第1章 概 述

1.1 程序与语言

1.1.1 程序

程序是用能够被计算机理解的一种语言编写的语句的集合。它以某种语言为工具编制出有目的的、预想好的动作序列，表达人的思想。

对于计算机来说，一组机器指令就是程序。当我们说机器代码或者机器指令时，都是指程序，它是按计算机硬件设计规范的要求编制出来的动作序列。对于使用计算机的人来说，程序员用某高级语言编写的语句序列也是程序。程序通常以文件的形式保存起来。所以，源文件、源程序和源代码都是程序。

一个好的程序应该有以下特点：

(1) 正确可靠。不正确的程序不仅不能解决问题，反而会带来不必要的麻烦和损失。

(2) 清晰易读。由于受计算机速度和存储容量的限制，早期的程序往往将程序的效率放在第一位，随着科技发展，程序的可读性和可理解性成为设计的重点考虑内容。

(3) 易维护。当业务需求发生变化时，不需要太多的开销就可以扩展和增强程序的功能.

(4) 可移植性好。编写的程序在各种计算机和操作系统上都能运行，并且运行结果一样。

1.1.2 程序设计语言的发展

程序设计语言的发展经历了机器语言、汇编语言和高级语言等阶段，总的趋势是描述手段越来越高级，越来越接近自然语言或数学语言，越来越贴近客观世界本身。

最早，程序员使用最原始的计算机指令，即机器语言程序。只有机器语言才能为机器所识别和运行。这些指令由一串二进制的数表示。不久，发明了汇编语言，它可以将机器指令映射为一些能被人读懂的助记符，如 ADD,SUB。程序员运行汇编程序，将用助记符写成的源程序转换成机器指令，然后再运行机器指令程序，得到所要的结果。那时，编写程序的都是计算机专业人员，编写程序的语言都是低级的或是较低级的。这些语言的优点是：写出的程序效率较高。缺点是 程序难以设计、理解和维护，难以保证程序的正确性，此外，可移植性不好。

面对上述问题，Fortran,BASIC,Pascal;C 等几十种甚至几百种高级语言应运而生，中间

经历了严酷的优胜劣汰过程，最后剩下的是一些比较优秀的高级语言。高级语言的优点在于 程序容易设计、理解和维护，容易保证程序的正确性，另外，用高级语言编写的程序与采用的具体计算机的指令系统无关，这就容易将程序移植到其他不同型号的计算机中执行。

但是从本质上说，目前的高级语言大都只是在抽象级上比低级语言略高级一些而已，它们大都还是基于冯·诺依曼计算机的计算模型，而采用这些语言就必须按照计算机解决问题的方式来描述解题过程，所以程序设计仍然很困难。因此，人们还在努力设计抽象级别更高的语言或让计算机能够理解自然语言，以便程序员能用更自然的方式来设计程序。

1.2 面向对象程序设计的方法

1.2.1 面向对象方法的由来

“对象”一词在现实生活中经常会遇到，它表示现实世界中的某个具体的事物。

社会的不断进步和计算机科学的不断发展是相互促进的，一方面计算机科学的发展推动了社会的发展，计算机的广泛应用给整个社会生产力带来了勃勃生机；另一方面社会的发展，又给计算机科学提出了许多新的要求，计算机科学只有不断地进行自身提高和自身完善，才能适应不断进步的社会生产力的需要。随着计算机的普及应用，人们越来越希望能更直接地与计算机进行交互，而不需要经过专门学习和长时间训练后才能使用它。这一强烈愿望使软件设计人员的负担越来越重，也为计算机领域自身的发展提出了新的要求。利用传统的程序设计思想无法满足这一要求，人们就开始寻求一种更能反映人类解决问题的自然方法，“面向对象”技术就是在这样的情况下产生的。

“面向对象”技术追求的是软件系统对现实世界的直接模拟，尽量实现将现实世界中的事物直接映射到软件系统的解空间。它希望用户用最小的气力，最大程度地利用软件系统来解决问题。

现实世界中的事物可分为两大部分，即物质和意识，物质表达的是具体的事物；意识描述的是某一个抽象的概念。例如‘汽车’和‘那辆红色的汽车’，‘那辆红色的汽车’是物质，它是具体的客观存在，‘汽车’是意识，它是一个抽象的概念，是对客观存在的事物的一种概括。这些现实世界中的事物可直接映射到面向对象系统的解空间，现实世界中的物质可对应于面向对象系统中的‘对象’，现实世界中的意识可对应于面向对象系统中的抽象概念——类。汽车在面向对象系统中可用汽车类来表达，那辆红色的汽车在面向对象系统中是一个具体的对象，是汽车类的一个实例。如图 1.1 所示。

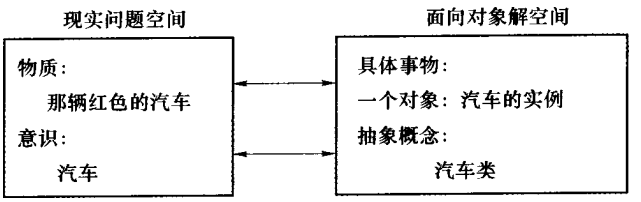

图 1.1 现实世界与面向对象系统之间的对应关系

1.2.2 面向对象的基本概念

面向对象程序设计(Object Oriented Programming)是软件系统设计与实现的新方法,这种新方法是通过增加软件的要扩充性和可重用性,来改善并提高程序员的生产能力,并控制维护软件的复杂性和软件维护的开销。下面先介绍面向对象程序设计的相关概念。

