucky's Savings and Loan",
19     9237.11
20   };
21
```

把结构作为参数传递 III

```
22    printf("Stan has a total of %.2f.\n",
23        sum(stan));
24 }
25
26 double sum(const struct funds moolah)
27 {
28    return (moolah.bankfund + moolah.savefund);
29 }
```

其他结构特性

现在的 C 允许把一个结构赋值给另一个结构。

例如，如果n_data 和o_data 是同一类型的结构，可以这样做

```
o_data = n_data;
```

这使得o_data 的每个成员都被赋成n_data 相应成员的值，即便其中有成员是数组。

其他结构特性

也可以把一个结构初始化为另一个同样类型的结构。

例如，

```
struct names right_field = {"Ruthie",
                            "George"};
struct names captin = right_field;
```

其他结构特性

现在的 C 中，结构不仅可作为参数传递给函数，也可以作为函数返回值返回。

把结构作为函数参数可以将结构信息传递给一个函数，使用函数返回结构可以将结构信息从被调用函数传递给调用函数。

同时，结构指针也允许双向通信，因此可使用任一种方法解决编程问题。

其他结构特性 I

```
1  // names1.c
2  #include <stdio.h>
3  #include <string.h>
4
5  struct namect {
6    char fname[20];
7    char lname[20];
8    int letters;
9  };
10
```

其他结构特性 II

```
11 void getinfo(struct namect *);
12 void makeinfo(struct namect *);
13 void showinfo(const struct namect *);
14
15 int main(void)
16 {
17   struct namect person;
18   getinfo (&person);
19   makeinfo(&person);
20   showinfo(&person);
21   return 0;
```

其他结构特性 III

```
22 }
23
24 void getinfo(struct namect * pst)
25 {
26   puts("Enter your first name.");
27   gets(pst->fname);
28   puts("Enter your last name.");
29   gets(pst->lname);
30 }
31
32 void makeinfo(struct namect * pst)
```

其他结构特性 IV

```
33 {
34   pst->letters = (int) strlen(pst->fname) +
35                  (int) strlen(pst->lname);
36 }
37
38 void showinfo(const struct namect * pst)
39 {
40   printf("%s %s, your name contains %d letters.\n",
41          pst->fname, pst->lname, pst->letters);
42 }
```

其他结构特性

```
Enter your first name.
Stepha
Enter your last name.
Prata
Stephan Prata, your name contains 12 letters.
```

其他结构特性

接下来看看如何使用结构参数和返回值来完成这个任务。

其他结构特性 I

```
1  // names2.c
2  #include <stdio.h>
3  #include <string.h>
4
5  struct namect {
6    char fname[20];
7    char lname[20];
8    int letters;
9  };
10
```

其他结构特性 II

```
11 struct namect getinfo(void);
12 struct namect makeinfo(struct namect);
13 void showinfo(struct namect);
14
15 int main(void)
16 {
17   struct namect person;
18
19   person = getinfo();
20   person = makeinfo(person);
21   showinfo(person);
```

其他结构特性 III

```
22
23    return 0;
24 }
25
26 struct namect getinfo(void)
27 {
28    struct namect temp;
29
30    puts("Enter your first name.");
31    gets(temp.fname);
32    puts("Enter your last name.");
```

其他结构特性 IV

```
33    gets(temp.lname);
34
35    return temp;
36  }
37
38  struct namect makeinfo(struct namect info)
39  {
40    info.letters = (int) strlen(info.fname) +
41                   (int) strlen(info.lname);
42    return info;
43  }
```

其他结构特性 V

```
44
45 void showinfo(struct namect info)
46 {
47   printf("%s %s, your name contains %d letters.\n",
48          info.fname, info.lname, info.letters);
49 }
```

结构，还是指向结构的指针

例

写一个有关结构的函数，使用结构指针作为参数，还是用结构作为参数和返回值呢？

都可以，但每种方法各有优点和不足。

结构，还是指向结构的指针

将结构指针作为参数，有两个优点：

1. 在新老的 C 实现上均可工作，且执行速度快；
2. 只需传递一个结构地址。

缺点：缺少对数据的保护。但 ANSI C 的关键词`const` 可解决这一问题。

结构，还是指向结构的指针

把结构作为参数传递，有如下优点：

1. 函数处理的是原始数据的副本，这比直接处理原始数据安全；
2. 编程风格更为清晰。

缺点有两个：

1. 早起的 C 实现可能不能工作，且浪费时间和空间；
2. 把一个大的结构传递给函数，而函数值使用其中一个或两个成员，尤其浪费时间和空间。这种情况下，传递指针或所需成员更为合理。

结构，还是指向结构的指针

假设定义了如下结构类型（可表示平面上的向量）

