C/C++

结构体、共用体、枚举体

张晓平

武汉大学数学与统计学院

Table of contents

示例：创建图书目录

示例：创建图书目录

C 结构体

在 C 语言中，结构体 (struct) 是由一系列具有相同类型或不同类型的数据构成的数据集合。

- 结构体是 C 语言的一种复合数据类型。
- 结构体可被声明为变量、指针或数组等，用以实现较复杂的数据结构。
- 结构体同时也是一些元素的集合，这些元素称为结构体的成员，且这些成员可以为不同的类型。

示例：创建图书目录

C 结构体

在 C 语言中，结构体 (struct) 是由一系列具有相同类型或不同类型的数据构成的数据集合。

- 结构体是 C 语言的一种复合数据类型。
- 结构体可被声明为变量、指针或数组等，用以实现较复杂的数据结构。
- 结构体同时也是一些元素的集合，这些元素称为结构体的成员，且这些成员可以为不同的类型。

例

- 一本书包含书名、作者、单价等内容
- 学生包含姓名、性别、学号、成绩等信息

示例：创建图书目录 i

```
// book.c:
#include<stdio.h>
#define MAX_TITLE 41
#define MAX_AUTHOR 31
struct book
{
  char title[MAX_TITLE];
  char author[MAX_AUTHOR];
  float price;
};
int main(void)
{
  struct book book;
  printf("Please enter the book title.\n");
  gets(book.title);
  printf("Please enter the author.\n");
```

示例：创建图书目录 ii

```
  gets(book.author);
  printf("Now enter the price.\n");
  scanf("%f", &book.price);
  printf("%s by %s: %.2f\n", book.title,
      book.author, book.price);
  return 0;
}
```

示例：创建图书目录

```
Please enter the book title.
C primer plus
Please enter the author.
Stephan Prata
Now enter the price.
90
C primer plus by Stephan Prata: 90.00
```

示例：创建图书目录

```
struct book
{
   char title[MAX_TITLE];
   char author[MAX_AUTHOR];
   float price;
};
```

注

- 结构体 `struct book` 由 3 个元素组成，每个元素称为成员 (member)。
- 结构体**是数据类型，不是变量**，必须先定义该类型然后才能定义相应的变量。

建立结构体声明

建立结构体声明

结构体声明（structure declaration）用于描述结构的组合方式，如

```
struct book
{
  char title[MAX_TITLE];
  char author[MAX_AUTHOR];
  float price;
};
```

建立结构体声明

结构体声明（structure declaration）用于描述结构的组合方式，如

```
struct book
{
  char title[MAX_TITLE];
  char author[MAX_AUTHOR];
  float price;
};
```

- 该声明定义了一个结构体，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。因此，结构体的定义不分配空间，只有结构体变量才分配空间。

建立结构体声明

- `struct` 后是一个可选标记 `book`，是用于引用该结构体的快速标记。声明

```
struct book book;
```

  把 `book` 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括起来的成员列表。
  - 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，也可以是其他结构体。
- 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。

建立结构体声明：使用 typedef

可使用 typedef 定义结构体：

```
typedef struct {
  char title[MAX_TITLE];
  char author[MAX_AUTHOR];
  float price;
} Book;
```

建立结构体声明：使用 typedef

可使用 typedef 定义结构体：

```
typedef struct {
  char title[MAX_TITLE];
  char author[MAX_AUTHOR];
  float price;
} Book;
```

然后定义结构体变量

```
Book book;
```

这样做的好处是，可省略掉可选标记，结构体名更为简洁，使用更加方便。请尽量使用这种方式定义结构体。

建立结构体声明 i

```
// book1.c:
#include<stdio.h>
#define MAX_TITLE 41
#define MAX_AUTHOR 31
typedef struct
{
  char title[MAX_TITLE];
  char author[MAX_AUTHOR];
  float price;
} Book;
int main(void)
{
  Book book;
  printf("Please enter the book title.\n");
  gets(book.title);
  printf("Please enter the author.\n");
```

建立结构体声明 ii

```
  gets(book.author);
  printf("Now enter the price.\n");
  scanf("%f", &book.price);
  printf("%s by %s: %.2f\n", book.title,
     book.author, book.price);
  return 0;
}
```

