C++编程(2)

唐晓晟
北京邮电大学电信工程学院

第四章 类型和声明

- 类型
- 布尔量
- 字符类型
- 整数类型
- 浮点类型
- 大小
- void
- 枚举
- 声明
- 忠告
- 练习

4.1 类型

- C++中，每个名字都有一个与之相关联的类型，类型决定了可以对其进行什么样的操作，并决定这些操作如何解释
- 基本类型有：
- 布尔、字符、整数、浮点、枚举、void、指针、数组、引用、数据结构和类
- 可以分为内部类型和用户自定义类型

4.2 布尔量(bool)

- 布尔量bool，可以具有两个值false或者true之一，用于表示逻辑运算的结果。
- 根据定义 true = 1, false = 0，实际：非0即true
- bool变量和整数（包括指针）可以相互转换

```
void f(int a, int b)
{
    bool bl = a == b;
    // …
}

bool b = 7;

int i = true;

void g()
{
    bool a = true;
    bool b = true;
    bool x = a + b;
    bool y = a | b;
}
```

4.3 字符类型

- 类型为char的变量可以保存具体实现所用的字符集里面的一个字符
- 常用字符类型都至少包括8bit，可以保存256种不同的数值
- 字符集一般都采用ISO-646的某个变形，比如说ASCII(ANSI3.4-1968)

以下假设都是不安全的

- 8位字符集中共有不超过127个字符(有的字符集提供了255个字符)
- 不存在超出英语的字符(大部分欧洲语言提供了更多的字符)
- 字母字符是连续排列的(EBCDIC在”i”和”j”之间留有空隙)
- 写C++所需要的每个字符都是可用的(有些国家的字符集中没有提供{}[]\，C++提供的解决方法有关键字、二联符和三联符)

解决受限的字符集

关键字(keywords) C++		二联符(digraph) C++		三联符(trigraph) C	
and	**&&**	**<%**	**{**	??=	#
and_eq	**&=**	**%>**	**}**	??(	[
bitand	**&**	**<:**	**[**	??<	{
bitor	**|**	**:>**	**]**	??/	\
compl	**~**	**%:**	**#**	??)	]
not	**!**	**%:%:**	**##**	??>	}
or	**||**			??'	^
or_eq	**|=**			??!	|
xor	**^**			??-	~
xor_eq	**^=**				
not_eq	**!=**				

digraph and trigraph Examples

使用C的trigraph

```
#include <stdio.h>
void main(int argc, char *argv??(??))
??<
    if (argc>1 && argv??(0??) != NULL)
        printf("Hello, %s!??/n",argv??(1??));
    else printf("Hello, world!??/n");
    return 0;
??>
```

使用C++的digraph

```
#include <stdio.h>
main(int argc, char *argv<::>)
<%
  if (argc> 1 and argv<:0:> != NULL)
     printf("Hello, %s!??/n",argv<:1:>);
  else
     printf("Hello, world!??/n");
  return 0;
%>
```

转义字符

名字	ASCII名字	C++名字
换行符	NL(LF)	\n
水平制表符	HT	\t
垂直制表符	VT	\v
退格符	BS	\b
回车符	CR	\r
换页符	FF	\f
警铃符	BEL	\a
反斜线符	\	\\
问号	?	\?
单引号	‘	\’
双引号	“	\”
八进制	ooo	\ooo
十六进制	hhh	\xhhh

字符类型

- char, signed char, unsigned char
- char和int之间的相互转换(由实现决定)
- wchar_t类型，用于保存更大的字符集里的字符，如Unicode，具体大小由实现决定，该名字来源于C语言，是一个typedef
- 对于字符类型可以进行算术和逻辑运算
- 字符文字量(字符常量)，如'a','0','\n'等
- 宽字符文字量形式为L'ab'，引号中的字符个数由wchar_t决定，其类型是wchar_t

字符类型示例

八进制	十六进制	十进制	ASCII	大字符集
\6	\x6	6	ACK	\uXXXX 或者 \UXXXX
\60	\x30	48	‘0’	\u1e2b
\137	\x05f	95	‘_’	\uXXXX等价于\U0000XXXX

```
char v1[] = “a\xah\129”;
char v2[] = “a\xah\127”;
char v3[] = “a\xad\127”;
char v4[] = “a\xad\0127”;
```

字符类型程序示例

```
char c = 255; // 0xff
int i = c;
```

问题：i = ?

