C 字符串 C 共用体

C 结构体

C 数组允许定义可存储相同类型数据项的变量，**结构**是 C 编程中另一种用户自定义的可用的数据类型，它允许您存储不同类型的数据项。

结构用于表示一条记录，假设您想要跟踪图书馆中书本的动态，您可能需要跟踪每本书的下列属性：

- Title
- Author
- Subject
- Book ID

定义结构

为了定义结构，您必须使用 **struct** 语句。struct 语句定义了一个包含多个成员的新的数据类型，struct 语句的格式如下：

```
struct tag {
member-list
member-list
member-list
...
} variable-list ;
```

tag 是结构体标签。

member-list 是标准的变量定义，比如 int i; 或者 float f，或者其他有效的变量定义。

variable-list 结构变量，定义在结构的末尾，最后一个分号之前，您可以指定一个或多个结构变量。下面是声明 Book 结构的方式：

```
struct Books
{
char title[50];
char author[50];
char subject[100];
int book_id;
} book;
```

在一般情况下，**tag、member-list、variable-list** 这 3 部分至少要出现 2 个。以下为实例：

```
//此声明声明了拥有3个成员的结构体，分别为整型的a，字符型的b和双精度的c
//同时又声明了结构体变量s1
//这个结构体并没有标明其标签
struct
{
int a;
char b;
double c;
```

```
} s1;
//此声明声明了拥有3个成员的结构体，分别为整型的a，字符型的b和双精度的c
//结构体的标签被命名为SIMPLE,没有声明变量
struct SIMPLE
{
int a;
char b;
double c;
};
//用SIMPLE标签的结构体，另外声明了变量t1、t2、t3
struct SIMPLE t1, t2[20], *t3;
//也可以用typedef创建新类型
typedef struct
{
int a;
char b;
double c;
} Simple2;
//现在可以用Simple2作为类型声明新的结构体变量
Simple2 u1, u2[20], *u3;
```

在上面的声明中，第一个和第二声明被编译器当作两个完全不同的类型，即使他们的成员列表是一样的，如果令 t3=&s1，则是非法的。

结构体的成员可以包含其他结构体，也可以包含指向自己结构体类型的指针，而通常这种指针的应用是为了实现一些更高级的数据结构如链表和树等。

```
//此结构体的声明包含了其他的结构体
struct COMPLEX
{
char string[100];
struct SIMPLE a;
};
//此结构体的声明包含了指向自己类型的指针
struct NODE
{
char string[100];
struct NODE *next_node;
};
```

如果两个结构体互相包含，则需要对其中一个结构体进行不完整声明，如下所示：

```
struct B; //对结构体B进行不完整声明
//结构体A中包含指向结构体B的指针
struct A
{
struct B *partner;
//other members;
};
//结构体B中包含指向结构体A的指针，在A声明完后，B也随之进行声明
struct B
{
struct A *partner;
```

```
//other members;
};
```

结构体变量的初始化

和其它类型变量一样，对结构体变量可以在定义时指定初始值。

实例

```
#include <stdio.h>
struct Books
{
char title[50];
char author[50];
char subject[100];
int book_id;
} book = {"C 语言", "RUNOOB", "编程语言", 123456};
int main()
{
printf("title : %s\nauthor: %s\nsubject: %s\nbook_id: %d\n", book.title, book.author, book.subj
ect, book.book_id);
}
```

执行输出结果为：

```
title : C 语言
author: RUNOOB
subject: 编程语言
book_id: 123456
```

访问结构成员

为了访问结构的成员，我们使用**成员访问运算符（.）**。成员访问运算符是结构变量名称和我们要访问的结构成员之间的一个句号。您可以使用 **struct** 关键字来定义结构类型的变量。下面的实例演示了结构的用法：