面向对象程序设计是由若干对象构造程序,每个对象由一些数据以及对这些数据所能实施的操作构成;对数据的操作通过向包含数据的对象发送消息(调用对象的操作)来实现;对象的特征(数据与操作)由相应的类来描述;一个类所描述的对象特征可以从其他的类继承。面向对象程序设计的定义包含了下面几个基本概念:

对象:是包含数据和处理这些数据的操作的程序单元,是构成面向对象程序的基本计算单位,由接口、数据及其操作构成。

通信:是指对象间的消息传递,是引起面向对象程序进行计算的惟一方式。

类:描述了一组具有相同或相近特征的对象的结构和行为。

继承:是指对象的一部分特征描述可以从其他的类获得,实现对数据和操作的共享。

1.2.3 面向对象程序设计与结构化程序设计

结构化程序设计(Structure Programming)是由 E. Dijkstra 等人于 1972 年提出的,它建立在 Bohm、Jacopini 证明的结构定理的基础上。结构定理指出:任何程序逻辑都可以用顺序、选择和循环三种基本结构来表示。

按照结构化程序设计的要求,程序在设计中应当采用"自顶向下,逐步求精"和"模块化"原则:"自顶向下,逐步求精"就是先需要对问题本身做出确切描述,并对问题解法做出全局性决策,把问题分解成相对独立的子问题,对每个子问题用抽象数据及其上的抽象操作来描述;然后,再以同样的方式对抽象数据和抽象操作进一步精确化,直到获得计算机能理解的程序为止。在这个过程中,要求程序设计必须先考虑全局,逐步将问题具体化:"模块化"是将一个大的系统按照子结构之间的疏密程度分解成较小的部分,每部分称为模块。分解的原则是模块间应相互独立,联系较少。

结构化程序设计对程序功能的描述比较清晰,所描述的计算过程容易理解,对规模较小的软件,它是非常适用的。但是由于它的数据与操作分离,对于不同的数据格式做相同的处理或是对相同的数据做不同的处理都需要编写不同的程序,随着问题复杂度的提高,数据量和数据类型的空前激增,会导致许多程序的规模和复杂性接近或达到用结构化程序设计方法无法管理的程度。

面向对象的程序设计是在吸收结构化程序设计的一切优点的基础上发展起来的一种新的程序设计方法,其本质是把数据和处理数据的过程抽象为一个具有特定身份和某些属性的自包含实体——对象。它将对象及对象的操作抽象成一种新的数据类型——类,并且考虑不同对象之间的联系和对象类的重用性。

面向对象程序设计是对问题领域活动的直接模拟,其中的对象往往对应着问题空间中的有形或无形的实体,它使得解题空间与问题空间有自然的对应关系,从而有利于对大型复杂问题给出解决方案,使得程序容易设计、理解和维护。面向对象程序设计的不足之处在

于：对程序的整体功能描述不明显；程序会包含较多的冗余信息，这对小型应用系统有时不合适 程序效率有时不高。

1.2.4 面向对象方法的优点

面向对象系统最突出的特点就是：封装、继承和多态性。

1．封装

在面向对象的程序设计中，封装是一种数据隐藏技术，它通过把一组数据和与数据有关的操作集合放在一起形成对象来实现。对象通过操作接口与外部发生联系，而内部的具体细节则被隐藏起来，对外不可见。封装机制是结构化程序设计方法难以支持的。

在结构化程序设计过程中，程序的各功能模块和数据之间的关系是由程序员在自己的头脑（或文档）中保持，在程序中这种约束关系没有显式地体现出来。随着将来系统功能的修改或扩充，程序结构可能会变得越来越差，难以维护。

而在面向对象程序设计中，数据及其相关的操作被封装在类或对象中，程序语言直接支持对这种相关性的描述和保护。这样，不仅结构清晰，而且系统各组成部分(对象)之间的独立性好，接口关系简单，交互途径单一——只能通过消息机制这一途径进行通信，这些特点都有利于系统功能的扩充和修改。类的封装机制将数据和代码捆绑在一起，避免了外界的干扰和不确定性。类可以用来创建对象。简单地说，一个对象就是一个封装了数据和操作这些数据的代码的逻辑实体。

2．继承

面向对象的程序设计中，同样有着继承的机制，通过继承，程序可以在扩展现有类的基础上声明新类——从原有类的基础上派生出来。原有类称为基类，又叫父类，新定义的类称为派生类。在继承基类属性和行为的同时，派生类可以添加自己的数据成员和函数成员。继承把派生类和基类联系起来，派生类对象同时也被认为是基类的对象。利用继承可以方便地重用已经定义的经过测试和调试的高质量的代码，提高软件开发的效率和软件质量。