定义结构体变量

定义结构体变量

有了结构体声明

```
typedef struct {
  char title[MAX_TITLE];
  char author[MAX_AUTHOR];
  float price;
} Book;
```

就可以创建一个结构体变量，如

```
Book book;
```

这样，编译器会使用结构体 Book 为变量 book 分配空间：

- 一个长度为 MAX_TITLE 的 char 数组
- 一个长度为 MAX_AUTHOR 的 char 数组
- 一个 float 变量

这些变量以一个名字 book 结合在一起。

定义结构体变量

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

```
Book book1, book2, * pbook;
```

定义结构体变量

就计算机而言，声明

```
Book book;
```

是以下声明的简化

```
struct
{
   char title[MAX_TITLE];
   char author[MAX_AUTHOR];
   float price;
} book;
```

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。

定义结构体变量

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

```
struct
{
   char title[MAX_TITLE];
   char author[MAX_AUTHOR];
   float price;
} book;
```

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。

定义结构体变量

初始化结构体变量

定义结构体变量：初始化结构体变量

要初始化一个结构体变量，可以这么做

```
Book library = {
  "C primer plus",
  "Stephan Prata",
  80
};
```

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。

定义结构体变量

访问结构体成员

定义结构体变量：访问结构体成员

问题

对于数组，可通过下标来访问其每一个元素。那如何访问结构体中的各个成员呢？

定义结构体变量：访问结构体成员

问题

对于数组，可通过下标来访问其每一个元素。那如何访问结构体中的各个成员呢？

用结构体成员运算符 .。

定义结构体变量：访问结构体成员

问题

对于数组，可通过下标来访问其每一个元素。那如何访问结构体中的各个成员呢？

用结构体成员运算符 .。

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

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

定义结构体变量

结构体的指定初始化项目

定义结构体变量：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 的成员 price，可以这样做：

```
Book surprise = {.price = 20.50};
```

- 可按任意顺序指定初始化项目：

```
Book gift = {
  .price = 40.50,
  .author = "Dennis M. Ritchie",
  .title = "The C programming language"
};
```

结构体数组

结构体数组

示例程序

示例程序 i

```
// manybook.h:
#include<stdio.h>
#define MAX_TITLE 41
#define MAX_AUTHOR 31
#define MAX_BOOK 100
typedef struct
{
  char title[MAX_TITLE];
  char author[MAX_AUTHOR];
  float price;
} Book;
```

示例程序 i

```
// manybook.c:
#include "manybook.h"
int main(void)
{
  Book book[MAX_BOOK];
  int count = 0;
  int i;
  printf("Enter the book title.\n");
  printf("Press [enter] at the start of a line to stop
  .\n");
  while (count < MAX_BOOK
     && gets(book[count].title) != NULL
     && book[count].title[0] != '\0') {
    printf("Enter the author.\n");
    gets(book[count].author);
    printf("Enter the price.\n");
```

示例程序 ii

```
    scanf("%f", &book[count++].price);
    while(getchar() != '\n')
      continue;
    if (count < MAX_BOOK)
      printf("Enter the next title.\n");
  }
  if (count > 0) {
    printf("Here is the list of your book:\n");
    for (i = 0; i < count; i++)
      printf("%s by %s: %.2f\n", book[i].title,
          book[i].author, book[i].price);
  } else {
    printf("No book? Too bad!\n");
  }
  return 0;
}
```

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 price.
80[enter]
Enter the next title.
C programing language[enter]
Enter the author.
Dennis Ritchie[enter]
Enter the price.
40[enter]
Enter the next title.
```

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
```

结构体数组

声明结构体数组

声明结构体数组

```
Book book[MAX_BOOK];
```

声明一个长度为 MAX_BOOK 的数组，每个元素都是 Book 类型的结构体。

注

book 本身不是结构体变量名，它是元素类型为 Book 的数组名。

结构体数组

标识结构体数组的成员

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

```
book[0].price;        //第 1 个数组元素的 price 成员
book[4].title;        //第 5 个数组元素的 title 成员
book[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

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。

例

John 创建一个有关他朋友信息的结构体。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