```
struct vector { double x; double y; };
```

要求两个向量 a 和 b 的和，可编写一个传递和返回结构的函数，形如

```
struct vector ans, a, b;
struct vector sum_vec(struct vector, struct vector);
...
ans = sum_vec(a, b);
```

结构，还是指向结构的指针

指针形式如下

```
struct vector ans, a, b;
struct vector sum_vec(const struct vector *,
                      const struct vector *, struct vector *);
...
sum_vec(&a, &b, &ans);
```

在指针形式中，用户必须记住总和的地址出现在参量列表的哪个位置。

结构，还是指向结构的指针

- 通常，程序员为了追求效率，会使用结构指针作为函数参数；当需要保护数据、防止意外修改数据时，对指针使用const 限定词。
- 而传递结构是处理小型结构最常用的办法。

在结构中使用字符数组还是字符指针

问题

能否将结构中的字符数组用指向字符的指针来代替？即如下结构声明

```
struct names {
  char first[20];
  char last [20];
}
```

能否改写成

```
struct pnames {
  char * first;
  char * last;
}
```

在结构中使用字符数组还是字符指针

问题

能否将结构中的字符数组用指向字符的指针来代替？即如下结构声明

```
struct names {
  char first[20];
  char last [20];
}
```

能否改写成

```
struct pnames {
  char * first;
  char * last;
}
```

在结构中使用字符数组还是字符指针

考虑如下代码

```
struct names name1 = {"Stephan", "Prata"};
struct pnames name2 = {"Dennis", "Ritche"};
printf("%s %s", name1.first, name2.last);
```

这是一段正确的代码，也能运行正常，但想想字符串存储在哪里。

在结构中使用字符数组还是字符指针

- 对于struct names 变量name1，字符串存储在结构内部；该结构分配了 40 个字节来存放两个字符串。
- 对于struct pnames 变量name2，字符串存储在编译器存储字符串常量的任何地方。该结构存放的只是两个地址，总共栈 16 个字节。（注：所有类型的指针变量在 32 位系统上都是 4 字节，64 位系统上都是 8 字节。）

  pnames 结构不为字符串分配任何存储空间，其中的指针应该只管理那些已创建的而在程序其他地方已经分配过空间的字符串。

在结构中使用字符数组还是字符指针

考虑如下代码

```
struct names accountant;
struct pnames attorney;
puts("Enter the last name of you accountant");
scanf("%s", accountant.last);
puts("Enter the last name of you attorney");
scanf("%s", attorney.last);    //存在潜在危险
```

在结构中使用字符数组还是字符指针

- 对于会计师，他的名字存储在accountant 的最后一个成员中。
- 对于律师，scanf() 把字符串放在由attorney.last 给出的地址中，而该地址未被初始化，可能为任意值，程序就可以把名字放在任何地方。

在结构中使用字符数组还是字符指针

- 如果需要一个结构来存储字符串，请使用字符数组成员。
- 若想在结构中使用指针处理字符串，请与`malloc()`搭配使用。

结构、指针和 malloc()

在结构中使用指针处理字符串时，可用`malloc()`分配内存。该方法的优点是可以请求`malloc()`分配刚好满足字符串需要数量的空间。

结构、指针和 malloc() I

```
1  // names3.c
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  struct namect {
7    char *fname;
8    char *lname;
9    int letters;
10 };
```

结构、指针和 malloc() II

```
11
12 void getinfo(struct namect *);
13 void makeinfo(struct namect *);
14 void showinfo(const struct namect *);
15 void cleanup(struct namect *);
16
17 int main(void)
18 {
19   struct namect person;
20   getinfo (&person);
21   makeinfo(&person);
```

结构、指针和 malloc() III

```
22    showinfo(&person);
23    cleanup (&person);
24    return 0;
25  }
26
27  void getinfo(struct namect * pst)
28  {
29    char temp[81];
30    puts("Enter your first name.");
31    gets(temp);
32    pst->fname = (char *) malloc (strlen(temp)+1);
```

结构、指针和 malloc() IV