3．多态性

多态性是指相同的消息为不同的对象接收到时，可能导致不同的动作。它使得属于同一类的不同对象可以按各自的需要对同一消息做出适当的响应，即为统一管理具有继承关系的不同类的对象提供了方便。使用多态性进行程序设计时，可以为具有继承关系的多个类定义统一的接口，而不同的类对于接口的实现则可各不相同，当通过统一的接口操纵这些对象时，程序可以根据被操纵对象的类型来确定具体该执行什么操作。而在结构化程序设计方法中，函数调用语句就确定了要执行的语句序列，要对不同的对象做不同的操作，就要对这些对象下不同的命令。

由于面向对象程序设计具有上述特点，面向对象方法就可以为软件产品的扩展及质量保证中的许多问题提供解决办法。它能大大提高生产力，并且可以提高软件的质量和降低软件维护费用。下面简单介绍面向对象方法的优点：

(1) 易于建模。允许将问题空间中的对象直接映射到程序中。以数据为中心的设计方法更容易抓住可实现模型的更多细节。

(2) 易于维护。面向对象程序设计的模块性是与生俱来的，其核心是类的设计。数据

隐藏可以保护程序免受外部代码的侵袭，基于对象的工程可以很容易地分割为独立的部分。对象间通信所使用的消息传递技术使得对象与外部系统之间的接口描述更加简单，软件复杂度变得更加容易控制。

(3) 要扩展性好。对象在程序中是一个个相对独立的包含数据和功能的实体，程序员可以向程序中增加一个新类或对象而不会影响到其他类的操作。

(4) 代码重用。继承可以大量减少多余的代码，并扩展现有代码的用途。如果已经有一个具有某种功能的类，则可以很快地扩展这个类，创建另一个具有更多功能的类，而不必一切从头开始，从而减少软件开发时间并提高生产效率。

1.3 程序开发过程

1．程序设计步骤

程序设计是一门科学，它有规律和步骤可循。程序设计一般遵循以下步骤：

(1) 明确问题

用计算机来解决实际问题，首先要明确解决什么问题，做什么？搞清楚要解决的问题并给出问题的明确定义是解决问题的关键。

(2)系统设计

明确了问题之后，就要考虑如何解决它。由于计算机解决问题的方式本质上就是对数据进行处理，因此，先对待解决的问题进行抽象，抽取出能够反映出问题本质特征的数据并对其进行描述，即给出数据结构的设计；然后考虑对设计好的数据如何进行操作从而达到解决问题的目的，即进行算法设计。

不同的程序设计在处置数据结构和算法这两者的关系上是有所区别的。在过程式程序设计中,把数据结构设计和算法设计分开考虑,程序由算法构成,数据结构处于附属地位 而面向对象程序设计是把两者结合成对象与类来考虑，程序由对象/类构成。

(3) 用某种语言进行编程

在进行数据结构和算法设计（或对象/类设计)时,往往采用某种与具体程序设计语言无关的语言（伪代码）来进行描述，以避免一开始就陷入某种程序设计语言的一些特殊表示和实现细节中去。过多地涉及实现细节，不利于从较高抽象层次对问题本质的东西进行考虑，会使得对设计过程难以把握和理解。

当然，用伪代码描述的解决方案不能被计算机接受，必须用某种实际的程序语言把它们表示出来，即编程实现。程序员可根据采用的设计方案及实际情况来选择编程语言。如：采用功能分解的设计方案，用某种结构化程序设计语言进行编程比较合适；对于面向对象的设计方案，采用面向对象的程序设计语言来实现就更自然和方便。

选定语言后，就开始编程实现了。但是对于同一个设计，不同的人会写出不同风格的程序。风格有好坏之分，它将影响到程序的正确性和易维护性。程序设计风格取决于编程人员对程序设计的基本思想、技术以及语言精髓掌握的程度。良好的程序设计风格可以通过学习和训练来获得。

(4)测试与调试

程序写好之后，可能会含有错误。程序错误通常有三种：语法错误、逻辑错误和运行异常错误。语法错误是指程序没有按照语言的语法规则来书写，这类错误可由编译程序来发现。逻辑错误是指程序没有完成预期的功能。运行异常错误是指程序对程序运行环境的非正常情况考虑不足而导致的程序异常终止。后两类错误可能是编程阶段导致的，也有可能是设计阶段或问题定义阶段的缺陷。

程序的逻辑错误和运行异常错误一般可以通过测试来发现。测试方法有静态测试、动态测试等等。静态测试是不运行程序，通过对程序的静态分析，找出逻辑错误。动态测试是利用一些测试数据，通过运行程序观察程序的运行结果是否与预期的结果相符。值得注意的是,不管采用什么样的测试方法,都只能发现程序有错,而不能证明程序是正确的。

(5) 运行与维护

程序通过测试就可以交付使用了。由于测试程序不能保证程序无错，因此，在使用的过程中可能会不断发现程序的错误。需要对程序的错误进行改正，即维护。此外，对程序的需求的改变可能也需要对程序做修改，这也要维护。程序的维护可分为三类：正确性维护、完善性维护和适应性维护。正确性维护是改正程序中的错误；完善性维护是根据用户的要求使得程序功能更加完善；适应性维护是使程序可以移植到不同的计算平台或环境中。