实例

```
#include <stdio.h>
#include <string.h>
struct Books
{
char title[50];
char author[50];
char subject[100];
int book_id;
};
int main( )
{
struct Books Book1; /* 声明 Book1，类型为 Books */
struct Books Book2; /* 声明 Book2，类型为 Books */
/* Book1 详述 */
strcpy( Book1.title, "C Programming");
```

```
strcpy( Book1.author, "Nuha Ali");
strcpy( Book1.subject, "C Programming Tutorial");
Book1.book_id = 6495407;
/* Book2 详述 */
strcpy( Book2.title, "Telecom Billing");
strcpy( Book2.author, "Zara Ali");
strcpy( Book2.subject, "Telecom Billing Tutorial");
Book2.book_id = 6495700;
/* 输出 Book1 信息 */
printf( "Book 1 title : %s\n", Book1.title);
printf( "Book 1 author : %s\n", Book1.author);
printf( "Book 1 subject : %s\n", Book1.subject);
printf( "Book 1 book_id : %d\n", Book1.book_id);
/* 输出 Book2 信息 */
printf( "Book 2 title : %s\n", Book2.title);
printf( "Book 2 author : %s\n", Book2.author);
printf( "Book 2 subject : %s\n", Book2.subject);
printf( "Book 2 book_id : %d\n", Book2.book_id);
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

结构作为函数参数

您可以把结构作为函数参数，传参方式与其他类型的变量或指针类似。您可以使用上面实例中的方式来访问结构变量：

实例

```
#include <stdio.h>
#include <string.h>
struct Books
{
char title[50];
char author[50];
char subject[100];
int book_id;
};
/* 函数声明 */
void printBook( struct Books book );
int main( )
{
struct Books Book1; /* 声明 Book1, 类型为 Books */
```

```
struct Books Book2; /* 声明 Book2，类型为 Books */
/* Book1 详述 */
strcpy( Book1.title, "C Programming");
strcpy( Book1.author, "Nuha Ali");
strcpy( Book1.subject, "C Programming Tutorial");
Book1.book_id = 6495407;
/* Book2 详述 */
strcpy( Book2.title, "Telecom Billing");
strcpy( Book2.author, "Zara Ali");
strcpy( Book2.subject, "Telecom Billing Tutorial");
Book2.book_id = 6495700;
/* 输出 Book1 信息 */
printBook( Book1 );
/* 输出 Book2 信息 */
printBook( Book2 );
return 0;
}
void printBook( struct Books book )
{
printf( "Book title : %s\n", book.title);
printf( "Book author : %s\n", book.author);
printf( "Book subject : %s\n", book.subject);
printf( "Book book_id : %d\n", book.book_id);
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

指向结构的指针

您可以定义指向结构的指针，方式与定义指向其他类型变量的指针相似，如下所示：

```
struct Books *struct_pointer;
```

现在，您可以在上述定义的指针变量中存储结构变量的地址。为了查找结构变量的地址，请把 & 运算符放在结构名称的前面，如下所示：

```
struct_pointer = &Book1;
```

为了使用指向该结构的指针访问结构的成员，您必须使用 -> 运算符，如下所示：

```
struct_pointer->title;
```

让我们使用结构指针来重写上面的实例，这将有助于您理解结构指针的概念：

实例

```
#include <stdio.h>
#include <string.h>
struct Books
{
char title[50];
char author[50];
char subject[100];
int book_id;
};
/* 函数声明 */
void printBook( struct Books *book );
int main( )
{
struct Books Book1; /* 声明 Book1, 类型为 Books */
struct Books Book2; /* 声明 Book2, 类型为 Books */
/* Book1 详述 */
strcpy( Book1.title, "C Programming");
strcpy( Book1.author, "Nuha Ali");
strcpy( Book1.subject, "C Programming Tutorial");
Book1.book_id = 6495407;
/* Book2 详述 */
strcpy( Book2.title, "Telecom Billing");
strcpy( Book2.author, "Zara Ali");
strcpy( Book2.subject, "Telecom Billing Tutorial");
Book2.book_id = 6495700;
/* 通过传 Book1 的地址来输出 Book1 信息 */
printBook( &Book1 );
/* 通过传 Book2 的地址来输出 Book2 信息 */
printBook( &Book2 );
return 0;
}
void printBook( struct Books *book )
{
printf( "Book title : %s\n", book->title);
printf( "Book author : %s\n", book->author);
printf( "Book subject : %s\n", book->subject);
printf( "Book book_id : %d\n", book->book_id);
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
```

```
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

位域

有些信息在存储时，并不需要占用一个完整的字节，而只需占几个或一个二进制位。例如在存放一个开关量时，只有 0 和 1 两种状态，用 1 位二进位即可。为了节省存储空间，并使处理简便，C 语言又提供了一种数据结构，称为"位域"或"位段"。

所谓"位域"是把一个字节中的二进位划分为几个不同的区域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作。这样就可以把几个不同的对象用一个字节的二进制位域来表示。

典型的实例：

- 用 1 位二进位存放一个开关量时，只有 0 和 1 两种状态。
- 读取外部文件格式——可以读取非标准的文件格式。例如：9 位的整数。

位域的定义和位域变量的说明

位域定义与结构定义相仿，其形式为：

