C++ 基本的输入输出

C++ vector 容器

C++ 结构体(struct)

*C/C++ 数组允许定义可存储相同类型数据项的变量，但是**结构**是 C++ 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： 标题

Author： 作者

Subject： 类目

Book ID： 书的 ID

定义结构

在 C++ 中，struct 语句用于定义结构体（structure）。

结构体是一种用户自定义的数据类型，用于将不同类型的数据组合在一起。与类（class）类似，结构体允许你定义成员变量和成员函数。

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 type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

type_name 是结构体类型的名称，**member_type1 member_name1** 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。在结构定义的末尾，最后一个分号之前，您可以指定一个或多个结构变量，这是可选的。下面是声明一个结构体类型 **Books**，变量为 **book**：

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

结构体优点：

- **简单数据封装**：适合封装多种类型的简单数据，通常用于数据的存储。
- **轻量级**：相比 `class`，结构体语法更简洁，适合小型数据对象。
- **面向对象支持**：支持构造函数、成员函数和访问权限控制，可以实现面向对象的设计。

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。

下面的实例演示了结构的用法：

实例

C++ 资源库


```
#include <iostream>
#include <cstring>

using namespace std;

// 声明一个结构体类型 Books
struct Books
{
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( )
{
   Books Book1;        // 定义结构体类型 Books 的变量 Book1
   Books Book2;        // 定义结构体类型 Books 的变量 Book2

   // Book1 详述
   strcpy( Book1.title, "C++ 教程");
   strcpy( Book1.author, "Runoob");
   strcpy( Book1.subject, "编程语言");
   Book1.book_id = 12345;

   // Book2 详述
   strcpy( Book2.title, "CSS 教程");
   strcpy( Book2.author, "Runoob");
   strcpy( Book2.subject, "前端技术");
   Book2.book_id = 12346;

   // 输出 Book1 信息
   cout << "第一本书标题 : " << Book1.title <<endl;
   cout << "第一本书作者 : " << Book1.author <<endl;
   cout << "第一本书类目 : " << Book1.subject <<endl;
   cout << "第一本书 ID : " << Book1.book_id <<endl;

   // 输出 Book2 信息
```

```
   cout << "第二本书标题 : " << Book2.title <<endl;
   cout << "第二本书作者 : " << Book2.author <<endl;
   cout << "第二本书类目 : " << Book2.subject <<endl;
   cout << "第二本书 ID : " << Book2.book_id <<endl;

   return 0;
}
```

实例中定义了结构体类型 Books 及其两个变量 Book1 和 Book2。当上面的代码被编译和执行时，它会产生下列结果：

```
第一本书标题 : C++ 教程
第一本书作者 : Runoob
第一本书类目 : 编程语言
第一本书 ID : 12345
第二本书标题 : CSS 教程
第二本书作者 : Runoob
第二本书类目 : 前端技术
第二本书 ID : 12346
```

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 <iostream>
#include <cstring>

using namespace std;
void printBook( struct Books book );

// 声明一个结构体类型 Books
struct Books
```

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

int main( )
{
   Books Book1;        // 定义结构体类型 Books 的变量 Book1
   Books Book2;        // 定义结构体类型 Books 的变量 Book2

    // Book1 详述
   strcpy( Book1.title, "C++ 教程");
   strcpy( Book1.author, "Runoob");
   strcpy( Book1.subject, "编程语言");
   Book1.book_id = 12345;

   // Book2 详述
   strcpy( Book2.title, "CSS 教程");
   strcpy( Book2.author, "Runoob");
   strcpy( Book2.subject, "前端技术");
   Book2.book_id = 12346;

   // 输出 Book1 信息
   printBook( Book1 );

   // 输出 Book2 信息
   printBook( Book2 );

   return 0;
}
void printBook( struct Books book )
{
   cout << "书标题 : " << book.title <<endl;
   cout << "书作者 : " << book.author <<endl;
   cout << "书类目 : " << book.subject <<endl;
```

```
    cout << "书 ID : " << book.book_id <<endl;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
书标题 : C++ 教程
书作者 : Runoob
书类目 : 编程语言
书 ID : 12345
书标题 : CSS 教程
书作者 : Runoob
书类目 : 前端技术
书 ID : 12346
```