解决方法：使用unsigned char 或者 signed char标明

但是有些函数只接受普通char类型

```
void f(char c, signed char sc, unsigned char uc)
{
    char* pc = &uc;
    signed char* psc = pc;
    unsigned char* puc = pc;
    psc = puc;
    // 以上四条语句均产生错误
}
```

4.4 整数类型

- 三种形式：int, signed int (signed) , unsigned int (unsigned)
- 三种大小：short int (short), int, long int (long)
- 与char不同，int 总是有符号的
- 整数文字量：7 1234 1234567890000
- 十六进制：0x3f 八进制：022
- L(大小写)结尾表示long，U(大小写)结尾表示unsigned，如：100UL

4.5 浮点类型

- 表示浮点数，即包括小数部分的数
- 三种大小：float, double, long double
- 浮点文字量：1.23 .23 0.23 1. 1.2e10
- 注意：浮点文字量的中间不能出现空格
- 以F(大小写)结尾表示float，不加后缀缺省表示为double类型，若需要long double类型的数字，可以加上L(大小写)后缀

4.6 大小

- C++基本类型的某些方面是由实现确定的，比如说int
- 为了保证程序的移植性，建议在所有的可能之处都使用标准库的功能
- 提供多种整数类型、无符号类型、浮点类型的原因是希望使程序员能够利用各种硬件特性。比方说：不同的硬件对不同的基础类型处理时，存储的需求、存储访问时间和计算速度方面存在明显的差异。

大小

C++对象大小由char的大小的倍数表示，定义char的大小为1，
则其他数据类型大小可以用sizeof运算符获得
1=sizeof(char)<=sizeof(short)<=sizeof(int)<=sizeof(long)
1 <= sizeof(bool) <= sizeof(long)
sizeof(char) <= sizeof(wchar_t) <= sizeof(long)
sizeof(float) <= sizeof(double) <= sizeof(long double)
sizeof(N) = sizeof(signed N) = sizeof(unsigned N)

numeric_limits

```
#include <limits>
#include <iostream>
using namespace std;
int main()
{
    cout << "largest float == " <<
    numeric_limits <float> :: max()
    << ", char is signed == " <<
    numeric_limits<char> :: is_signed
    << endl;
}
```

4.7 void

- void类型是一个语法上的基本类型
- void可以用来标明一个函数并不返回数值，这可以理解为一个“伪返回类型”，可以加强语法的规范性
- 也可以用做指向不明类型的对象的指针的基础类型

例如：

```
void x;// error
void f();
void* pv;
```

4.8 枚举

- 枚举是一种类型，可以保存一组由用户刻画的值，一旦定义，使用起来很像整数类型
- 例如：
- enum{ ASM, AUTO, BREAK };
  - 定义三个枚举型的整数常量并赋值，默认方式下，数值从0开始
  - ASM = 0, AUTO = 1, BREAK = 2
  - 枚举也可命名，如enum keyword{ ASM, AUTO, BREAK }; 这样产生一个新的数据类型keyword

枚举类型的取值范围

- 如果某个枚举中的所有枚举符的值均非负，该枚举的表示范围就是”0”到”2的k次幂-1”，k是能够使得所有枚举符号都在此范围内的最小的2的幂
- 如果存在负的枚举符号值，该枚举的取值范围就是”负2的k次幂”到”2的k次幂-1”之间

```
enum e1 {dark, lignt};    // 0:1
enum e2 {a = 3, b = 9}; // 0:15
enum e3 {min = -10, max = 1000000}; //-1048576:1048576
```

枚举数值和整数之间的转换

- 一个整型数可以显式地转换到一个枚举值(前提是转换的结果位于该枚举范围之内，否则是无定义的)
- 默认情况下，枚举值可以转换到整数参加算术运算，此外，由于枚举是用户自定义类型，用户可以为枚举定义自身的操作，例如定义++或者<<

枚举数值转换示例

```
enum flag { x=1, y=2, z=4, e=8}; // 0:15

flag f1 = 5; // wrong

flag f2 = flag(5); // ok

flag f3 = flag(z | e); // ok, z|e = 12

flag f4 = flag(99); //not defined
```

4.9 声明

- 一个名字(标识符)在被使用之前必须声明，以刻画清楚它的类型，通知编译器这个名字所引用的是哪一类实体，如：double d;
- 通常，大部分声明同时也是定义
- C++中，每个命名实体必须有恰好一个定义，当然，可以有很多声明，但是所有声明必须类型完全一致

```
int error_number = 1;

extern int error_number;
extern int error_number;
```

```
int count;
int count; // Error

extern int error_number;
extern short error_number;
```

4.9.1 声明的结构

- 声明包括四部分：可选的“描述符”、基础类型、声明符、可选的初始式

描述符：**virtual extern**等，表明被声明事物的某些非类型的属性

char* kings[] = {“Seleucus”, “Ptolemy”};