2．C++ 程序开发过程

C++语言是一个计算机编程语言，利用它编写的程序并不能直接在计算机上运行，而要经过编辑、编译和链接三步生成可执行文件。

程序设计从开始到完成所需的步骤如下：

第一步　程序员进行计算机登录。

第二步　程序员用喜爱的系统的编辑器编辑源程序。

第三步　程序员把编辑好的源程序存入文件中，该文件叫做源代码文件，文件的扩展名习惯上取为 CPP。

第四步　程序员用适当的命令指示计算机编译源代码文件。

第五步　编译器将源程序代码翻译成汇编语言格式，然后传给计算机汇编器。

第六步　汇编器将汇编语言代码翻译成可重定位的目标码，转换为 OBJ 文件，并将这些目标码传送给链接器。

第七步　链接器将程序引用的所有支持例程连接在一起。这些例程存在于运行库中或早已存在于被程序所引用的预编译程序中，然后链接器产生源代码的最终版本，我们把它叫做可执行代码。该代码被传给执行程序的装载器。

第八步　运行程序。

虽然整个处理过程显得很长，但是这些步骤对于程序员来说大多数是透明的。需要程序员做的只是以下几下步骤：

第一步　产生一个包含源代码的文件。

第二步　输入编译命令，得到编译结果。

第三步　运行程序。

用流程图可以表示如图 1.2 所示。

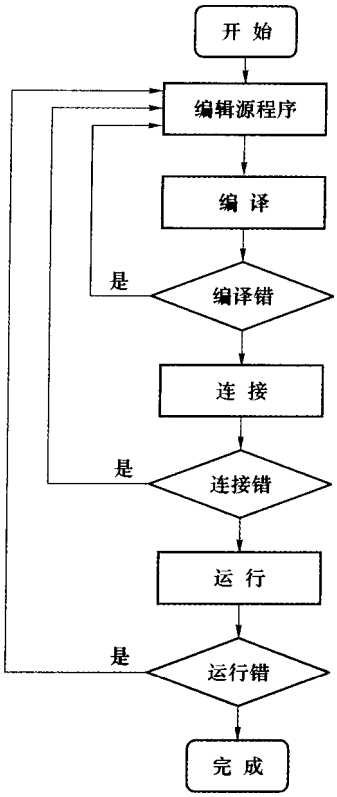

图 1.2　开发 C++ 程序的步骤

习　题　1

1.1　一个好的程序应具备哪些特点？

1.2　什么是面向对象程序设计，它与传统的结构化程序设计有什么不同？

1.3　程序开发的过程是怎样的？

第2章 C++简单程序设计

2.1 C++基础

2.1.1 C++的产生

20世纪60年代，Martin Richards设计出BCPL语言，用于开发软件系统时作记述语言。1970年，Ken Thompson在BCPL的基础上开发了B语言，美国DEC公司的PDP-7计算机中的UNIX操作系统就是用B语言开发的。1972年，Dennis Ritchie等在为PDP-11计算机开发UNIX操作系统时，对B语言做了进一步的改进，推出了更加通用的C语言。

由于C语言只是一种面向过程的编程语言，无法满足运用面向对象方法开发软件的需要。为此美国贝尔实验室的Bjarne Stroustrup博士及其同事在C语言的基础上，从Simula中引进面向对象的特征对C语言进行改进和扩充，于1980年开发出一种过程性与对象性相结合的程序设计语言。最初称之为“带类的C”，1983年正式取名为C++，在经历了不断修订后，于1994年制定了ANSI C++标准的草案，以后又经过不断完善，成为目前的C++。

2.1.2 C++的特点

C语言具有数据类型丰富、运算符丰富、使用灵活方便、生成代码质量高、程序执行效率高、可移植性好等特点。

C++是C语言的一个超集，保持与C语言的相兼性，既支持传统的结构化程序设计，又支持面向对象程序设计。因此，C++既有C语言的特点，又有自己的特点。如支持数据封装，C++类中包含私有成员、公有成员和保护成员，可通过发送消息来处理对象，允许友元破坏封装性，允许函数名和运算符重载，支持继承性，支持动态联编等等。

2.1.3 最简单的C++程序

我们从最简单的程序开始学习C++.

例2.1 一个简单的输出程序。

```
//example2_1.cpp
#include <iostream.h>
void main()
{
    cout<<"This is my first C++ program!\n";
}
```

C++的程序结构是由注释、编译预处理和程序主体组成。

我们来看程序 example2_1.cpp 。程序的第一行是：

```
//test 2_1.cpp
```

"//"用来表示注释，以增强程序的可读性，它可以用在程序的任意地方，从"//"开始，本行后面的信息在编译时被忽略。

第二行是：

```
#include <iostream.h>
```

符号"#"后面紧接关键字"include"，用来告诉编译器，在当前程序编译时还要包括另一个文件。被包括的文件名是"iostream.h"文件名用尖括号括起来表示这个文件处在一个特定的目录中(这是在安装编译器时预定义的)。

"iostream.h"含有 C++ 中的标准输入输出设备，换句话说，在需要执行输入或输出语句的每个程序中都必须包含这个文件。我们的程序既有输入又有输出，所以就包含了这个文件。

第三行：

```
void main()
```

"void"表示该函数没有返回值。main()是每个 C++ 程序的入口，必须存在。通常用左花括号和右花括号括起该函数的全部内容，表示程序的打开和关闭。如下面的语句：

```
main()
{
    <程序体>
}
```

