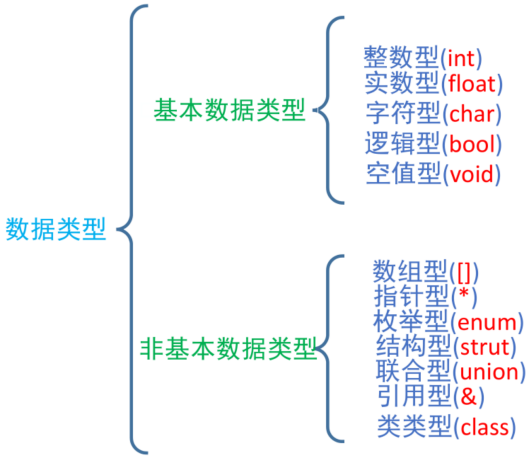
第二讲 C++程序设计基础

C++ 的数据类型



- 基本数据类型修饰符有四种，分别为long、short、signed、unsigned
- 整型数据的长度随着系统的不同而不同，在16位系统下，长度与短整型相同，在32位系统下，其长度为32位。 默认int为有符号整型signed int
- whar_t是C/C++的字符类型，是一种扩展的存储方式，whar_t数据类型一般为16位或32位，超过char类型

字符集和标识符

字符集

字符集是构成C++程序语句的最小元素，C++程序语句（字符串除外），只能由字符集中的字符构成

- 字符集组成如下：
  - 26个小写字母：a～z
  - 26个大写字母：A～Z
    - 10个数字：0～9 数字：0～9
    - 标点和特殊字符：+ - * / , : ; ? \ ” ’ ~ | ! # % & ( ) [ ] { } ^ < > 空格
    - 空字符：ASCII码为0的字符，用作字符串的结束符

标识符

由字母、下划线和数字组成的字符序列，用来命名C++程序中的常量、变量、函数、语句标号及类型定义符等

- 可以大小写字母或下划线开头，第1个不能是数字
  - 大写字母和小写字母分别代表不同的标识符（区分大小写）
  - 不能是C++的关键字

  如：Aa、ABC、A_Y、ycx11、_name是合法标识符;而5xyz、m.x、!abc、x-y是非法标识符。

常量与变量

常量

常量通常指在程序运行过程中不能被改变的量。常量可以是任何的基本数据类型，可以为整型数字、浮点数字、字符、字符串和布尔值。常量包括字面常量，字符常量，符号常量等多种类型

字面常量

程序中表示的直接参加运算的数，被称为字面常量或字面值、常量。字面是指程序中直接用符号表示的数值，而不是机器码
字面常量的类型可以根据书写形式来区分的，例如15，-2.2，'a',"hello"等都是字面常量，它们的类型分别为：整型、浮点型、字符型、字符串型，每个字面常量的字面本身就是它的值

字符常量

字符常量所表示的值是字符型变量所包含的值

- 字符常量可以赋值给字符变量，如"char b = 'a'"，但不能把一个字符串常量赋给一个字符变量，也不能对字符串常量赋值（char a = "asdasdasdasd" 或者 "asd" = "pwq" 都是被禁止的）
- 字符串中的字符依次存储在内存中一块连续的区域内，并且把空字符' \0'自动附加到字符串的尾部作为字符串的结束标志。故字符个数为n的字符串在内存中应占(n+1)个字节

普通字符常量

用一对单引号括起来的一个字符就是字符型常量，如‘a’，‘T’都是合法的字符常量

转义字符常量

在C++语言中，有一些字符用于控制输出或编译系统本身保留，无法作为字符常量来表示。为了表示这些特殊字符，C++中引入了转义字符的概念。

C++语言规定，采用反斜杠后跟一个字母来代表一个控制字符，反斜杠后的字符不再作原有的字符使用，而转义为具有某种控制功能的字符，称为转义字符。

转义序列	含义
\\	反斜杠字符
\'	'字符
\"	"字符
\?	?字符
\a	警报铃声
\b	退格键
\f	换页符
\n	换行符
\r	回车
\t	水平制表符
\v	垂直制表符
\ddd	任意字符
\xhh	任意字符

符号常量

用一个标识符来表示一个常量，称其为符号常量。符号常量在使用之前必须先定义（宏定义）

```
#define 标识符 常量值
```

- #define定义的符号常量，末尾不要有分号，是预处理语句
- 习惯上符号常量的标识符用大写字母表示，变量标识符用小写字母表示
- 使用符号常量的好处是，含义清楚， 能做到一改全改

变量

变量用于保存程序运算过程中所需要的原始数据、中间运算结果和最终结果，其值是可以改变的量。

变量有类型、名字和值三个要素。通过定义变量，程序给该变量一个标识符作为变量名，指定该变量的数据类型，并根据数据类型分配存储空间的大小。某个变量的值被改变后，将一直保持到下一次被改变。

变量作用域

在函数体内部

又称为局部变量，不同函数体内部可以定义相同名称的变量，而互不干扰

形式参数

当定义一个有参函数时，函数名后面括号内的变量统称为形式参数

```
int func(int x, int y)
{
        return x + y;
}
```

全局变量

在所有函数体的外部定义的变量，其作用范围是整个程序，并在整个程序运行期间有效

"::同名变量"：对被隐藏的同名全局变量进行访问

```
int x = 123;
void f()
{
        {
                int x;
                x = 1;          // 修改局部变量
                ::x = 2;        // 修改全局变量
        }
        x = 3;                  // 修改全局变量
}
```