```
33    strcpy(pst->fname, temp);
34    puts("Enter your last name.");
35    gets(temp);
36    pst->lname = (char *) malloc (strlen(temp)+1);
37    strcpy(pst->lname, temp);
38  }
39
40  void makeinfo(struct namect * pst)
41  {
42    pst->letters = (int) strlen(pst->fname) +
43                   (int) strlen(pst->lname);
```

结构、指针和 malloc() V

```
44 }
45
46 void showinfo(const struct namect * pst)
47 {
48   printf("%s %s, your name contains %d letters.\n",
49          pst->fname, pst->lname, pst->letters);
50 }
51
52 void cleanup(struct namect * pst)
53 {
54   free(pst->fname);
```

结构、指针和 malloc() VI

```
55    free(pst->lname);
56 }
```

结构、指针和 malloc()

- 必须理解两个字符串并没有存储在结构中，而是被保存在由`malloc()`管理的内存块中。
- 两个字符串的地址被存储在结构中，而这些地址正好是字符串函数所需要知道的。
- 调用`malloc()`后应该调用`free()`，故程序添加了一个`cleanup()`，在程序使用完内存后释放内存。

联合

联合 (union) 是一个使用同一存储空间（但不同时）存储不同数据的数据类型。

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

联合以与结构同样的方式建立，需要一个联合模板和一个联合变量。以下是一个创建带标记的联合模板的例子：

```
union hold {
  int digit;
  double bigfl;
  char letter;
}
```

该联合可以含有一个int 值、一个double 值或一个char 值。

联合

以下是定义 3 个 hold 类型联合变量的例子

```
union hold fit;
union hold save[10];
union hold * pu;
```

- 第一个声明创建一个变量fit。编译器分配足够多的空间一保存所描述的可能性的最大需要。在此情况下，最大可能性是double 数据，需要 8 个字节。
- 第二个声明创建一个save 数组，含 10 个元素，每个元素占 8 个字节。
- 第三个声明创建一个指针，可以存放一个hold 联合的地址。

联合

可以初始化一个联合。因联合只存储一个值，故其初始化规则与结构的初始化不同，它有三种选择：

1. 可以把一个联合初始化为同类型的另一个联合；
2. 可以初始化联合的第一个元素；
3. 按照 C99 标准，可以使用一个指定初始化项目。

```
union hold valA;
valA.letter = 'R';
union hold valB = valA;
union hold valC = {88};
union hold valD = {.bigfl = 118.2};
```

联合

以下代码说明了如何使用联合：

```
union hold fit;
fit.digit = 23;     //把 23 存储在 fit 中，使用 2 字节
fit.bigfl = 2.0;    //清除 23，存储 2.0，使用 8 字节
fit.letter = 'h';   //清除 2.0，存储'h'，使用 1 字节
```

点运算符表示正在使用哪种数据类型。在同一时间只能存储一个值。

联合

可以与指向联合的指针一样使用-> 运算符：

```
union hold * pu;
union hold fit;
pu = & fit;
x = pu->digit;    //相当于 x = fit.digit
```

接下来的语句告诉你什么是不能做的：

```
fit.letter = 'A';
flnum = 3.2 * fit.bigfl;    //错误
```

联合：联合的应用

假定有一个表示一辆汽车的结构。如果是私车，就要一个结构成员来描述汽车所有者；如果是租车，需要一个成员来描述租赁公司。

```
struct owner {
  char socsecurity[20];
  ...
};

struct leasecompany {
  char name[40];
  char headquarters[40];
  ...
}
```

联合：联合的应用

```
union data {
  struct owner owncar;
  struct leasecompany leasecar;
};

struct car_data{
  char make[15];
  int status;  // 0 = 私有, 1 = 租赁
  union data owerinfo;
  ...
}
```

联合：联合的应用

假定honda 是一个car_data 结构，则

- 若honda.status 为0，则程序可使用honda.owerinfo.owncar.socsecurity;
- 若honda.status 为1，则程序可使用honda.owerinfo.leasecar.name;

枚举类型

可使用枚举类型 (enumerated type) 声明代表整数常量的符号名称。用关键字`enum`，可创建一个新“类型”并指定它可以具有的值。实际上，`enum`常量是`int` 类型，故在使用`int` 类型的任何地方都可使用它。

枚举类型的目的是为了提高程序可读性，其语法与结构相同。

枚举类型

声明方式如下：

