C 结构体 C 位域

C 共用体

共用体是一种特殊的数据类型，允许您在相同的内存位置存储不同的数据类型。您可以定义一个带有多成员的共用体，但是任何时候只能有一个成员带有值。共用体提供了一种使用相同的内存位置的有效方式。

定义共用体

为了定义共用体，您必须使用 **union** 语句，方式与定义结构类似。union 语句定义了一个新的数据类型，带有多个成员。union 语句的格式如下：

```
union [union tag]
{
member definition;
member definition;
...
member definition;
} [one or more union variables];
```

union tag 是可选的，每个 member definition 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。在共用体定义的末尾，最后一个分号之前，您可以指定一个或多个共用体变量，这是可选的。下面定义一个名为 Data 的共用体类型，有三个成员 i、f 和 str：

```
union Data
{
int i;
float f;
char str[20];
} data;
```

现在，**Data** 类型的变量可以存储一个整数、一个浮点数，或者一个字符串。这意味着一个变量（相同的内存位置）可以存储多个多种类型的数据。您可以根据需要在一个共用体内使用任何内置的或者用户自定义的数据类型。

共用体占用的内存应足够存储共用体中最大的成员。例如，在上面的实例中，Data 将占用 20 个字节的内存空间，因为在各个成员中，字符串所占用的空间是最大的。下面的实例将显示上面的共用体占用的总内存大小：

实例

```
#include <stdio.h>
#include <string.h>
union Data
{
int i;
float f;
char str[20];
};
int main( )
{
union Data data;
printf( "Memory size occupied by data : %d\n", sizeof(data));
```

```
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Memory size occupied by data : 20
```

访问共用体成员

为了访问共用体的成员，我们使用**成员访问运算符（.）**。成员访问运算符是共用体变量名称和我们要访问的共用体成员之间的一个句号。您可以使用 **union** 关键字来定义共用体类型的变量。下面的实例演示了共用体的用法：

实例

```
#include <stdio.h>
#include <string.h>
union Data
{
int i;
float f;
char str[20];
};
int main( )
{
union Data data;
data.i = 10;
data.f = 220.5;
strcpy( data.str, "C Programming");
printf( "data.i : %d\n", data.i);
printf( "data.f : %f\n", data.f);
printf( "data.str : %s\n", data.str);
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming
```

在这里，我们可以看到共用体的 **i** 和 **f** 成员的值有损坏，因为最后赋给变量的值占用了内存位置，这也是 **str** 成员能够完好输出的原因。现在让我们再来看一个相同的实例，这次我们在同一时间只使用一个变量，这也演示了使用共用体的主要目的：

实例

```
#include <stdio.h>
#include <string.h>
union Data
{
int i;
```

```
float f;
char str[20];
};
int main( )
{
union Data data;
data.i = 10;
printf( "data.i : %d\n", data.i);
data.f = 220.5;
printf( "data.f : %f\n", data.f);
strcpy( data.str, "C Programming");
printf( "data.str : %s\n", data.str);
return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

在这里，所有的成员都能完好输出，因为同一时间只用到一个成员。

← C 结构体 C 位域 →