USTC

Chapter 3
Objects, types, and values

王子磊 (Zilei Wang)
Email: zlwang@ustc.edu.cn
http://vim.ustc.edu.cn

Overview

- 所有的编程任务都不可避免的需要对数据进行操作，本章重点介绍：
  - 字符串及字符串的输入输出
  - 整数及整数的输入输出
  - 类型与对象
  - 类型安全

输入和输出

程序注释

```
// read first name:
#include "std_lib_facilities.h"          // our course header

int main()
{
    cout << "Please enter your first name (followed " << "by 'enter'):\n";
    string first_name;
    cin >> first_name;
    cout << "Hello, " << first_name << '\n';
}
```

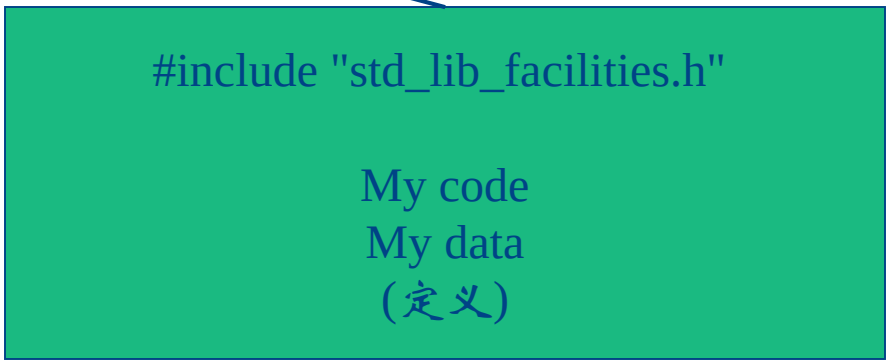
行注释

- 同一行可以输出多个值，通过连接输出符<<即可
- 引入一个变量的语句是声明(后文介绍变量的深层含义)
- main函数中的 ***return 0*** 可以省略 (部分编译器不可以，不建议)

源文件

std_lib_facilities.h:

Myfile.cpp:



- “std_lib_facilities.h” 是本课程定义的头文件

输入与类型

- 读取一个变量
  - 此处为： **first_name**
- 每个变量都有一个类型
  - 此处为： **string**
- 变量的类型决定了
  - 它在内存中的存储大小
  - 变量上能够进行的操作
    - 每个类型具有不同的操作集
- 此处，**cin>>first_name;** 从终端读取一个字符串，直到出现空白符
  - 包括：空格、tab、新行...

字符串输入

```
// read first and second name:
int main()
{
    cout << "please enter your first and second names\n";
    string first;
    string second;
    cin >> first >> second;                    // read two strings
    string name = first + ' ' + second;        // concatenate strings
                                               // separated by a space
    cout << "Hello, "<< name << '\n';
}
```

- 此处省略了头文件 #include “std_lib_facilities.h” 以关注重点内容
  - 后文不加说明都认为省略，在实际程序中需要添加上
  - 同理还有keep_window_open();
- 输入也可以连续读取多个字符 (通过连续 >> 操作)
- 字符串可以进行 “+” 操作：”first second”

整数

```
// read name and age:
int main()
{
    cout << "please enter your first name and age\n";
    string first_name;              // string variable
    int age;                        // integer variable
    cin >> first_name >> age;       // read
    cout << "Hello, " << first_name << " age " << age << '\n';
}
```

- 不同类型的变量可以连续输入，但输入值必须是合法的
  - 输入操作时类型敏感的，变量类型和值类型需要一致
- 如：输入 wang 18，则输出为(注意空格)：

  Hello, wang age 18

整数与字符串

- 字符串
  - cin >> 读入(直到空白符)
  - cout << 写出
  - + 连接
  - += s 在结尾追加字符串 s
  - ++ 错误
  - - 错误
  - …
- 整数和浮点数
  - cin >> 读入一个数字
  - cout << 写出
  - + 加
  - += n 加n
  - ++ 加1
  - - 减
  - …

➢ 变量类型决定了哪些操作符是合法的，以及代表的含义
  - 同一操作符能够表示不同含义称之为“操作符重载”("operator overloading")

命名

- C++程序中的命名方式
  - 以字母开头，可包括字母、数字和下划线
    - 合法：**x, number_of_elements, Fourier_transform, z2**
    - 非法命名：
      - **12x**
      - **timetomarket**
      - **main line** (操作系统允许但C++不允许)
    - 不要以下划线开始进行命名 (尽管是合法的): **_foo**
      - 这些是为系统实现预留的，以免冲突
  - 用户不能以预留的关键字 (代表特别的语法) 进行命名
    - *e.g.*:
      - **int**
      - **if**
      - **while**
      - **double**