```
struct 位域结构名
{

 位域列表

};
```

其中位域列表的形式为：

```
类型说明符 位域名: 位域长度
```

例如：

```
struct bs{
int a:8;
int b:2;
int c:6;
}data;
```

说明 data 为 bs 变量，共占两个字节。其中位域 a 占 8 位，位域 b 占 2 位，位域 c 占 6 位。

让我们再来看一个实例：

```
struct packed_struct {
unsigned int f1:1;
unsigned int f2:1;
unsigned int f3:1;
unsigned int f4:1;
unsigned int type:4;
```

```
unsigned int my_int:9;
} pack;
```

在这里，packed_struct 包含了 6 个成员：四个 1 位的标识符 f1..f4、一个 4 位的 type 和一个 9 位的 my_int。

对于位域的定义尚有以下几点说明：

- 一个位域存储在同一个字节中，如一个字节所剩空间不够存放另一位域时，则会从下一单元起存放该位域。也可以有意使某位域从下一单元开始。例如：

```
struct bs{
unsigned a:4;
unsigned :4; /* 空域 */
unsigned b:4; /* 从下一单元开始存放 */
unsigned c:4
}
```

  在这个位域定义中，a 占第一字节的 4 位，后 4 位填 0 表示不使用，b 从第二字节开始，占用 4 位，c 占用 4 位。

- 由于位域不允许跨两个字节，因此位域的长度不能大于一个字节的长度，也就是说不能超过8位二进位。如果最大长度大于计算机的整数字长，一些编译器可能会允许域的内存重叠，另外一些编译器可能会把大于一个域的部分存储在下一个字中。

- 位域可以是无名位域，这时它只用来作填充或调整位置。无名的位域是不能使用的。例如：

```
struct k{
int a:1;
int :2; /* 该 2 位不能使用 */
int b:3;
int c:2;
};
```

从以上分析可以看出，位域在本质上就是一种结构类型，不过其成员是按二进位分配的。

位域的使用

位域的使用和结构成员的使用相同，其一般形式为：

```
位域变量名.位域名
位域变量名->位域名
```

位域允许用各种格式输出。

请看下面的实例：

实例

```
main(){
struct bs{
unsigned a:1;
unsigned b:3;
unsigned c:4;
```

```
} bit,*pbit;
bit.a=1; /* 给位域赋值（应注意赋值不能超过该位域的允许范围） */
bit.b=7; /* 给位域赋值（应注意赋值不能超过该位域的允许范围） */
bit.c=15; /* 给位域赋值（应注意赋值不能超过该位域的允许范围） */
printf("%d,%d,%d\n",bit.a,bit.b,bit.c); /* 以整型量格式输出三个域的内容 */
pbit=&bit; /* 把位域变量 bit 的地址送给指针变量 pbit */
pbit->a=0; /* 用指针方式给位域 a 重新赋值，赋为 0 */
pbit->b&=3; /* 使用了复合的位运算符 "&="，相当于：pbit->b=pbit->b&3，位域 b 中原有值为 7，与 3 作按位与运算的结果为 3（111&011=011，十进制值为 3） */
pbit->c|=1; /* 使用了复合位运算符"|="，相当于：pbit->c=pbit->c|1，其结果为 15 */
printf("%d,%d,%d\n",pbit->a,pbit->b,pbit->c); /* 用指针方式输出了这三个域的值 */
}
```

上例程序中定义了位域结构 bs，三个位域为 a、b、c。说明了 bs 类型的变量 bit 和指向 bs 类型的指针变量 pbit。这表示位域也是可以使用指针的。

← C 字符串 C 共用体 →