```
// friend.h
#include<stdio.h>
#define LEN 20
const char * msgs[5] =  {
  "  Thank your for the wonderful evening, ",
  "You certainly prove that a ",
  "is a special kind of guy. We must get together",
  "over a delicous ",
  "and have a few laughs"
};
typedef struct {
  char first[LEN];
  char last[LEN];
} Name;
typedef struct {
  Name name;
```

嵌套结构体 ii

```
    char favfood[LEN];
    char job[LEN];
    float income;
} Guy;
```

嵌套结构体 i

```
// friend.c
#include "friend.h"
int main(void)
{
  Guy guy = {
    {"Ewen", "Villard"},
    "grilled salmon",
    "personality coach",
    158812.0
  };
  printf("Dear %s, \n\n", guy.name.first);
  printf("%s%s.\n", msgs[0], guy.name.first);
  printf("%s%s\n", msgs[1], guy.job);
  printf("%s\n", msgs[2]);
  printf("%s%s%s", msgs[3], guy.favfood, msgs[4]);
  if (guy.income > 150000.0)
```

嵌套结构体 ii

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

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,
                                        John
```

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：

```
guy.name.first
```

指向结构体的指针

指向结构体的指针

为什么要使用指向结构体的指针，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，指向结构体的指针通常比结构体本身更容易操作；
2. 在一些早期的 C 实现中，结构体不能作为参数被传递给函数，但指向结构体的指针可以；
3. 许多奇妙的数据表示都使用了包含指向其他结构体的指针的结构体。

指向结构体的指针

声明和初始化结构体指针

声明和初始化结构体指针

声明

```
Guy * him;
```

意味着指针 him 可以指向任意 Guy 类型的结构体。如果 john 是一个 Guy 类型的结构体，可以这样做

```
him = &john;
```

声明和初始化结构体指针

声明

```
Guy * him;
```

意味着指针 him 可以指向任意 Guy 类型的结构体。如果 john 是一个 Guy 类型的结构体，可以这样做

```
him = &john;
```

注

和数组不同，一个结构体的名字不是该结构体的地址，必须使用 & 运算符。

声明和初始化结构体指针

设 fellow 是一个结构体数组，即

```
Guy  fellow[5];
```

则fellow[0] 是一个结构体，以下代码让 him 指向 fellow[0]：

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

指向结构体的指针

使用指针访问结构体成员

使用指针访问结构体成员

若 him 指向 fellow[0]，则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 所指向结构体的一个成员。

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。

使用指针访问结构体成员

设 him 指向 fellow[0]，则以下表达式等价

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

向函数传递结构体信息

向函数传递结构体信息

向函数传递结构体信息，有三种方式：

1. 将结构体成员作为参数传递
2. 将结构体本身作为参数传递
3. 将指向结构体的指针作为参数传递

向函数传递结构体信息

传递结构体成员

传递结构体成员

若结构体成员为基本类型（即 `int`，`char`，`float`，`double` 或指针），可将结构体成员作为参数直接传递给函数。

传递结构体成员

若结构体成员为基本类型（即 `int`，`char`，`float`，`double` 或指针），可将结构体成员作为参数直接传递给函数。

例

设一个学生有姓名、数学、物理、化学等信息，已知三科成绩计算总成绩。

传递结构体成员

```
// student.h
#include <stdio.h>
#define LEN 50
typedef struct {
  char name[LEN];
  int math, phys, chem;
} Student;
```

传递结构体成员

```
#include "student.h"
int total(int math, int phys, int chem);
int main(void)
{
  Student std = {"Zhang San", 90, 85, 80};
  printf("Total score of %s is %d.\n", std.name,
      total(std.math, std.phys, std.chem));
}
int total(int math, int phys, int chem)
{
  return math + phys + chem;
}
```

向函数传递结构体信息

使用结构体地址

使用结构体地址

```
// student2.c
#include "student.h"
int total(const Student *);
int main(void)
{
  Student std = {"Zhang San", 90, 85, 80};
  printf("Total score of %s is %d.\n",
      std.name, total(&std));
}
int total(const Student * std)
{
  return std->math + std->phys + std->chem;
}
```

