# 表达式与操作符

## 整型常量表达式(integer constat expression)

整型常量是在编译期确定的值，不能在运行期更改。整型常量表达式是由常量组成并且在编译期已经计算为常量的表达式。

整型常量表达式是仅由以下元素组成的表达式：

1. 字面值
2. 枚举值
3. 常量变量(被const限定的变量)
4. 枚举类型或是静态类型的数据成员(data members)
5. 被强制类型转换为整型的数值
6. sizeof表达式，其参数不能是可变长度的数组

对于可变长度的数组,sizeof表达式在运行期才能取得结果，因此以可变长度数组为参数的sizeof表达式不是整型常量表达式。

我们必须在以下情况使用整型常量表达式：

1. 作为下标操作符的参数时(下标操作符即[ ],在[ ]内的必须为一个常量表达式)
2. 在switch case语句中，‘case :’ 之后的必须是一个常量表达式
3. 在枚举类型中，作为枚举常量的数值
4. 在位于宽度说明符中
5. 在预处理的条件编译语句#if中

## 作用域解析运算符(operator::)

作用域解析运算符可以用限定隐藏名称，方便用户使用。如果名称空间作用域或者全局作用域名称被当前程序块中的相同名称的显式声明所“遮掩”，那么可以使用作用域解析运算符的一元版本，例如：

int count = 0;

int main(void) {

int count = 0; //声明一个局部变量count，其名称会掩盖全局变量count

::count = 1; // 这里使用的是::运算符的一元版本，使得该名称被解析为全局名称count

count = 2; // 该名称被解析为本程序块显式声明的局部变量count

return 0;

}

此外，我们可以通过使用作用域解析运算符来访问类中的静态成员，如：

class X{

public:

static int count;

};

X::count = 5; //访问class X的static data member

int main(){

int count = 10; //显式声明整型变量count，这是一个局部变量

cout <<X::count<<endl; //这里的count是类X中的静态数据成员

}

## 函数调用

函数调用也是一个表达式，包含函数名、后跟函数调用运算符()。如果函数定义了形参，则需要将参数值罗列在调用运算符()内。参数列表可以包含由逗号分隔任意数目表达式。参数列表也可为空。

函数调用表达式的类型就是函数的返回类型，此类型可以是完全类型，也可以是不完全类型，甚至可以是void类型。

在C中，函数的返回值总是一个右值

在C++中，函数的返回值取决于以下情况：

1. 如果结果的类型是一个左值引用，或者是一个函数类型的右值引用，则返回结果是左值
2. 如果结果类型是一个右值引用，那么返回结果是一个消亡值
3. 其余情况均为右值

原文：rvalue references to function,按照字面意思是对于函数的右值引用。但是函数本身没有生存期或者存储时间，所以左右值对它均没有意义。而函数指针是一个左值，可以被左值引用绑定。对于函数本身，尽管左右值含义不明确，但是可以被右值引用绑定，如下：

int Solve(int a){

return a;

};

function<int(int)> &&p = Solve;

//正确，一个右值引用可以绑定函数，p是一个函数指针类型的右值引用。

function<int(int)> &f = p;

//正确，左值引用可以绑定到函数指针上

funtion<int(int)> &e = Solve;

//错误，左值引用不能绑定函数

当使用(函数指针类型的)右值引用绑定函数时，那么函数指针依然是一个左值(这点看上去可能有些绕…)。为什么说这个函数指针还是一个左值呢，因为我们确实可以打印出这个函数指针右值引用的地址。所以，我们完全可以把绑定函数类型的右值引用当成左值引用一样使用。

## 成员表达式

#### 句号(dot)运算符

句号运算符可以用于访问类、结构或者联合体的成员。句号运算符后跟可能的标识符或者函数名，句号运算符左侧必须是一个后缀表达式，这个后缀表达式可以是一个类、结构体、联合体的对象。而句号表达式后的名称必须是该对象的成员。

表达式的值是选定成员的值，如果成员是左值，则这个表达式也是左值。如果后缀表达式被类型限定，那么相同的类型限定也会应用于本表达式中的指定成员。

#### 箭头(->)运算符

箭头运算符应用于使用指针访问类、结构体、联合体成员的情况下。后缀表达式后跟->运算符，运算符后接续可能的标识符或者函数名，用于指定访问的成员。后缀表达式必须是一个类、结构体、联合体的对象的指针，而后续的名称必须为该对象的成员。