第五行：

```
cout<<"This is my first C++ program! \n";
```

调用了"cout"，"cout"用来向标准输出设备输出。"<<"符号是输出操作符，用来输出紧接其后的变元。在这个例子中，输出一个字符串。在"iostream.h"中，"cout"被定义为标准输出流。通常认为终端就是标准输出流。因此：

```
This is my first C++ program!
```

就显示在屏幕上。

"\n"告诉"cout"在输出"This is my first C++ program!"后换行。

语句以分号结束。

注意：每个程序中有且只能有一个名为 main()的函数，它是程序的主函数。

例 2.2 cin 的使用。

```
//example2_2.cpp
#include <iostream.h>
void main()
{
    char name[20];
    cout<<"My name is :";
    cin>>name;
    cout<<name<<"\n";
}
```

第五行：

```
char name[20] ;
```

是程序所使用的一个变量的声明。“char”表示这个变量是字符数据类型的。“name 是一个数组的变量名，长度为 20 个字节。这样，字符数组的变量的大小由括在方括号中的数字来表示。

第六行：

```
cout<<"My name is :";
```

输出“My name is :”。

第七行：

```
cin>>name;
```

调用了“cin”。“cin”是标准输入流，也在“iostream.h”中定义。系统等待输入，利用“cin”后面的“>>”来给“>>”右边的数据输入值，此例所做的输入就存放到变量“name 中。

第八行：

```
cout<<name<<"\n";
```

向标准输出设备输出刚刚输入的内容，并换行。请注意，在上一个输出语句没有用转义字符。这是故意的，以使读者输入的名字紧接在提示信息的后面。

例 2.3 注释的使用。

```
//example2_3.cpp
/**********************************************************
This program adds two integers that are entered by us, then writes the sum to screen
**********************************************************/
#include <iostream.h>
void main()
{
    int x, y, z;                          //x, y, z are integers
    cout<<"Please input two integers:";
    cin>>x>>y;                            //enter x, y
    cout<<"x="<<x<<"y="<<y<<endl;  //writes x, y to screen
    z = x + y;
    cout<<"x+y="<<z<<endl;          //writes the sum to screen
}
```

第二行至第四行：

“/*”表示注释的开始；“*/”表示注释的结束。在“/*”与“*/”之间的字符均被看作注释。这种注释方式主要用于大块注释，但注意它不能嵌套使用。

第八行：

```
int x, y, z;
```

“int”说明数据 x，y，z 是整数，有关整数“int”的使用，本章稍后将有说明。

第十行：

```
cin>>x>>y;
```

输入两个整数，分别给 x 和 y，整数之间要用空格符相隔。

第十一行：

```
cout<<"x="<<x<<"y="<<y<<endl;
```

“endl”在程序中的作用等同于“\n”，不过“\n”是C语言的换行符，在C++语言中保留。

程序设计风格提示：包含文件是必须要存在的；main()函数以及其后用来打开、关闭的花括号{}也是必须要存在的，程序总是先从main()函数开始执行，所以main()函数只能在程序中出现一次；程序中用到的所有变量都必须事先声明；除了#include、#define、函数头和花括号之外的每条语句必须以分号作结束符；C++的程序书写非常自由，甚至可以把整个函数体全部书写在一行上，这对人来说却是难以理解的，在书写C++程序时一般采用比较美观的“缩进”格式书写。程序注释一般出现在函数之前和语句结束之后，这样让源代码看起来更为简洁。

2.2 数据类型和表达式

由程序操纵的实际的值叫做数据(data)，数据可以表达成多种不同的形式。数据由数字和字符组成。在C++程序中，不同的数据类型以不同的方式存取和处理。一种数据类型可以看成由两个集合构成：值集和操作(运算)集。值集描述了该数据类型包含哪些值(包括这些值的结构)；操作集描述了对值集中的值能实施哪些运算。如整数类型就是一种数据类型，它的值集就是由整数所构成的集合，它的操作集包括加、减、乘、除等运算。

数据类型往往可以分为简单数据类型和复合数据类型。简单数据类型是指构成类型值集的数据是不可再分解的简单数据，如整数类型、实数类型等；复合数据类型是指构成类型值集的数据是由其他类型的数据按照一定的方式组合而成，如数组类型、结构与联合类型等。图2.1列出了C++提供的数据类型。

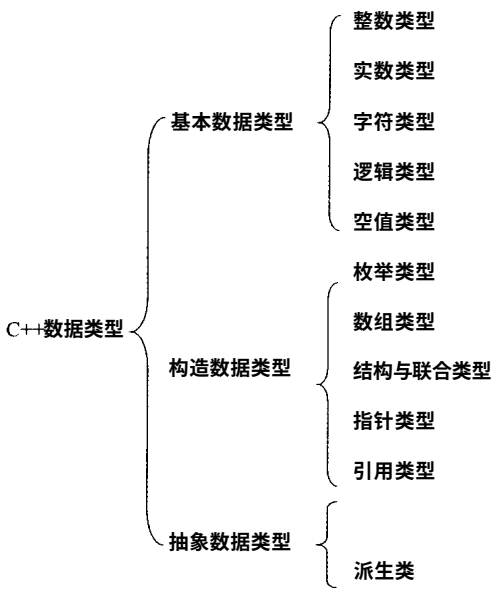

图2.1 C++数据类型

2.2.1 基本数据类型

C++基本数据类型指的是语言预定义的数据类型，称为标准或内置数据类型，它们对应着能由计算机直接表示和处理（机器指令能对它们直接进行操作）的数据类型，包括整数类型(int)、实数类型(float)、字符类型(char)、逻辑类型(bool)以及空值类型(void)。