```
enum spectrum {red, orange, yellow, green, blue};
enum spectrum color;
```

- 第一个声明设置spectrum 为标记名，从而允许你把enum spectrum 作为一个类型名使用。
- 第二个声明使得color 成为该类型的一个变量。花括号中的标识符枚举了spectrum 变量可能有的值。

枚举类型

可以使用以下语句：

```
int c;
color = blue;
if (color == yellow)
  ...
for (color = red; color <= blue; color++)
  ...
```

实际上，为spectrum 枚举的常量在0 到5 之间。

枚举类型

执行以下代码：

```
printf("red = %d, orange = %d\n", red, orange);
```

结果为

```
red = 0, orange = 1
```

red 为一个代表整数0 的命名常量，其他标识符分别是代表1 到5 的命名常量。

枚举类型

默认时，枚举列表中的常量被指定为整数值0、1、2 等。故，以下声明使得nina 具有值3：

```
enum kids {nippy, slats, skippy, nina, liz};
```

枚举类型

- 也可指定常量具有特定的整数值：

```
enum levels {low = 100, medium = 500, high =
2000};
```

- 若只对一个常量赋值，而没对后面的常量赋值，则后面的常量会被赋予后续的值：

```
enum feline {cat, lynx = 10, puma, tiger};
```

则cat 的默认值为0，lynx、puma、tiger 的默认值分别为10、11、12。

枚举类型：enum 的用法

枚举类型的目的是为了提高程序可读性。如果是处理颜色，采用`red`和`blue` 要比使用`0` 和`1` 更显而易见。

枚举类型：enum 的用法 I

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdbool.h>
4
5 enum spectrum {red, orange, yellow, green, blue,
  violet};
6 const char * colors[] = { "red", "orange", "yellow",
7               "green", "blue", "violet" };
8 #define LEN 30
9
```

枚举类型：enum 的用法 II

```
10 int main(void)
11 {
12   char choice[LEN];
13   enum spectrum color;
14   bool color_is_found = false;
15
16   puts("Enter a color (empty line to quit):");
17   while (gets(choice) != NULL && choice[0] != '\0') {
18     for (color = red; color <= violet; color++) {
19       if (strcmp(choice, colors[color]) == 0) {
20         color_is_found = true;
```

枚举类型：enum 的用法 III

```
21            break;
22        }
23    }
24    if (color_is_found)
25        switch (color) {
26        case red :
27            puts("Roes are red.");
28            break;
29        case orange :
30            puts("Poppies are orange.");
31            break;
```

枚举类型：enum 的用法 IV

```
32        case yellow :
33          puts("Sunflowers are yellow.");
34          break;
35        case green :
36          puts("Grass is green.");
37          break;
38        case blue :
39          puts("Bluebells are blue.");
40          break;
41        case violet :
42          puts("Violets are violet.");
```

枚举类型：enum 的用法 V

```
43          break;
44        }
45      else
46        printf("I don't know about the color %s.\n",
          choice);
47      color_is_found = false;
48      puts("Next color, please (empty line to quit): ")
        ;
49    }
50    puts("Goodbye!");
51
```

枚举类型：enum 的用法 VI

```
52   return 0;
53 }
```

枚举类型：enum 的用法 I

```
Enter a color (empty line to quit):
orange
Poppies are orange.
Next color, please (empty line to quit):
blue
Bluebells are blue.
Next color, please (empty line to quit):
red
Roes are red.
Next color, please (empty line to quit):
```

枚举类型：enum 的用法 II

```
sdf
I don't know about the color sdf.
Next color, please (empty line to quit):

Goodbye!
```

typedef 简介

typedef 工具是一种高级数据特性，它使你能够为某一种类型创建自己的名字。它与#define 相似，但有如下不同

- 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 简介

观察代码

```
typedef unsigned char BYTE;
BYTE x, y[10], * z;
```

- 该代码为unsigned char 创建了一个名字BYTE，接下来便可用BYTE 来定义变量。
- 该定义的作用域取决于typedef 语句所在的位置。如果定义在一个函数内部，则其作用域是局部的，限定在该函数内。若定义在函数外部，则具有全局作用域。

typedef 简介

```
typedef char * STRING;
```

使STRING 成为char 指针的标识符。因此

```
STRING name, sign;
```

的意思是

```
char * name, * sign;
```

typedef 简介

若这样做：

```
#define STRING char *;
```

则

```
STRING name, sign;
```

的意思是

```
char * name, sign;
```

typedef 简介

也可对结构使用typedef:

```
typedef struct complex {
  float real;
  float imag;
} COMPLEX;
```