命名建议

- 选择有意义的名字
  - 非常用的缩写和简称使人容易迷惑
    - **mtbf, TLA, myw, nbv**
  - 短名字可能更合适 (meaningful)
    - 编程中的习惯表达:
      - **x** 表示一个临时变量
      - **i** 表示循环的索引
  - 不建议使用过长的名字
    - Ok:
      - **partial_sum**
      - **element_count**
      - **staple_partition**
    - Too long:
      - **the_number_of_elements**
      - **remaining_free_slots_in_the_symbol_table**

关于编码风格

- 不同单位和组织有自己的编码风格
  - 包括文件组织、变量命名、注释、代码结构等
  - 上页命名建议是BS针对变量命名而建议的一般性要求
    - 大部分编码风格也都是这样要求的
  - Google C++ style
    - http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml
  - 参考书
    - 《高质量C/C++编程》
    - 《代码大全》——Microsoft
  - 须选择一种风格，并作为一种习惯保持

简单的算术运算

```
// do a bit of very simple arithmetic:

int main()
{
    cout << "please enter a floating-point number: "; // prompt for a number
    double n;                                          // floating-point variable
    cin >> n;
    cout << "n == " << n
         << "\nn+1 == " << n+1                         // '\n' means "a newline"
         << "\nthree times n == " << 3*n
         << "\ntwice n == " << n+n
         << "\nn squared == " << n*n
         << "\nhalf of n == " << n/2
         << "\nsquare root of n == " << sqrt(n)       // library function
         << endl;                                      // another name for newline
}
```

A simple computation

```
int main()                    // inch to cm conversion
{
    const double cm_per_inch = 2.54;    // number of centimeters per inch
    int length = 1;                     // length in inches
    while (length != 0)                 // length == 0 is used to exit the program
    {                                   // a compound statement (a block)
        cout << "Please enter a length in inches: ";
        cin >> length;
        cout << length << "in.  = "
             << cm_per_inch*length << "cm.\n";
    }
}
```

- While语句会重复执行，直到判断条件为false
  - 输入：10，输出：10in. = 25.4cm.

类型

- C++ 语言本身提供了一些类型
  - *e.g.* **bool, char, int, double**
  - 称之为内建类型 (“built-in types”)
- 但程序员可以定义C++新类型
  - 称之为用户自定义类型 (“user-defined types”)
  - *e.g.* **enum, struct, class,** 后续内容会逐渐展开
- C++ 标准库提供了一些常用类型
  - *e.g.* **string, vector, complex**
  - 本质上说, 这些属于用户自定义类型
    - 在库中预先定义好，便于程序员的使用
    - 再一次理解什么是代码库 (工具)

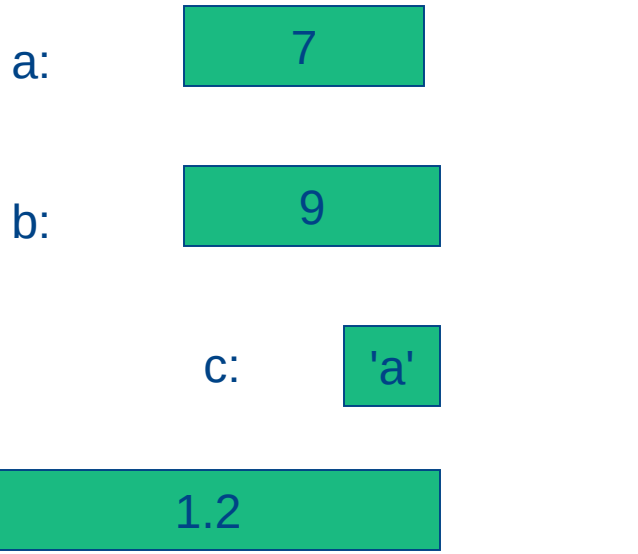
类型及字符常量

- 内建类型
  - Boolean type
    - **bool**
  - Character types
    - **char**
  - Integer types
    - **int**
      - and **short** and **long**
  - Floating-point types
    - **double**
      - and **float**
- 标准库类型
  - **string**
  - **complex<*Scalar*>**

- Boolean literals
  - **true false**
- Character literals
  - **'a', 'x', '4', '\n', '$'**
- Integer literals
  - **0, 1, 123, -6, 0x34, 0xa3**
- Floating point literals
  - **1.2, 13.345, .3, -0.54, 1.2e3, . 3F, .3F**
- String literals **"asdf"**, **"Howdy, all y'all! "**
- Complex literals
  - **complex<double>(12.3,99)**
  - **complex<float>(1.3F)**