结构体的各个部分详细介绍

struct 关键字：用于定义结构体，它告诉编译器后面要定义的是一个自定义类型。

成员变量：成员变量是结构体中定义的数据项，它们可以是任何基本类型或其他自定义类型。在 struct 中，这些成员默认是 public，可以直接访问。

成员函数：结构体中也可以包含成员函数，这使得结构体在功能上类似于类。成员函数可以操作结构体的成员变量，提供对数据的封装和操作。

访问权限：与 class 类似，你可以在 struct 中使用 public、private 和 protected 来定义成员的访问权限。在 struct 中，默认所有成员都是 public，而 class 中默认是 private。

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 <iostream>
#include <string>

using namespace std;

// 声明一个结构体类型 Books
struct Books
{
    string title;
    string author;
    string subject;
    int book_id;

    // 构造函数
    Books(string t, string a, string s, int id)
        : title(t), author(a), subject(s), book_id(id) {}
};

// 打印书籍信息的函数
void printBookInfo(const Books& book) {
    cout << "书籍标题: " << book.title << endl;
    cout << "书籍作者: " << book.author << endl;
```

```
    cout << "书籍类目: " << book.subject << endl;
    cout << "书籍 ID: " << book.book_id << endl;
}

int main()
{
    // 创建两本书的对象
    Books Book1("C++ 教程", "Runoob", "编程语言", 12345);
    Books Book2("CSS 教程", "Runoob", "前端技术", 12346);

    // 输出书籍信息
    printBookInfo(Book1);
    printBookInfo(Book2);

    return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
书标题  : C++ 教程
书作者 : Runoob
书类目 : 编程语言
书 ID : 12345
书标题  : CSS 教程
书作者 : Runoob
书类目 : 前端技术
书 ID : 12346
```

typedef 关键字

下面是一种更简单的定义结构的方式，您可以为创建的类型取一个"别名"。例如：

```
typedef struct Books
{
   char  title[50];
```

```
    char  author[50];
    char  subject[100];
    int   book_id;
} Books;
```

现在，您可以直接使用 *Books* 来定义 *Books* 类型的变量，而不需要使用 struct 关键字。下面是实例：

```
Books Book1, Book2;
```

您可以使用 **typedef** 关键字来定义非结构类型，如下所示：

```
typedef long int *pint32;

pint32 x, y, z;
```

x, y 和 z 都是指向长整型 long int 的指针。

结构体与类的区别

在 C++ 中，struct 和 class 本质上非常相似，唯一的区别在于默认的访问权限：

- `struct` 默认的成员和继承是 `public`。
- `class` 默认的成员和继承是 `private`。

你可以将 `struct` 当作一种简化形式的 `class`，适合用于没有太多复杂功能的简单数据封装。

结构体与函数的结合

你可以通过构造函数初始化结构体，还可以通过引用传递结构体来避免不必要的拷贝。

实例

```
struct Books {
    string title;
```

```
    string author;
    string subject;
    int book_id;

    // 构造函数
    Books(string t, string a, string s, int id)
        : title(t), author(a), subject(s), book_id(id) {}

    void printInfo() const {
        cout << "书籍标题: " << title << endl;
        cout << "书籍作者: " << author << endl;
        cout << "书籍类目: " << subject << endl;
        cout << "书籍 ID: " << book_id << endl;
    }
};

void printBookByRef(const Books& book) {
    book.printInfo();
}
```

← C++ 基本的输入输出

C++ vector 容器 →

- Ajax 实例
- jQuery 实例
- XML 实例
- Java 实例

- JS 混淆/加密
- PNG/JPEG 图片压缩
- HTML 拾色器
- JSON 格式化工具
- 随机数生成器

- PyTorch 基础
- PyTorch 安装
- PyTorch 简介
- PyTorch 教程
- Tailwind CSS 布...

- 文章归档

关注微信