向函数传递结构体信息

把结构体作为参数传递

把结构体作为参数传递

```c
// student3.c
#include "student.h"
int total(const Student);
int main(void)
{
  Student std = {"Zhang San", 90, 85, 80};
  printf("Total score of %s is %d.\n",
      std.name, total(std));
}
int total(const Student std)
{
  return std.math + std.phys + std.chem;
}
```

向函数传递结构体信息

其他结构体特性

其他结构体特性

C 新标准允许把一个结构体赋值给另一个结构体。

例

如果 n_data 和 o_data 是同一类型的结构体，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。

其他结构体特性

也可以把一个结构体初始化为另一个同样类型的结构体。

例如，

```
Name name1 = {"Ruthie",
              "George"};
Name name2 = name1;
```

其他结构体特性

C 新标准中，结构体不仅可作为参数传递给函数，也可以作为函数返回值返回。

- 把结构体作为函数参数可以将结构体信息传递给一个函数
- 使用函数返回结构体可以将结构体信息从被调用函数传递给调用函数
- 结构体指针也允许双向通信

其他结构体特性 i

```
// names.h
#include <stdio.h>
#include <string.h>
typedef struct {
  char first[20];
  char last[20];
  int letters;
} Name;
void getinfo(Name *);
void makeinfo(Name *);
void showinfo(const Name *);
```

其他结构体特性 i

```
// info.c
#include "names.h"
void getinfo(Name * pname)
{
  puts("Enter your first name.");
  gets(pname->first);
  puts("Enter your last name.");
  gets(pname->last);
}
void makeinfo(Name * pname)
{
  pname->letters = (int) strlen(pname->first) +
    (int) strlen(pname->last);
}
void showinfo(const Name * pname)
{
```

其他结构体特性 ii

```
  printf("%s %s, your name contains %d letters.\n",
         pname->first, pname->last, pname->letters);
}
```

其他结构体特性 i

```
// main.c
#include "names.h"
int main(void)
{
  Name person;
  getinfo (&person);
  makeinfo(&person);
  showinfo(&person);
  return 0;
}
```

其他结构体特性

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

其他结构体特性

问题

如何使用结构体参数和返回值来完成这个任务。

其他结构体特性 i

```
// names.h
#include <stdio.h>
#include <string.h>
typedef struct {
  char first[20];
  char last[20];
  int letters;
} Name;
Name getinfo(void);
Name makeinfo(Name);
void showinfo(Name);
```

其他结构体特性 i

```
// info.c
#include "names.h"
Name getinfo(void)
{
  Name name;
  puts("Enter your first name.");
  gets(name.first);
  puts("Enter your last name.");
  gets(name.last);
  return name;
}
Name makeinfo(Name name)
{
  name.letters = (int) strlen(name.first) +
    (int) strlen(name.last);
  return name;
```

其他结构体特性 ii

```
}
void showinfo(Name name)
{
  printf("%s %s, your name contains %d letters.\n",
         name.first, name.last, name.letters);
}
```

其他结构体特性 i

```c
// main.c
#include "names.h"
int main(void)
{
  Name person;
  person = getinfo();
  person = makeinfo(person);
  showinfo(person);
  return 0;
}
```

向函数传递结构体信息

结构体，还是指向结构体的指针

结构体，还是指向结构体的指针

问题

写一个有关结构体的函数，使用结构体指针作为参数，还是用结构体作为参数和返回值呢？

都可以，但每种方法各有优点和不足。

结构体，还是指向结构体的指针

将结构体指针作为参数

优点

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

缺点

缺少对数据的保护。但可使用 `const` 解决这一问题。

结构体，还是指向结构体的指针

把结构体作为参数传递

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

1. 早期的 C 编译器可能不能编译
2. 实参传给形参是一个复制的过程，故把一个大的结构体传递给函数需要拷贝一份，这非常浪费时间和空间，尤其是当函数只使用了它的一个或两个成员。这种情况下，传递指针或所需成员更为合理。

