# 变量和基本类型

2021年10月25日　17:46

1、大多数计算机以2的整数次幂个比特作为块来处理内存，可寻址的最小内存块称为“字节(byte)”，存储的基本单元称为“字(word)”，它通常由几个字节组成

2、类型char和类型signed char不同

3、尽量用double而不用float，float精度不够而且与double计算代价相差无几

4、字符串实际长度比内容多1（末尾有'\0'）

5、

首先了解一下字符串的前后缀：　　后缀：u或U　表示该字面值为无符号类型

| 前缀：u | unicode16字符 | l或L　表示该字面值的类型至少为long |
|---|---|---|
| U | unicode32字符 | ll或LL　表示该字面值的类型至少为long long |
| L | 宽字符 | f或F　表示该字面值为float类型 |
| u8 | utf-8 | 前后缀可以交叉结合使用：后缀UL时，表示无符号长整型。 |

字符用单引号‘’，字符串用双引号“”。

十进制：20　八进制：020　十六进制：0x20或者0X20

6、虽然有默认初始化，但最好还是要初始化变量

7、extern

如果想声明一个变量而非定义它，就在变量名前添加关键字extern，而且不要显式地初始化变量：

extern int i;　//声明i而非定义i

在函数体内部，如果试图初始化一个由extern关键字标记的变量，将报错

8、引用

引用一旦初始化完成，就无法绑定到另外的对象，因此引用必须初始化

引用并非对象，相反的，它只是为一个已经存在的对象所起的另一个名字

因为引用本身不是一个对象，所以不能定义引用的引用

引用类型的初始值要和与之绑定的对象严格匹配比如都是int或都是double（除特殊情况）

引用类型的初始值必须是一个对象，不能是常数之类的

9、因为引用不是对象，没有实际地址，所以不能定义指向引用的指针

10、指针类型要和指向的对象严格匹配（除特殊情况）

11、尽量用nullptr而不是NULL

12、初始化所有指针，不清楚指向哪就初始化为nullptr或0

13、void*是一种特殊的指针类型，可用于存放任意对象的地址。具体百度

14、引用本身不是一个对象，因此不能定义指向引用的指针。但指针是对象，所以存在对指针的引用。

int *p;

int *&r=p;　//r是一个对指针p的引用

要理解r的类型到底是什么，最简单那的办法就是从右向左阅读r的定义。离变量名最近的符号（此例中是&r的符号&）对变量的类型有最直接的影响，因此r是一个引用。声明符的其余部分用以确定r引用的类型是什么，此例中的*说明r引用的是一个指针。最后，声明的基本数据类型部分指出r引用的是一个int指针。

15、const对象一旦创建后其值就不能改变，所以const对象必须初始化

16、const对象被设定为仅在文件内有效。如果想在多个文件之间共享const对象，必须在变量的定义之前添加extern关键字

17、const的引用

可以把引用绑定到 const 对象上，就像绑定到其他对象上一样，我们称之为**对常量的引用**（reference to const）。与普通引用不同的是，对常量的引用不能被用作修改它所绑定的对象：

```
const int ci = 1024;
const int &r1 = ci;  // 正确：引用及其对应的对象都是常量
r1 = 42;               // 错误：r1 是对常量的引用
int &r2 = ci;          // 错误：试图让一个非常量引用指向一个常量对象
```

因为不允许直接为 ci 赋值，当然也就不能通过引用去改变 ci。因此，对 r2 的初始化是错误的。假设该初始化合法，则可以通过 r2 来改变它引用对象的值，这显然是不正确的。

初始化和对const的引用

2.3.1 节（第 46 页）提到，引用的类型必须与其所引用对象的类型一致，但是有两个例外。第一种例外情况就是在初始化常量引用时允许用任意表达式作为初始值，只要该表达式的结果能转换成（参见 2.1.2 节，第 32 页）引用的类型即可。尤其，允许为一个常量引用绑定非常量的对象、字面值，甚至是个一般表达式：

```
int i = 42;
const int &r1 = i;        // 允许将 const int&绑定到一个普通 int 对象上
const int &r2 = 42;       // 正确：r1 是一个常量引用
const int &r3 = r1 * 2;   // 正确：r3 是一个常量引用
int &r4 = r1 * 2;         // 错误：r4 是一个普通的非常量引用
```

对 const 的引用可能引用一个并非 const 的对象

必须认识到，常量引用仅对引用可参与的操作做出了限定，对于引用的对象本身是不是一个常量未作限定。因为对象也可能是个非常量，所以允许通过其他途径改变它的值：

```
int i = 42;
int &r1 = i;              // 引用 r1 绑定对象 i
const int &r2 = i;        // r2 也绑定对象 i，但是不允许通过 r2 修改 i 的值
r1 = 0;                   // r1 并非常量，i 的值修改为 0
r2 = 0;                   // 错误：r2 是一个常量引用
```

r2 绑定（非常量）整数 i 是合法的行为。然而，不允许通过 r2 修改 i 的值。尽管如此，i 的值仍然允许通过其他途径修改，既可以直接给 i 赋值，也可以通过像 r1 一样绑定到 i 的其他引用来修改。