这样你就可以使用COMPLEX 来代替struct complex 来表示复数。

使用typedef 的原因之一是为经常出现的类型创建一个方便的、可识别的名称。

typedef 简介

使用typedef 来命名一个结构类型时，可省去结构的标记

```
typedef struct  {
  double x;
  double y;
} vector;
```

typedef 简介

然后，以下代码

```
vector v1 = {3.0, 6.0};
vector v2;
v2 = v1;
```

会被翻译成

```
struct { double x; double y; } v1 = {3.0, 6.0};
struct { double x; double y; } v2;
v2 = v1;
```

typedef 简介

使用typedef 的另一个原因是typedef 的名称经常被用于复杂的类型。如

```
typedef char(* FRPTC()) [5];
```

这把FRPTC 声明为一个函数类型，该类型的函数返回一个指向含 5 个元素的char 数组的指针。

typedef 简介

切记：使用typedef 并不创建新的类型，它只是创建了便于使用的标签。

奇特的声明

```
int board[8][8];      //int 数组的数组
int ** ptr;           //指向 int 的指针的指针
int * risk[10];
    //具有 10 个元素的数组，每个元素是一个指向 int 的指针
int (* rusk) [10];
    //一个指针，指向具有 10 个元素的 int 数组
int * oof[3][4];
    //一个 3x4 的数组，每个元素是一个指向 int 的指针
int (* uuf) [3][4];
    //一个指针，指向 3X4 的 int 数组
int (* uof [3]) [4];
    //一个具有 3 个元素的数组，每个元素是一个指向
```

奇特的声明

```
char * fump();
   //返回指向 char 的指针的函数
char (* frump) ();
   //指向返回类型为 char 的函数的指针
char (* flump[3]) ();
   //由 3 个指针组成的数组，每个指针指向返回值为 char 的函数
```

奇特的声明

```
typedef int arr5[5]
typedef arr5 * p_arr5;
typedef p_arr5 arrp10[10];
arr5 togs;
  //togs 为含 5 个元素的 int 数组
p_arr5 p2;
  //p2 为一个指针, 指向具有 5 个元素的 int 数组
arrp10 ap;
  //ap 是具有 10 个元素的指针数组,
  //每个指针指向具有 5 个元素的 int 数组
```

指针函数

先看下面的函数声明，注意，此函数有返回值，返回值为int *，即返回值是指针类型的。

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

指针函数 I

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int * f(int a, int b);
4  int main(void)
5  {
6    int * p1 = NULL;
7    printf("The memeory address of p1 = %p \n", p1);
8    p1 = f(1, 2);
9    printf("The memeory address of p1 = %p \n", p1);
10   printf("*p1 = %d \n", *p1);
```

指针函数 II

```
11    return 0;
12 }
13
14 int * f(int a, int b) {
15    int * p = (int *)malloc(sizeof(int));
16
17    printf("The memeory address of p = %p \n", p);
18    *p = a + b;
19    printf("*p = %d \n", *p);
20
21    return p;
```

指针函数 III

```
22 }
```

指针函数

```
The memeory address of p1 = (nil)
The memeory address of p = 0x12c0010
*p = 3
The memeory address of p1 = 0x12c0010
*p1 = 3
```

函数指针

函数指针说的就是一个指针，但这个指针指向函数，不是普通的基本数据类型或者类对象。

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

- 函数指针与指针函数的最大区别是函数指针的函数名是一个指针，即函数名前面有一个*。
- 上面的函数指针定义为一个指向一个返回值为整型，有两个参数并且两个参数的类型都是整型的函数。

函数指针 I

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int max(int a, int b)
5  {
6    return (a > b ? a : b);
7  }
8
9  int min(int a, int b)
10 {
```

函数指针 II

```
11    return (a < b ? a : b);
12  }
13
14  int operation(int a, int b, int (*op)())
15  {
16    return (*op)(a, b);
17  }
18
19  int main(void)
20  {
21    int (*f) (int, int);
```

函数指针 III

```
22
23    f = max;
24    printf("The max value is %d \n", operation(1, 2,
      max));
25
26    f = min;
27    printf("The min value is %d \n", f(1, 2));
28
29    return 0;
30 }
```

函数指针

```
The max value is 2
The min value is 1
```