和句号表达式一样，表达式的值是选定成员的值，如果成员是左值，那么这个表达式也是左值。如果后缀表达式被类型限定，那么相同的类型限定也会应用于本表达式中的指定成员。

## 一元表达式(Unary expressions)

#### 递增和递减操作符(++和--)

递增操作符将标量操作数的值+1，或者如果操作数是指针，则将操作数按其指向对象的大小递增。操作数接收增量操作的结果，操作数必须是算术或者指针类型可修改的左值。

可以在操作数后放置++。前置递增操作符会先将操作数递增，再返回该操作数；后置递增操作符会先生成一个操作数的副本，将该副本返回，然后递增原操作数。

对于递减操作符的情况，和递增操作符的情况类似。

#### 一元加号和减号运算符(+，-)

一元加号运算符保持操作数的值，操作数可以是任何算术类型或者指针类型。返回结果不是左值。

结果类型和整型提升后的操作数类型相同。

对于一元减号运算符的情况类似。

#### 逻辑非运算符(!)

逻辑非运算符可以确定操作数的结果是0(false)还是非0(true)

在C中：

如果操作数计算结果为0，则表达式将生成值1(true)；如果操作数结果为非0值，则表达式将生产值0(false)

在C++中：

如果操作数的计算结果为false(0)，则表达式生成结果true；如果操作数的计算结果为true(非0)，则表达式将生成值0。操作数会被隐式转换为bool类型，返回结果也是bool类型。

#### 按位取反运算符(~)

按位取反运算符产生操作数的按位取反(补)结果。在结果的二进制表示中，每个位在操作数的二进制表示法中具有相反值。操作数类型必须是整型，结果的类型与操作数相同，但不是左值。

#### 地址运算符 (&)

&运算符生成指向操作数的指针。操作数必须是左值、函数标识符或者限定名，它不能是位字段。

如果操作数是左值或者函数，则结果类型是指向表达式类型的指针。

如果操作数是限定名(比如X::a,X是一个类名,a是其中一个non-static member)而成员是非静态的，那么结果是指向类的成员指针。

如果p\_to\_y被定义为指向int的指针，y被定义为int类型的变量，那么下述表达式将变量y的地址赋予p\_to\_y:

int \*p\_to\_y = &y;

#### 解引用/间接运算符(\*)

本运算符确定指针类型操作数所引用的值。操作数不能是指向不完全类型的指针。如果操作数指向某个对象，则该操作将生成一个引用该对象的左值。如果操作数指向一个函数，则结果会是C中的函数标识符，或者在C++中，是指操作数指向的对象的左值。数组和函数被转换为指针。

操作数的类型决定结果的类型，比如操作数是指向int的指针，那么结果类型就是int。

不要将间接寻址运算符应用于包含无效地址(如空)的任何指针，此时结果是未定义的。

#### 补齐操作符(\_\_alignof\_\_)

补齐运算符是C99和标准C++的语言扩展，它返回在其操作数的对齐中使用的字节数。操作数可以是表达式或带圆括号的类型标识符。如果操作数是表示左值的表达式，则由alignof返回的数字表示已知左值具有的对齐方式。表达式的类型在编译时确定，但不计算表达式本身。如果操作数是类型，则数字表示目标平台上类型通常需要的对齐方式。

比如:

class A{

int a;

char b;

}

由于A中最大类型是int，\_\_alignof\_\_(A)的返回值是4;

如果将int a改成int \*a,结果就变为8了(64位机器中指针大小为8字节)

注意，\_\_alignof\_\_是C99的扩展，标准C中并不包含此操作符

#### sizeof操作符

sizeof运算符生成操作数的字节大小，操作数可以是表达式、类型。

应用于类型名的sizeof产生该类型对象所需使用的内存量，包括任何内部或者尾部的alignment所占用的字节数。

#### \_\_typeof\_\_操作符

typeof运算符返回参数的类型，可以是表达式或者类型。

typeof的结果会保留顶层的引用或者限定，和decltype的效用一样。

注意，如果typeof的参数如果是表达式，则类型与表达式结果一样(比如，参数是一个const int&,那么返回类型也是const int&,但是如果表达式是一些变量计算的结果，则表达式结果仅仅是一个int类型的右值)

注意，\_\_typeof\_\_也是C99的扩展，标准C中并不包含此操作符

#### \_\_real\_\_ 和 \_\_imag\_\_运算符

使用\_\_real\_\_运算符可以表示复数类型的实部的值

\_\_imag\_\_运算符可以表示复数类型的虚部的值

注意，\_\_real\_\_和\_\_imag\_\_都是C99的扩展，标准C中并不包含这两个运算符。