结构体，还是指向结构体的指针

例

用结构体表示三维向量，并实现向量的一些基本操作

- 两向量的加法、减法、内积
- 两向量之间的夹角
- 向量的长度

结构体，还是指向结构体的指针 i

```c
#include <stdio.h>
#include <math.h>
typedef struct {
  double val[3];
} Vec;
Vec add(Vec a, Vec b)
{
  Vec c;
  for (int i = 0; i < 3; i++)
    c.val[i] = a.val[i] + b.val[i];
  return c;
}
Vec sub(Vec a, Vec b)
{
  Vec c;
  for (int i = 0; i < 3; i++)
```

结构体，还是指向结构体的指针 ii

```
    c.val[i] = a.val[i] - b.val[i];
  return c;
}
double dot(Vec a, Vec b)
{
  return a.val[0]*b.val[0]
    +    a.val[1]*b.val[1]
    +    a.val[2]*b.val[2];
}
double norm(Vec a)
{
  return sqrt(dot(a, a));
}
void show(Vec a)
{
```

结构体，还是指向结构体的指针 iii

```
  printf("<% .2f, % .2f, % .2f>\n", a.val[0], a.val
  [1], a.val[2]);
}
```

结构体，还是指向结构体的指针

```c
#include "vector.h"
int main(void)
{
  Vec a = {1.0, 1.0, 1.0};
  Vec b = {1.0, 2.0, 3.0};
  Vec c;
  c = add(a, b); printf("a + b = "); show(c);
  c = sub(a, b); printf("a - b = "); show(c);
  printf("(a, b) = %.2f\n", dot(a, b));
  printf("||a|| = %.2f\n", norm(a));
  return 0;
}
```

结构体，还是指向结构体的指针

- 通常，程序员为了追求效率，会使用结构体指针作为函数参数；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。
- 而传递结构体是处理小型结构体最常用的办法。

向函数传递结构体信息

在结构体中使用字符数组还是字符指针

在结构体中使用字符数组还是字符指针

问题

能否将结构体中的字符数组用指向字符的指针来代替？即如下结构体声明

```
typedef struct {
  char first[20];
  char last [20];
} Name;
```

能否改写成

```
typedef struct {
  char * first;
  char * last;
} pName;
```

在结构体中使用字符数组还是字符指针

问题

能否将结构体中的字符数组用指向字符的指针来代替？即如下结构体声明

```
typedef struct {
  char first[20];
  char last [20];
} Name;
```

能否改写成

```
typedef struct {
  char * first;
  char * last;
} pName;
```

可以，但可能会遇到麻烦。

在结构体中使用字符数组还是字符指针

考虑如下代码

```
Name name1 = {"Stephan", "Prata"};
pName 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。

在结构体中使用字符数组还是字符指针

- 对于 Name 变量 name1，字符串存储在结构体内部；该结构体分配了 40 个字节来存放两个字符串。
- 对于 pName 变量 name2，字符串存储在编译器存储字符串常量的任何地方。该结构体存放的只是两个地址，总共占 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。）

  pName 结构体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。

在结构体中使用字符数组还是字符指针

考虑如下代码

```
Name boy;
pName girl;
puts("Enter the last name of the boy");
scanf("%s", boy.last);
puts("Enter the last name of the girl");
scanf("%s", girl.last);    //存在潜在危险
```

在结构体中使用字符数组还是字符指针

- 对于 `boy`，他的名存储在 `boy` 的第二个成员 `last` 中。
- 对于 `girl`，`scanf()` 把字符串放在由 `girl.last` 给出的地址中，而该地址未被初始化，可能为任意值，程序就可以把名放在任何地方。

在结构体中使用字符数组还是字符指针

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

向函数传递结构体信息

结构体、指针和 malloc()

结构体、指针和 malloc()

在结构体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。

结构体、指针和 malloc() i

```
// names.h
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {
  char * first;
  char * last;
  int letters;
} Name;

void getinfo(Name *);
void makeinfo(Name *);
void showinfo(const Name *);
void cleanup(Name *);
```

结构体、指针和 malloc() i