除了上面这些基本类型之外，还有一些数据类型修饰符。修饰符用于改变基本类型的意义，以便更准确地适应各种情况的需要。修饰符有长型符（long)、短型符(short)、有符号(signed)和无符号(unsigned)。

数据类型的描述确定了其内存所占空间大小，也确定了其表示范围。表 2.1 给出了一个以 32 位计算机为例的基本数据类型加上修饰符的描述。

表 2.1 常用基本数据类型描述

类 型	长度(字节)	表示范围	备 注
bool	1	false (0) true (1)	
char	1	-128～127	$-2^7 \sim (2^7-1)$
signed char	1	-128～127	$-2^7 \sim (2^7-1)$
unsigned char	1	0～255	$0 \sim (2^8-1)$
short int	2	-32 768～32 767	$-2^{15} \sim (2^{15}-1)$
signed short int	2	-32 768～32 767	$-2^{15} \sim (2^{15}-1)$
unsigned short int	2	0～65 535	$0 \sim (2^{16}-1)$
int	4	-2 147 483 648～2 14 7 48 3 647	$-2^{31} \sim (2^{31}-1)$
signed int	4	-2 147 483 648～2 147 483 647	$-2^{31} \sim (2^{31}-1)$
unsigned int	4	0～4 294 967 295	$0 \sim (2^{32}-1)$
long int	4	-2 147 483 648～2 147 483 647	$-2^{31} \sim (2^{31}-1)$
signed long int	4	-2 147 483 648～2 147 483 647	$-2^{31} \sim (2^{31}-1)$
unsigned long int	4	0～4 294 967 295	$0 \sim (2^{32}-1)$
float	4	-3.4E38～3.4E38	7 位有效位
double	8	-1.7E308～1.7E308	15 位有效位
long double	10	-3.4E4932～1.1E4932	19 位有效位

2.2.2 关键字与标识符

1. 关键字

关键字是 C++语言的保留字，好比是已经赋予特殊含义的专用单词。它们各自有不同的使用目的，在程序中不能把它们用作别的用途。如：int、for 等。

2. 标识符

标识符是在 C++程序中为常量、变量（数据类型）和其他实体选定的名字。它的第一个字符必须是字母或下划线，随后的字符可以包含数字，如表 2.2。字符可以是大写形式，也可以是小写形式。标识符名字所允许的长度因系统而异。需参阅具体的编译器的有关资

料，查看编译器的具体规定。

表 2.2 标识符的命名规则

标识符的命名规则	举 例	
由字母、数字和下划线组成	正确的	错误的
以字母或下划线作为第一个字符，其后跟零个或多个字母、数字、下划线。	sum	1_peo //起始字符非法
	Dram_	operator //是关键字
大写字母与小写字母分别代表不同的标识符	rectangl	my $ //含有非法字符
不能与关键字相同	e_myfri	

2.2.3 变量

有些数据在程序运行过程中是可以改变的，我们称之为变量。一个变量对应着计算机中的一组内存单元，这组内存单元在 C++ 语言中用一个标识符来标识，即变量名。变量名是一个标识符，在组成的变量名字中大小写是不同的。一般情况下，变量名用小写字母。注意定义的变量名不要与关键字、库函数名、类名和对象名相同。

C++语言中使用变量前必须对变量进行声明，当有多个变量时，其间用逗号隔开。变量声明的格式如下：

数据类型变量名 1，变量名 2，…，变量名 n；

数据类型是指 C++语言中的任一合法类型，每个变量都必须属于一种类型。通过类型定义，变量被分配固定的存储空间，直到程序结束时存储空间被释放。变量的操作受类型控制，如整型变量与浮点型变量的操作是不同的。

在声明变量时，必须注意变量类型的选择。应该保证该变量中将要存储的值不突破该变量类型所能表示的最大值。

与变量有关的有两个值：一个是变量所表示的数据值，另一个是变量的地址值。

例如：

```
char a;
a = c ;
```

变量 a 被定义以后，它就在内存中对应着一个内存地址值，该内存地址保存着变量 a 的值——字符 c。在 C++ 中许多操作是针对变量的地址进行的。

变量在定义时可直接给变量一个初始值，称为变量初始化。

例如：

```
float x,y= 3.9, z=0.0;
char ch1='a', ch2='\n';
```

程序设计风格提示：变量名一般是由以小写字母开头能反映变量意义的英文单词组成，如果需要用多个单词来命名变量，那么第一个单词的首字母小写，后续单词的首字母大写，必要时可以对较长的单词进行缩写，如 totalScore。