基础类型是：**char**

声明符是：***kings[]**，常用的声明符有***,*const,&**等前缀，以及**[],()**等后缀，一般情况下，后缀的声明运算符比前缀的声明运算符约束力更强

初始式是：**{… …}**

注意：声明中不能没有类型

const c=7; // Error

gt(int a, int b){ return (a>b)?a:b;} //Error

4.9.2 声明多个名字

- 单个声明中可以有多个名字，多个名字可以使用逗号分隔
- 例如：int x,y; // int x; int y;
- 需要注意：运算符只作用于一个单独的名字，不是同一声明中随后写的所有名字
- 例如：
- int* p, y; // int *p; int y; 不是int* y;

4.9.3 名字

- 名字(标识符)由一系列字母和数字组成，第一个字符必须是字母(包括下划线)
- C++对于名字中的字符个数没有任何限制，但是字符数目可能会受编译器或者连接器的限制
- 以下划线开头的名字是保留变量
- 编译器读程序时，总是设法寻找最长的能够组成一个名字的字符序列

- C++中，大小写字符是区分的，选择名字时，尽量避免选择不容易区分的字符，比如说1,I,L,l,O,0等
- 较少使用的名字可以选择相对比较长，频繁使用的名字可以选择比较短的
- 名字的选择应该反映一个实体的意义，而不是实现方式(phone_book比number_list更好)
- 设法保持统一风格

4.9.4 作用域

- 一个声明将一个名字引入一个作用域
- 作用域指这个名字的有效区间，包括局部作用域({}语句块)和全局作用域(整个文件内部可用，如果是在所有函数、类、名字空间之外定义)
- 注意同名局部变量和全局变量之间的覆盖问题

作用域程序示例

```
int x;
void f()
{
    int x;
    x = 1;
    {
        int x;
        x = 2;
    }
    x = 3;
}

int* p = &x;
```

```
int x;
void f2()
{
    int x = 1;
    ::x = 2;
    //
}
```

```
int x = 1;

void f()
{
    int y = x;
    int x = 22;
    y = x;
}
```

```
void f5(int x)
{
    int x;
}

//Error
```

4.9.5 初始化

- 如果为一个对象提供了初始式，这个初始式将确定对象的初始值，如果没有，全局的、名字空间的、局部静态的对象将被自动初始化为适当类型的0
  - int a; // a = 0
  - double d; // d = 0.0
- 局部对象(自动对象)和在自由存储区内建立的对象(动态对象或者堆对象)不会用默认值做初始化
  - ```
    void f()
    {
        int x; //x没有定义良好的值
    }
    ```
    ```

4.9.6 对象和左值

- 一个对象就是存储中一片连续的区域
- 左值就是引用某个对象的表达式，其原本意思是“某个可以放在赋值号左边的东西”，但是引用了某个常量的左值除外
- 没有被声明为常量的左值常常被称做是可修改的左值
- 左值在很多语法书中或者C++语言中称为lvalue

4.9.7 typedef

- typedef就是为类型声明了一个新名字，而不是声明一个指定类型的对象，更不是生成新的类型
  - typedef char* Pchar;
    Pchar p1,p2;
    char* p3 = p1;
- typedef的另一类使用是将对某个类型的直接引用限制到一个地方
  - typedef int int32;
    typedef short int16;

4.9.10 忠告

- [1]保持较小的作用域；
- [2]不要在一个作用域和它外围的作用域里采用同样的名字；
- [3]在一个声明中(只)声明一个名字；
- [4]让常用的和局部的名字比较短，让不常用的和全局的名字比较长；
- [5]避免看起来类似的名字；
- [6]维持某种统一的命名风格；
- [7]仔细选择名字，反映其意义而不是反映实现方式；

忠告

- [8]如果所用的内部类型表示某种可能变化的值，请用typedef为它定义一个有意义的名字；
- [9]用typdef为类型定义同义词，用枚举或类去定义新类型；
- [10]切记每个声明中都必须描述一个类型(没有“隐式的int”)；
- [11]避免有关字符数值的不必要假设；
- [12]避免有关整数大小的不必要假设；
- [13]避免有关浮点类型表示范围的不必要假设；
- [14]优先使用普通的int而不是short int或者long int；

忠告

- [15]优先使用double而不是float，或者long double；
- [16]优先使用普通的char而不是signed char或者unsigned char；
- [17]避免做出有关对象大小的不必要假设；
- [18]避免无符号算术；
- [19]应该带着疑问去看待signed到unsigned，或者从unsigned到signed的转换；
- [20]应该带着疑问去看待从浮点到整数的转换；
- [21]应该带着疑问去看待向较小类型的转换，如将int转换到char；