18、要想存放常量对象的地址，只能使用指向常量的指针

之前提到指针类型必须与其所指对象类型一致，但是有两个例外。第一种例外情况是：允许令一个指向常量的指针指向一个非常量对象，但是不能通过这个指针改变对象的值。

总结一下17、18点：const类型的引用可以引用无论是否是const的对象，甚至引用常数也可以（普通引用不能初始化引用常数）。然而const类型的变量只能被const类型的引用所引用。

19、const指针

指针常量const pointer：

int *const p=&i;　//p将一直指向i，但是i的值可以修改

指针常量必须初始化，而且一旦初始化完成，则它的值（也就是存放在指针中的那个地址）就不能再改变了。

常量指针pointer to point：

const int *p=&i;　//i的值不可以修改，但p可以指向别处

指向常量的指针常量：

const int *const p=&i;　//p将一直指向i，且i的值也不能修改

20、顶层const表示指针本身是个常量，底层const表示指针所指的对象是一个常量。用于声明引用的const都是底层const

21、顶层const的拷贝不受限制，但是底层const的拷贝的对象必须具有相同的底层const资格。一般来说：非常量可以赋值给常量，反之则不行。

22、常量表达式（const expression）是指值不会改变并且在编译过程就能得到计算结果的表达式。

一个对象（或表达式）是不是常量表达式由它的数据类型和初始值共同决定，例如：

```
const int max_files = 20;          // max_files 是常量表达式
const int limit = max_files + 1;   // limit 是常量表达式
int staff_size = 27;               // staff_size 不是常量表达式
const int sz = get_size();         // sz 不是常量表达式
```

C++11 新标准规定，允许将变量声明为 **constexpr** 类型以便由编译器来验证变量的值是否是一个常量表达式。声明为 constexpr 的变量一定是一个常量，而且必须用常量表达式初始化：

```
constexpr int mf = 20;            // 20 是常量表达式
constexpr int limit = mf + 1;     // mf + 1 是常量表达式
constexpr int sz = size();        // 只有当 size 是一个 constexpr 函数时
                                  // 才是一条正确的声明语句
```

一般来说，如果你认定变量是一个常量表达式，那就把它声明成constexpr类型。

23、在constexpr声明中如果定义了一个指针，限定符constexpr仅对指针有效，与指针所指的对象无关：

const int *p=nullptr;　//p是一个指向整型常量的指针

constexpr int *q=nullptr;　//q是一个指向整数的指针常量（与int *const q相同）

24、类型别名

方法一、关键字typedef

typedef double wages;　//wages是double的同义词

typedef wages base,*p;　//base是double的同义词，p是double*的同义词

方法二、使用别名声明来定义类型的别名

using db=double;　//db是double的同义词

25、auto类型说明符，可以让编译器自己去分析表达式所属类型。

auto一般会忽略掉顶层const，而底层const会被保留。

auto定义的变量必须有初始值。

26、decltype：作用是选择并返回操作数的数据类型

decltype 处理顶层 const 和引用的方式与 auto 有些许不同。如果 decltype 使用的表达式是一个变量，则 decltype 返回该变量的类型（包括顶层 const 和引用在内）：

```
const int ci = 0, &cj = ci;
decltype(ci) x = 0;       // x 的类型是 const int
decltype(cj) y = x;       // y 的类型是 const int&，y 绑定到变量 x
decltype(cj) z;           // 错误：z 是一个引用，必须初始化
```

decltype 和 auto 的另一处重要区别是，decltype 的结果类型与表达式形式密切相关。有一种情况需要特别注意：对于 decltype 所用的表达式来说，如果变量名加上了一对括号，则得到的类型与不加括号时会有不同。如果 decltype 使用的是一个不加括号的变量，则得到的结果就是该变量的类型；如果给变量加上了一层或多层括号，编译器就会把它当成是一个表达式。变量是一种可以作为赋值语句左值的特殊表达式，所以这样的 decltype 就会得到引用类型。

答案

了一对括号，则得到的类型与不加括号时会有不同。如果 decltype 使用的是一个不加括号的变量，则得到的结果就是该变量的类型；如果给变量加上了一层或多层括号，编译器就会把它当成是一个表达式。变量是一种可以作为赋值语句左值的特殊表达式，所以这样的 decltype 就会得到引用类型：

```
// decltype 的表达式如果是加上了括号的变量，结果将是引用
decltype((i)) d;      // 错误：d 是 int&，必须初始化
decltype(i) e;        // 正确：e 是一个（未初始化的）int
```


切记：decltype((variable))（注意是双层括号）的结果永远是引用，而 decltype(variable)结果只有当 variable 本身就是一个引用时才是引用。

答案

a类型为int，值为3

b类型为int，值为4

c类型为int，值为3

d类型为int&，值为3

27、

练习 2.37：赋值是会产生引用的一类典型表达式，引用的类型就是左值的类型。也就是说，如果 i 是 int，则表达式 i=x 的类型是 int&。根据这一特点，请指出下面的代码中每一个变量的类型和值。

```
int a = 3, b = 4;
decltype(a) c = a;
decltype(a = b) d = a;
```

28、auto与decltype主要区别：

1.如果使用引用类型，auto会识别为其所指对象的类型，decltype则会识别为引用的类型

2.decltype(())双括号的区别

29、预处理变量（#define）无视C++语言中关于作用于的规则。整个程序中的预处理变量包括头文件保护符必须唯一。