2.2.4 常量

在程序中，有些数据在运行期间是不允许改变的，我们称之为常量。常量分为文字常量和符号常量，其中，文字常量的数据类型是由它的表示方法决定的。

1. 整型常量

整型常量就是以文字形式出现的整数,包括三种形式,见表 2.3。各种表示形式前均可加上正或负号以表示它们值的正负，正号可以省略。

表 2.3 整型常量的三种形式

形 式	组 成	举 例	备 注
十进制	由若干个 0～9 的数字组成,但不能以 0 开头	1980, -50	L(或 l)表示长整型,U(或 u)表示无符号型,如果后缀包括 L(或 l)和 U(或 u)则表示无符号长整型。
八进制	以 0 开头,由若干 0～7 的数字组成	010, -0276	
十六进制	以 0X 或 0x 开头,由若干 0～9 的数字及 A～F(大小写均可)的字母组成	0x1Fa -0X4Ab	

2. 实型常量

浮点型常量是由整数部分和小数部分组成的，只有十进制表示。实型常量有两种表示形式：一般表示形式和指数表示形式，见表 2.4。

表 2.4 实型常量的两种形式

形 式	组 成	举 例	备 注
一般表示	又称小数表示形式。使用这种表示形式时,实型常量由整数和小数两部分组成。其中的一部分在实际使用时可省略,但不允许两部分同时省去。	10.2 10. .2	默认数据类型为 double 型,如果加上后缀 F(或 f)则为 float 型,加上 L(或 l)则为 long double 型。
指数表示	表示很大或很小的实数,由尾数部分、字母 E(或 e)、指数部分三部分组成。尾数部分的表示和一般表示形式相同,指数部分必须是整数,但可正可负,当指数大于零时,正号可省。	1.2E20 .24e100	

3. 字符常量

字符常量通常是指用单引号括起来的一个字符，其数据类型是 char,在内存中以 ASCII 码形式存储。其中单引号只是用来说明被它括起来的字符是字符常量，它本身不是字符常量的内容。如'a','#'。

C++语言中，还有一种转义序列的表示方法可用来表示字符常量。这种方法是用转义符号' \ '后跟一个字符或一个 ASCII 码来表示一个单一字符。若' \ '后跟一个 ASCII 码，则表示的是该 ASCII 码所代表的字符。注意在这里 ASCII 码用八进制或十六进制表示，这里八进制和十六进制的表示与前面表示整型常量的方式不同，应无第一个“0”。例如'\X62'就表示字符'b'。较常用的转义字符见表 2.5。

表 2.5 C++ 预定义的转义序列

符 号	含 义	符 号	含 义
\a	响铃	\\	反斜杠
\n	换行	\"	双引号
\t	水平制表符(Tab 键)	\'	单引号
\v	垂直制表符(竖向跳格)	\f	换页
\b	退格(Backspace 键)	\ddd	1～3 位八进制数
\r	回车	\xhh	1～2 位十六进制数

4. 字符串常量

字符串常量又称字符串或串常量，是用一对双引号括起来的字符序列。例如："xyz"，"I am a student"，"This is a string"都是字符串。字符串中可以出现空格符、转义序列或其他字符，也可以包含 C++以外的字符，如汉字等，只要编译器支持汉字系统就行。

由于双引号在字符串中用做定界符，所以，若字符串中需要出现双引号时，则必须采用转义序列。

字符常量与字符串常量有以下几点区别：

(1) C++语言中字符串的存储与字符的存储不同，它是用一个一维字符数组来存放的，在内存中的存放也不是简单地按串中字符的顺序排放，而是在末尾加上一个'\0'，表示字符串的结束。

(2) 字符型常量用单引号括起，而串常量用双引号括起。

(3) 一个字符常量被存放在内存中只占一个字节，而串常量要占多个字节。例如字符串"I"和字符'I'对应的存储形式：

"I"

I	\0

'I'

I

(4) 字符常量与字符串常量的操作功能也不相同。例如，字符常量具有加法和减法运算，而字符串常量不具有这种运算：

'I'+3 是合法的，而"I"+3 则是非法的。

5. 布尔常量

布尔常量的数据类型为 bool，它仅有两个类型值：false(假)和 true(真)，在计算机中的值分别为 0和 1。

6. 符号常量

为了增加程序易读性，增强程序的易维护性，C++语言中的常量常用符号常量来表示，即用一个与常量相关的标识符来代替常量出现在程序中，这种相关的标识符称为符号常量。注意：符号常量在声明时一定要赋初值，而且其值在初始化后不能再改变。

C++语言提供了两种声明符号常量的方法，见表 2.6。

表 2.6 两种声明符号常量的方法

类别	用 const 声明符号常量	用 # define 声明符号常量
解 释	C++语言广泛采用的声明符号常量的方法	C语言中声明符号常量的方法。其中 # define 是预处理指令。缺点是不能显式地声明常量的类型。
形 式	const 数据类型 常量名 = 常量值; 或:数据类型 const 常量名 = 常量值;	# define 常量名 常量值
正确声明	const double pi = 3.1415926;	# define pi 3.1415926

错误声明	const double pi; //错误
	pi = 3.1415926; //错误
		# define pi 3.1415926; //错误
		pi = 3.14; //错误

程序设计风格提示：成为常量的“名字”的标识符要能反映该常量的意义。如果一个常量代表了不同的意义，也应当为该常量定义多个反映其意义的“名字”。一般情况下，符号常量名用大写字母组成，如果用多个单词来描述一个符号常量，那么这些单词之间可以用下划线分隔，如 TOTAL_SCORE。程序中应当尽量减少文字常量的出现次数，尽量多使用符号常量。