```
// info.c
#include "names.h"
void getinfo(Name * pname)
{
  char temp[81];
  puts("Enter your first name.");
  gets(temp);
  pname->first = (char *) malloc (strlen(temp)+1);
  strcpy(pname->first, temp);
  puts("Enter your last name.");
  gets(temp);
  pname->last = (char *) malloc (strlen(temp)+1);
  strcpy(pname->last, temp);
}

void makeinfo(Name * pname)
```

结构体、指针和 malloc() ii

```
{
  pname->letters = (int) strlen(pname->first) +
                   (int) strlen(pname->last);
}

void showinfo(const Name * pname)
{
  printf("%s %s, your name contains %d letters.\n",
         pname->first, pname->last, pname->letters);
}

void cleanup(Name * pname)
{
  free(pname->first);
  free(pname->last);
}
```

结构体、指针和 malloc()

```c
// main.c
#include "names.h"
int main(void)
{
  Name person;
  getinfo (&person);
  makeinfo(&person);
  showinfo(&person);
  cleanup (&person);
  return 0;
}
```

结构体、指针和 malloc()

注

- 两个字符串并没有存储在结构体中，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体中，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。

联合体（共同体）

联合体（共同体）

联合体（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。

使用联合体类型的数组，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。

联合体（共同体）

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

```
typedef union {
  int digit;
  double bigfl;
  char letter;
} Hold;
```

该联合体可以含有一个 int 值、一个 double 值或一个 char 值。

联合体（共同体）

以下是定义 3 个 hold 类型联合体变量的例子

```
Hold fit;
Hold save[10];
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 联合体的地址。

联合体（共同体）

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

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

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

联合体（共同体）

以下代码说明了如何使用联合体：

```
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。

联合体（共同体）

可以与指向联合体的指针一样使用 -> 运算符：

```
Hold * pu;
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
```

联合体（共同体）：联合体的应用

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。

```
typedef struct {
  char socsecurity[20];
  ...
} Owner;

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

联合体（共同体）：联合体的应用

```
typedef union {
  Owner owncar;
  LeaseCompany leasecar;
} Info;

typedef struct {
  char make[15];
  int status;  // 0 = 私有, 1 = 租赁
  Info info;
  ...
} CarInfo;
```

联合体（共同体）：联合体的应用

假定 honda 是一个 CarInfo 结构，则

- 若 honda.status 为 0 (私家车)，则程序可使用 honda.info.owncar.socsecurity;
- 若 honda.status 为 1 (租赁车)，则程序可使用 honda.info.leasecar.name.

枚举类型

枚举类型

可使用**枚举类型**（enumerated type）声明代表整型常量的符号名称。

使用关键字 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**结构体**相同。

枚举类型

声明方式如下：

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

- 第一个声明定义一个新类型 Spectrum。
- 第二个声明定义 Spectrum 变量 color，花括号中的标识符枚举了 Spectrum 变量的所有可能值。

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++)
   ...
```

注

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：

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

- 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：

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

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

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
typedef enum {
  red, orange, yellow, green, blue, violet
} Spectrum;
const char * colors[] = { "red", "orange", "yellow",
                "green", "blue", "violet" };
#define LEN 30
int main(void)
{
  char choice[LEN];
  Spectrum color;
  bool color_is_found = false;
  puts("Enter a color (empty line to quit):");
  while (gets(choice) != NULL && choice[0] != '\0') {
```

枚举类型：enum 的用法 ii

```
for (color = red; color <= violet; color++) {
  if (strcmp(choice, colors[color]) == 0) {
     color_is_found = true;
     break;
  }
}
if (color_is_found)
  switch (color) {
  case red:    puts("Roes are red."); break;
  case orange: puts("Poppies are orange."); break;
  case yellow: puts("Sunflowers are yellow.");
  break;
  case green:  puts("Grass is green."); break;
  case blue:   puts("Bluebells are blue."); break;
  case violet: puts("Violets are violet."); break;
  }
```

枚举类型：enum 的用法 iii

```
    else
      printf("I don't know about the color %s.\n",
      choice);
    color_is_found = false;
    puts("Next color, please (empty line to quit): ");
  }
  puts("Goodbye!");
  return 0;
}
```