变量赋值

定义变量的同时，也可以给它赋予一个初值，说明它代表的数据是什么，称为变量的初始化

```
int a = 3, b = 6*(3+5);
double sum = 0.618;
char c1 = 'a', c2 = 'b';
```

若需要对多个变量赋同样的初值，则有下面两种写法：

1. `int a = 8, b = 8, c = 8;`
2. `int a,b,c = 8; a = b = c;`

- 但不可以使用 `int a = b = c = 8`，这样是错误的
- 如果定义变量时没有赋初值，则该变量的值是不可预测的，即对应的存储单元内容是不确定的，若该变量参与运算则会导致程序的逻辑错误

常变量

在定义变量时，如果加上关键字const，则变量的值在程序运行期间不能改变，这种变量称为常变量(constant variable)。

常变量定义语句同变量定义语句类似，其语法格式为：`const 类型名 常变量名=<初值表达式>`

```
const int A = 3;
// 用const来声明这种变量的值不能改变，指定其值始终为3
```

明晰：

- 常变量通常定义为大写变量名称
- const和 #define 都可以达到相同的效果，但是一般采用的是 const ，因为 #define只是简单的「全文替换」，而const可以参与编译器的类型检查，有利于程序运行时安全
- 常变量具有作用域
- #define 不是常变量，因此没有作用域，如不 #undef，则一直有效

运算符与表达式

对各种类型的数据进行加工的过程称为运算，表示各种不同运算的符号称为运算符，参与运算的数据称为操作数据。

表达式是运算符与数据连接起来的表达运算的式子，表达式也称运算式

根据运算对象个数分类：

- 一元运算符（单目运算符）
- 二元运算符（双目运算符）
- 三元运算符（三目运算符）

根据运算符功能分类：

- 算术
- 赋值
- 关系
- 逻辑
- 条件

运算符在此不罗列。

浮点比较方法

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

int main(int argc, char const *argv[])
{
    double a = 3.3333, b = 4.4444;
    cout << (a + b == 7.7777) << endl;
    return 0;
}
// 结果:        0
//                     7.7777
```

上面这个程序很显然是正确的逻辑，用来比较7.7777与3.3333+4.4444是否相等。

这里需要引入一种浮点判定的方法，而不能直接使用==作为判断语句。

fabs() 判断法（通过绝对值）

判断两个实数是否相等的正确方法是：判断两个实数之差的绝对值是否小于一个给定的允许误差数，如：

```
#include <iostream>
#include <math.h>
using namespace std;

bool isEq(double a, double b, double error = 1e-6) {
    return fabs(a - b) <= error;
}

int main(int argc, char const *argv[])
{
    double a = 3.3333, b = 4.4444;
    cout << isEq(a + b, 7.7777) << endl;
    cout << a + b << endl;
    return 0;
}
// 结果:        1
//                     7.7777
```

其中`fabs()`方法是包含于头文件`math.h`的，注意引用再使用

条件运算符6+

三目运算符`?:`是一种非常常用的运算符。其一般形式为：

表达式1 ：表达式2 ： 表达式3

通过一行实现`int max(int, int)`：

```
#include <iostream>
using namespace std;
int maxx(int a, int b)
{
    return a >= b ? a : b;
}
int main(int argc, char const *argv[])
{
    cout << maxx(1,2) << endl;
    return 0;
}
// 结果==:      2==
```

逗号运算符

由逗号运算符构成的表达式称为逗号表达式，其一般形式为：

表达式1，表达式2……，表达式n

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

int main(int argc, char const *argv[])
{
    int b=2 ,c=7 ,d=5;
    int a1, a2;
    a1 = (++b,c--,d+3);
    a2 = ++b,c--,d+3;
    cout << "a1: " << a1 << "\ta2: " << a2 << endl;
    return 0;
}
// 结果:         a1: 8   a2: 4
```

结果很奇怪，为什么是 8 和 4 呢?

- 对于第一行代码，括号中有一个逗号表达式，包含三个部分，用逗号分开，所以最终的值应该是最后一个表达式的值，也就是d+3，为8，所以a1=8
- 对于第二行代码，由于赋值运算符比逗号运算符优先级高，所以整个看成一个逗号表达式，这时逗号表达式中的三个表达式为a2=++b、c--、d+3，所以虽然最终整个表达式的值也为8，但a2=3。

类型转换

隐式类型转换

对数据类型不一致的两个运算量，系统会进行数据类型转换。即将其中的一个低级别类型的数据向另一个高级别类型的数据转换（按空间大小和数值范围），然后才进行相应的算术运算，运算的结果为高级别类型。这种按照固有的规则自动进行的内部转换称为隐式类型转换。

```
double f = 7/3;
int a = 'a' + 3;
```

强制类型转换

有两种用法：

- `(类型名称) 待转换数据`：(int) 123.1
- `类型名称 (待转换数据)`：int(123.1)

  注意：
- 隐式类型转换时类型之间必须相容
- 各整型之间，浮点与整型之间是相容的；
- 整型与指针之间，浮点与指针之间是不相容；
- 采用强制类型转换将高级别类型数据（按空间大小和数值范围）转换为低级别类型数据时，可能数据精度会受到损失

程序流程控制结构

略，就是if，*else*，switch这些东西。