声明和初始化

```
int a = 7;
int b = 9;
char c = 'a';
double x = 1.2;
string s1 = "Hello, world";
string s2 = "1.2";
```

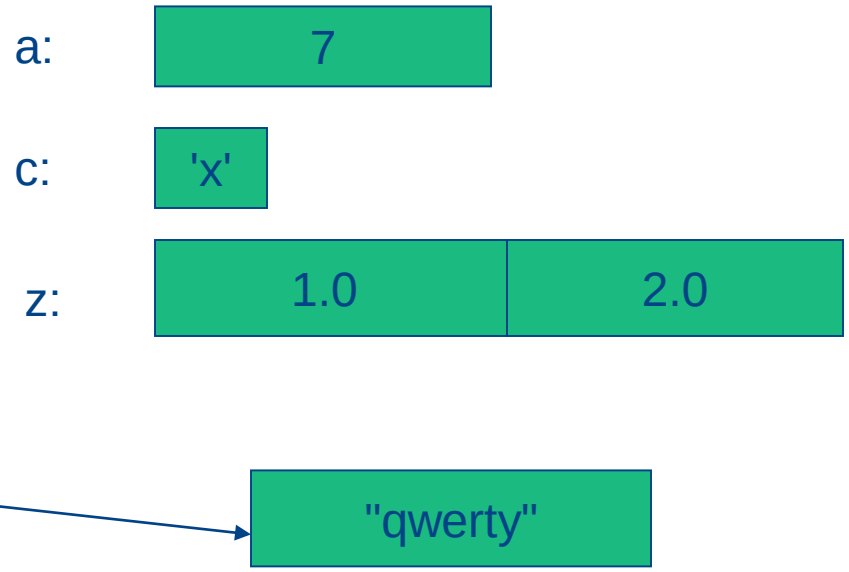


- 声明是命名一个对象的一条语句
- 初始化与赋值采用相同的操作符号
  - 但初始化总是从类型描述开始，赋值不是

对象

- 对象是一块具有某种类型的内存空间
- 变量是一个命名的对象
- 声明用来命名一个对象

```
int a = 7;
char c = 'x';
complex<double> z(1.0,2.0);
string s = "qwerty";
```



对象总结

- Type
  - 内存空间的大小和执行的操作，是必须的→编译器
- Name
  - 用于检索内存空间 → 编译器(符号表)
    - 字面常量是没有命名的对象，但具有默认的类型
    - 变量是命名的对象，包括字符常量(编译器)
- Value
  - 内存空间上实际的数值
    - 初始时由初始化值给定，后根据赋值操作进行修改
    - 操作变量实际上是操作其对应内存空间上的数值

类型安全

- 类型安全是编程语言规则(编译器规则)
  - 每个对象必须有对应的类型(包括常量和变量)
    - 变量初始化后才能开始使用
    - 只有符合声明类型的操作才能够应用在变量上
    - 每个有效的操作都会为变量赋一个新的合法值
- 静态类型安全(理论上)
  - 违反类型安全规则的程序无法编译通过
    - 编译器报错每个可能的地方(不全是)
- 动态类型安全(理论上)
  - 在运行阶段能够探测编写的程序是否违反类型规则
    - 额外的代码(典型的为"the run-time system")负责探测每个可能的违反规则之处(编译器没有查出来的)

C++类型安全

- 类型安全是相当理想化的
  - 编程时尽最大努力不要违反类型安全规则
  - “当你编程时，编译器是你最好的朋友”
- C++不是完全静态类型安全的
  - 目前，还没有常用的语言是完全静态类型安全的
  - 完全的静态类型安全不便于程序员的自由表达
- C++ 不是完全动态类型安全的
  - 很多语言是动态类型安全的，如脚本语言 (不需要编译过程)
  - 完全的动态类型安全特性限制对程序细节的控制能力，其产生的代码效率较低 (大而慢)
- 但在本课程上，我们学习的内容基本上都是类型安全的
  - 如果不是，会进行特别说明

赋值和加法

```
// changing the value of a variable
int a = 7;        // a variable of type int called a
                  // initialized to the integer value 7
a = 9;            // assignment: now change a's value to 9

a = a+a;          // assignment: now double a's value

a += 2;           // increment a's value by 2

++a;              // increment a's value (by 1)
```