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

Goodbye!
```

typedef 简介

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

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
    //具有 4 个元素的 int 数组的指针
```

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
```

指针函数与函数指针

指针函数与函数指针

指针函数

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

```c
#include <stdio.h>
#include <stdlib.h>
int * f(int a, int b);
int main(void)
{
  int * q = NULL;
  printf("Memory address of q = %p\n", q);
  q = f(1, 2);
  printf("Memory address of q = %p\n", q);
  printf("*q = %d \n", *q);
  return 0;
}
int * f(int a, int b) {
  int * p = (int *) malloc(sizeof(int));
```

指针函数 ii

```
  printf("Memory address of p = %p\n", p);
  *p = a + b;
  printf("*p = %d \n", *p);
  return p;
}
```

指针函数

```
Memory address of q = (nil)
Memory address of p = 0x5627e8f93670
*p = 3
Memory address of q = 0x5627e8f93670
*q = 3
```

指针函数与函数指针

函数指针

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

例

使用函数指针计算两个 `int` 值的较大值和较小值。

函数指针 i

```c
#include <stdio.h>
#include <stdlib.h>
int max(int a, int b);
int min(int a, int b);
int (*f) (int, int);
int main(void)
{
  f = max;
  printf("The max value is %d \n", (*f)(1, 2));
  f = min;
  printf("The min value is %d \n", (*f)(1, 2));
  return 0;
}
int max(int a, int b)
```

函数指针 ii

```
{
  return (a > b ? a : b);
}
int min(int a, int b)
{
  return (a < b ? a : b);
}
```

函数指针

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

函数指针 i

例

使用函数指针对两个 `double` 值做加减乘除。

函数指针 i

```
// fun_ptr1.h
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void Add(double a, double b)
{
  printf("%8.3f + %8.3f = %8.3f\n", a, b, a + b)
  ;
}
void Sub(double a, double b)
{
  printf("%8.3f - %8.3f = %8.3f\n", a, b, a - b)
  ;
}
```

函数指针 ii

```
void Mul(double a, double b)
{
  printf("%8.3f * %8.3f = %8.3f\n", a, b, a * b)
  ;
}
void Div(double a, double b)
{
  if (fabs(b) < 1.e-12) {
    printf("Denominator is nearly zero!\n");
    exit(EXIT_FAILURE);
  }
  printf("%8.3f / %8.3f = %8.3f\n", a, b, a / b)
  ;
}
```

函数指针 i

```
// fun_ptr1.c
#include "fun_ptr1.h"
int main()
{
  void (*mathop[])(double, double) = {Add, Sub,
  Mul, Div};
  int choice;
  double a = 15.0, b = 3.0;
  printf("Enter choice: 0 for add;       1 for
  substract\n");
  printf("                2 for multiply; 3 for
  division\n");
  scanf("%d", &choice);
  if (choice > 3) return 0;
```

函数指针 ii

```
    (*mathop[choice])(a, b);
    return 0;
}
```

函数指针 i

将函数指针作为参数传递给一个函数

对两个 `double` 值做加减乘除。

函数指针 i

```
// fun_ptr2.h
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double Add(double a, double b)
{
  return a + b;
}
double Sub(double a, double b)
{
  return a - b;
}
double Mul(double a, double b)
{
```

函数指针 ii

```
  return a * b;
}
double Div(double a, double b)
{
  if (abs(b) < 1.e-12) {
    printf("Denominator is nearly zero!\n");
    exit(EXIT_FAILURE);
  }
  return a / b;
}
double domath( double a, double b,
           double (*mathop)(double, double) )
{
  return (*mathop)(a, b);
}
```

函数指针 iii

函数指针 i

```
// fun_ptr2.c
#include "fun_ptr2.h"
int main(void)
{
  printf("Add gives: %8.3f\n", domath(10., 2.,
  Add));
  printf("Sub gives: %8.3f\n", domath(10., 2.,
  Sub));
  printf("Mul gives: %8.3f\n", domath(10., 2.,
  Mul));
  printf("Div gives: %8.3f\n", domath(10., 0.,
  Div));
  return 0;
}
```