C++ Primer Plus 自学笔记1：C++特性及程序主体简介

原创 Zerg_Wang 于 2020-05-12 01:09:58 发布 227 收藏 编辑 版权

分类专栏： C 文章标签： c++ c语言 数据结构 算法 入门

C语言特性

结构化编程（Structured Programming）

早期语言（如 Fortran 和 Basic ）在代码组织上可能会存在问题（例如分支语句常常会让代码逻辑和可读性变得特别差）。对此，C语言使用了结构化编程这一方法，将程序中的分支限制为一组易读的代码结构（例如 for、while、do while 循环以及 if else 语句）。

自顶向下（ top-down ）

将大型程序分解为多个小型、易管理、可重复调用的小任务，这个分解过程可以持续下去，把小任务分解为更小的任务。在C语言中，通过函数来实现“自顶向下”。

C++语言特性

面向过程编程

即 Procedure Oriented Programming（POP），以过程为中心的编程思想。以“什么正在发生”为主要目标进行编程，是 C语言的重要特性（前文所述的结构化编程、自顶向下思想亦根植于此特性）。由于 C++ 继承自 C，因此 C++ 也拥有此特性。

面向对象编程

即 Object Oriented Programming（OOP）。在 C++ 中通过类与对象这两种数据结构及封装、多态、继承等方法实现。类规定了用何数据描述一个物像（或者概念）并规定了如何用这些数据去描述，对象则是根据类的要求构造的而成的，代表某个物像（或概念）的特殊数据结构。因此与面向过程不同，面向对象的重点不在任务上，而在于表示概念。

面向对象编程常常从低级组织（类）开始构建，进而构建高级组织（程序），该过程称为自下向上编程（ bottom-up ）

泛型编程

即Generic Programming。泛型也是一种特殊的抽象数据结构，相较于面向对象编程关注于数据，泛型编程关注算法本身，因此其不拘泥于数据类型。例如，写一个两数相加返回结果的函数，若输入的数据类型可能为int，也可能为float，则比较麻烦，通过泛型（更具体地说，使用模板）则可较为方便地解决：

```
#include <iostream>
using namespace std;

template <typename T>
T add(T a,T b){
    return a+b;
}

int main(){
    int n1, n2, n3;
    float f1, f2, f3;
    cin>>n1>>n2;
    cin>>f1>>f2;
    n3 = add(n1, n2);
    f3 = add(f1, f2);
    return 0;
}
```

可移植性及C++标准

可移植性是指同样的代码，只需要使用针对不同平台设计的编译器进行编译，而不必更改代码本身，就可以跨平台运行（这里所说的平台，是CPU + 操作系统，也就是说，即使操作系统相同，CPU不同，也可能算作不同平台）。然而程序的可移植性的实现有两大阻碍。一是硬件，与硬件相关的程序不可移植。举个例子，汇编语言的几乎就是不可移植的，因为它过于底层，直接操作硬件，然而不同的CPU有不同的指令集（不同的指令集意味着不同的汇编语法），因此同样的汇编语言，可能换个CPU就不行了……二是标准。不同操作系统上，即使是同一种语言，由于标准不同语法上可能会有些许差异，这也是影响移植性的一大原因。

要解决第二种阻碍，可通过制定针对所有平台的统一标准来实现。对此，ISO标准委员会分别于1998、2003、2011年推出了C++标准（分别为C++98、C++03、C++11标准，其中C++03是C++98的补丁版，两者语言特性一致）。之后也ISO也颁布过C++14、C++17等标准……

同理，C语言也有国际标准，如C89、C90、C99等（其中C89和C90一样）

虽然C++一般可以视为功能更强大的C，但两者并不能完全兼容。C++标准可“近似地”视为C标准的超集，但也并不是所有的C语言的代码都可正确运行于C++中。
C++ Primer Plus 书中所述基于C++98标准，并介绍部分C++11特性。

再识 A+B Problem

下面我们将逐行解释以下程序

```
#include <iostream>
using namespace std;
int main(){
    int a, b, c;
    cin>>a>>b;
    c = a + b;
    cout<<c<<endl;
    return 0;
}
```