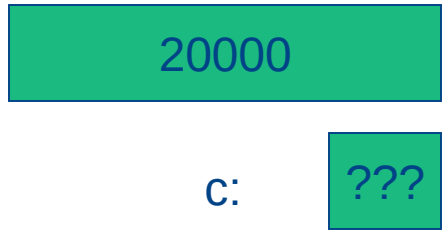
a:
7
9
18
20
21

- 操作就是对变量值的改变！

类型安全——隐式窄化

注意：C++本身不会禁止你将一个大的值赋给一个小值的变量(部分编译器可能会发出警告)，但这是类型不安全的

```
int main()
{
    int a = 20000;
    char c = a;
    int b = c;
    if (a != b)                    // != means "not equal"
        cout << "oops!: " << a << "!=" << b << '\n';
    else
        cout << "Wow! We have large characters\n";
}
```



- “宽化”赋值是类型安全的 (bool → char → int → double)
- 在自己的机器上尝试一下，该程序中 b 的输出值是什么？

类型安全——忘记初始化

注意：C++不会禁止你在初始化之前去使用该变量(一般情况下编译器会给出警告)

```
int main()
{
    int x;          // x gets a "random" initial value
    char c;         // c gets a "random" initial value
    double d;       // d gets a "random" initial value
                    //     – not every bit pattern is a valid floating-point value
    double dd = d;          // potential error: some implementations
                            // can't copy invalid floating-point values
    cout << " x: " << x << " c: " << c << " d: " << d << '\n';
}
```

- 切记：总是初始化你使用的任何变量！(函数的输入变量除外)
  - 包括临时变量、全局变量、类成员变量、复合变量等
  - 养成一个好的编程习惯

对象的技术细节

- 在内存中，所有的东西都只是比特串(二进制)，而类型决定这些比特位代表的含义

  (bits/binary) **01100001** is the int **97** is the char **'a'**
  (bits/binary) **01000001** is the int **65** is the char **'A'**
  (bits/binary) **00110000** is the int **48** is the char **'0'**

```
char c = 'a';
cout << c;    // print the value of character c, which is a
int i = c;
cout << i;    // print the integer value of the character c, which is 97
```

- 实际上，这与现实世界中的情况是类似的：
  - “42” 代表什么意思?
  - 必须加上单位以后，你才能知道它是什么
    - 米? 英尺? 摄氏度? 秒?门牌号?高度 or 长度是多少米? …

关于效率

- 目前，不必担心效率(“efficiency”)的问题
  - 重点考虑编码的正确性和代码的简洁
- C++是从C演化的，它是一种系统化的编程语言
  - C++的内建数据类型可以直接映射到计算机内存中
    - **char** 存储为一个字节 **byte**
    - **int** 存储为一个字 **word** (在32位机上通常为4个字节)
    - **double** 则存储在一个浮点寄存器中(通常为8个字节)
  - C++的内建操作符可以直接映射为机器指令
    - 整数加 + 通过机器指令“整数加”操作来实现
    - 整数赋值 = 通过简单的“拷贝”操作来实现
  - C++ 提供有效机制，使编程能够直接访问大部分现代硬件提供的接口设施
- 利用内建类型，C++能够方便地建立安全、优美和高效的新类型和新操作(用户自定义)
  - *e.g.*, **string**
  - 后续内容将会介绍更多这方面的内容

一点哲学考虑

- 像其它工程科学一样，编程也需要考虑很多折中问题
- 你可能有很多理想的目标，但在实际情况下，它们往往是互相矛盾和冲突的，这时就需要折中考虑，程序真正需要的是什么？
  - 类型安全
  - 运行时性能
  - 能够在指定的平台上运行
  - 具有跨平台运行的能力
  - 与其它代码和系统的兼容性
  - 能够简单的构建
  - 易维护
- 不要吝啬时间去保证正确性并进行测试
- C++默认情况下，注重类型安全和可移植性

> C++的设计哲学参见：
> 《C++语言的设计和演化》

Another simple computation

```
// inch to cm and cm to inch conversion:

int main()
{
    const double cm_per_inch = 2.54;
    int val;
    char unit;
    while (cin >> val >> unit) {      // keep reading
        if (unit == 'i')              // 'i' for inch
            cout << val << "in == " << val*cm_per_inch << "cm\n";
        else if (unit == 'c')         // 'c' for cm
            cout << val << "cm == " << val/cm_per_inch << "in\n";
        else
            return 0;                 // terminate on a "bad unit", e.g. 'q'
    }
}
```

- while将循环提示输入，需要终止符结束该循环
  - Ctrl+Z for windows， Ctrl+D for Unix-like

Next

- 下一章将讨论：
  - 表达式
  - 语句
  - 调试
  - 简单的错误处理
  - 程序构建的简单规则