2.2.5 数组

数组是一个由若干同类型变量组成的集合。

1. 数组的声明

数组的声明形式如下：

数据类型　　数组名 [常量表达式 1] [常量表达式 2]…[常量表达式 n];

其中：数据类型给出了数组中元素的类型，这个数据类型可以是 C++语言中任何一个合法类型。数组名是一个标识符，代表着数组元素在内存中的起始地址，是一个常量，不能给它赋值。“[ ]”的个数代表着数组的维数。一个方括号表示一维数组，两个表示二维数组，…，n 个表示 n 维数组。常量表达式又称下标表达式，必须是 unsigned int 类型的整数。常量表达式 i 表示的是第 i 维的大小。上面所有常量表达式的值的乘积就是数组中含有数组元素的个数。

例如：声明一个一维数组‘ char c[5]; ”和一个二维数组‘ int i[5][2];”。

一维数组中 c 是数组名，该数组有 5 个元素，每个元素都是 char 型变量。二维数组中 i 是数组名，该数组有 10 个元素，每个元素都是 int 型变量。

2. 数组元素的访问

下标表达式也限定了数组中元素的排列次序及每个元素在数组中的位置，因此，我们可以通过数组名和下标顺利地访问每一个元素。

访问数组中元素的一般形式如下：

数组名 [下标表达式 1][下标表达式 2]…[下标表达式 n]

其中下标表达式的个数应与数组的维数相同，下标表达式的值从 0 开始。

则上面定义的数组 c 有 5 个元素：c[0], c[1], c[2], c[3], c[4] 。而数组 i 有 10 个元素：i[0][0], i[0][1], i[1][0], i[1][1], i[2][0], i[2][1], i[3][0], i[3][1], i[4][0], i[4][1]。

3. 数组的存储方式

在内存分配时，数组对应着一组顺序排放的存储单元，数组的每个元素按照声明时的次序在其中存放。一维数组声明的数组在内存中的存放顺序组，就是按下标顺序存储。

例如： int i[5];

i[0]	i[1]	i[2]	i[3]	i[4]

二维数组，可以把它看成是由多个一维数组构成的。

例如： int i[2][3];

i[0][0]	i[0][1]	i[0][2]	i[1][0]	i[1][1]	i[1][2]

三维数组，可把它看成是由多个二维数组构成的。

例如： int i[2][3][4];

i[0][0][0]	i[0][0][1]	i[0][0][2]	i[0][0][3]	i[0][1][0]	i[0][1][1]
i[0][1][2]	i[0][1][3]	i[0][2][0]	i[0][2][1]	i[0][2][2]	i[0][2][3]
i[1][0][0]	i[1][0][1]	i[1][0][2]	i[1][0][3]	i[1][1][0]	i[1][1][1]
i[1][1][2]	i[0][1][3]	i[1][2][0]	i[1][2][1]	i[1][2][2]	i[1][2][3]

4. 数组元素的初始化

在进行数组声明时，也可以给其中部分或全部元素赋初值。

对于一维数组，初始化有以下两种形式：

(1) 全部元素都初始化

如：int i[5]={1,2,3,4,5} 则在数组中有 i[0]=1,i[1]=2,i[2]=3,i[3]=4,i[4]=5。

(2)部分元素被初始化

如：int i[5]={1,2} 则在数组中有 i[0]=1,i[1]=2 ,其他元素则未赋值,程序一般都自动给它们赋 0。

对于全部元素都被初始化的情况，还有如下声明形式：

```
int i[]={1,2,3,4,5};
```

这种情况在声明时可不说明数组元素的个数。元素的个数与初始化数据的个数相同。

多维数组的初始化与一维数组相似，需要注意的是必须按照前面所讲的存储顺序列出数组元素的值。

例如：int i[2][3]={1,2,3,4,5,6};⇔int i[2][3]={ {1,2,3} , {4,5,6}} ;

初始化如下：i[0][0]=1,i[0][1]=2,i[0][2]=3,i[1][0]=4,i[1][1]=5 , i[1][2]=6。

多维数组也可以只初始化部分元素，例如：int i[2][3]={1,2,3,4 } ;即 i[0][0]=1, i[0][1]=2,i[0][2]=3,i[1][0]=4 ,其他取默认值 0。

如果在声明的同时给元素赋初值，最多可以省略第一维中的下标个数，例如：

```
int i[][3]={ {1,2,3}, {4,5,6}};⇔int i[2][3]={ {1,2,3}, {4,5,6}};
```

但不能将所有中的下标个数都省略，如

```
int i[][]={ {1,2,3}, {4,5,6}};
```

是错的。

此外，在赋初值时，初始值表中的数据个数不能多于数据元素的个数。例如：

```
int i[5]={1 ,2,3, 4,5, 6};
```

就是错误的,程序在编译时,系统会报错。

与数组元素的初始化不同，在给数组元素赋值时，必须逐一赋值。例如：对于下述的数组初始化：

```
int a[3]={1,2,3};      ⇔      int a[3]
                              a[0]=1;  a[1]=2;  a[2]=3;
```

例 2.4 一维数组的使用。

```
#include<iostream.h>
void main()
{
```