第一行：头文件

头文件的作用、编译器对头文件的处理方式（预编译）这里就不展开了，这里讲讲头文件的几种形式：

- C++的头文件，用C++的写法（不带后缀名）。例如，iostream是C++的头文件，调用时就写为：#include <iostream>
- C++的头文件，但用C的写法（即使用.h）。例如：#include <iostream.h>
- C的头文件，用C的写法，例如：#include <math.h>
- C的头文件，但后来被转换为C++的头文件，则不需要后缀名，但要在前面加上“c”，例如：#include <cmath>

再讲讲语法规则：

如果include的是官方的标准头文件，用尖括号“包围”头文件名，如：

```
#include <iostream>
```

预编译时编译器会在系统中存放标准头文件的目录中寻找。

如果是include的是用户编写的，非官方的头文件，则用双引号：

```
#include "abc.h"
```

这样编译器在预编译的时候会优先在用户目录下搜索对应头文件，没有的话再到系统中存放标准头文件的目录中寻找。

第二行：命名空间（namespace）

假设程序引用的两个头文件a和b中，都有函数add的声明，那在main中调用add就会产生冲突，因此可通过定义两个命名空间A和B进行对应，那么在函数调用时，A::add()调用的就是头文件a中声明的函数，B::add()调用的就是头文件b中声明的函数。

目前，C++标准的函数、类、变量等，都置于命名空间std之下，假设#include <iostream>后使用cin、cout，则需要写出完整名字：std::cin和std::cout。当然，也可在main之前进行命名空间的声明：

```
using std::cout;
```

之后在使用cout时，可不必再写std::

当然，也可以直接声明使用std命名空间中的所有名称，从而在后续调用中免去所有std::

```
using namespace std;
```

该行命令可以写在函数内，则该函数从这条命令后的调用可省略std::，也可以写在函数外，则全局有效。

第三行：main函数

无论是单个cpp，还是大量文件的巨型工程，一般而言代码中有且只有一个main函数（没有main函数的其他cpp文件则被视为库文件，一般用于实现函数、类）。通常，我们视main函数为操作系统与程序的桥梁。系统通过main，来开始执行我们的源代码。

main函数无需参数传入，因此也可以写为：int main(void)

main函数一般写为int类型，返回值为0，然而实际上，根据C++标准，main函数可以省略掉return 0;当然也只有main函数有这个特殊待遇……

经典的C语言写法往往还能省略掉main前面的int：

```
main()
{
    int a, b, c;
    c = a + b;
    cout<<c;
    return 0;    //这句也能再省略
}
```

此外还有写成void main()的，大一的垃圾C++课用的VC 6.0支持的就是这个……不过这种写法比较新的编译器都不支持了 。

第四行：定义

没什么好说的，略了……

第五行：cin

按书中所述，cin和cout并不是函数，而是一种对象，cin为istream类的对象，cout为ostream类的对象，这两个类均在iostream中定义。符号“<<”和“>>”是用于指示信息的流动方向，如cout<<c，意思就是变量c中的内容（信息）被插入到输出流中。这里还涉及到运算符重载的问题。符号“<<”和“>>”实际上是位运算中左移和右移的符号，但在输入输出流中它们被重载了。

第六行：赋值

也没啥好说的，记一个不太熟的知识点：C++支持连续赋值。

```
int a = 1;
int d = c = b = a;
```

赋值依次进行，a的值赋给b，此时b==1，再把b的1赋值给c，依次类推。

第七行：cout

具体可看“第五行：cin”的相关内容，这里记个知识点：控制符“endl”实际上通过iostream声明，其全名为std::endl。

另外，endl是控制符，换行符是“\n”。

第八行：返回值

前文提到了，略了……

第九行：空白

最后讲讲格式问题。

代码中不可分割的元素叫做标记（token），例如int、main、return这些关键字、变量名等，标记之间一般需要用空白来隔开。所谓空白，是指空格、制表符（Tab）或者回车。

空白用于隔开标记，但实际上，并未规定用何种标记来隔开标记，因此程序可以写成这个样子：

```
#   include <iostream>
using
namespace
std
;
int             main
(       )
{return 0;}
```

也真是有够恶心的……