C++关键字全集整合

asm	do	if	return	typedef
auto	double	inline	short	typeid
bool	dynamic_cast	int	signed	typename
break	else	long	sizeof	union
case	enum	mutable	static	unsigned
catch	explicit	namespace	static_cast	using
char	export	new	struct	virtual
class	extern	operator	switch	void
const	false	private	template	volatile
const_cast	float	protected	this	wchar_t
continue	for	public	throw	while
default	friend	register	true	
delete	goto	reinterpret_cast	try	

asm

语法:

```
asm( "instruction" );
```

允许你在你的代码中直接插入汇编语言指令， 各种不同的编译器为这一个指令允许不一致形式, 比如：

```
asm{

    instruction-sequence}
```

or `asm( instruction );`

bool

是用来声明布尔逻辑变量的；也就是说,变量要么是真，要么是假。举个例子：

```
bool done = false;
while( !done ) {
...}
```

catch

通常通过 throw 语句捕获一个异常.

class

语法:

```
class class-name : inheritance-list {
        private-members-list;
             protected:
        protected-members-list;
               public:
         public-members-list;
          } object-list;
```

允许你创建新的数据类型. *class-name* 就是你要创建的类的名字,并且 *inheritance-list* 是一个对你创建的新类可供选择的定义体的表单.类的默认为私有类型成员,除非这个表单标注在公有或保护类型之下. *object-list* 是一个或一组声明对象.举个例子:

```
class Date {
  int Day;
  int Month;
  int Year;
public:
  void display();};
```

const_cast

语法:

```
const_cast<type> (object);
```

用于移除"const-ness"的数据,目标数据类型必须和原类型相同,目标数据没有被 const 定义过除外.

delete

语法:

```
    delete p;
delete[] pArray;
```

用来释放 *p* 指向的内存.这个指针先前应该被 new 调用过.上面第二种形式用于删除数组.

dynamic_cast

语法:

```
dynamic_cast<type> (object);
```

强制将一个类型转化为另外一种类型，并且在执行运行时时检查它保证它的合法性。如果你想在两个互相矛盾的类型之间转化时，cast 的返回值将为 NULL.

explicit

当构造函数被指定为 explicit 的时候,将不会自动把构造函数作为转换构造函数,这仅仅用在当一个初始化语句参数与这个构造函数的形参匹配的情况.

false

"false"是布尔型的值.

friend

允许类或函数访问一个类中的私有数据.

inline

语法:

```
inline int functionA( int i ) {
                ...}
```

请求编译器扩张一个给定的函数。它向这个函数发出一条插入代码的 call。函数里面有静态变量，嵌套的，switches，或者是递归的时候不给予内联。当一个函数声明包含在一个类声明里面时，编译器会尝试的自动把函数内联。（当 C++在编译时使用函数体中的代码插入到要调用该函数的语句之处，同时用实参代替形参，以便在程序运行时不再进行函数调用，消除函数调用时的系统开销，以提高系统的运行速度。在程序执行过程中调用函数，系统要将程序当前的一些状态信息存到栈中，同时转到函数的代码处去执行函数体的语句，这些参数保存和传递过程中需要时间和空间的开销，使得程序效率降低。但是并不是什么函数都可以定义为内联函数，一般情况下，只有规模很小而是用频繁的函数才定义为内联函数，这样可以大大提高运行速率。）

mutable

忽略所有 const 语句.一个属于 const 对象的 mutable 成员可以被修改.

namespace

语法:

```
   namespace name {
   declaration-list;}
```

允许你创建一个新的空间.名字由你选择,忽略创建没有命名的名字空间.一旦你创建了一个名字空间,你必须明确地说明它或者用关键字 using. 例如:

```
namespace CartoonNameSpace {
  int HomersAge;
  void incrementHomersAge() {
    HomersAge++;}}
int main() {
  ...
  CartoonNameSpace::HomersAge = 39;
  CartoonNameSpace::incrementHomersAge();
  cout << CartoonNameSpace::HomersAge << endl;
  ...}
```

new

语法:

```
        pointer = new type;
pointer = new type( initializer );
     pointer = new type[size];
```

可以给数据类型分配一个新结点并返回一个指向新分配内存区的首地址. 也可以对它进行初始化.中括号中的 *size* 可以分配尺寸大小.（1.内存分配的基本形式：指针变量名=new 类型，如：int *x;x=new int;或 char *chr;chr=new char;释放内存(delete 指针变量名)：delete x;delete chr;虽然 new 和 delete 的功能和 malloc 和 free 相似，但是前者有几个优点：（1）new 可以根据数据类型自动计算所要分配的内存大小，而 malloc 必须使用 sizeof 函数来计算所需要的字节;(2)new 能够自动返回正确类型的指针，而 malloc 的返回值一律为 void*,必须在程序中进行强制类型转换;new 可以为数组动态分配内存空间如：int *array=new int[10]或 int *xyz=new int[8][9][10];释放时用 delete []array 和 delete []xyz;另外 new 可以在给简单变量分配内存的同时初始化，比如 int *x=new int(100);但不能对数据进行初始化;有时候没有足够的内存满足分配要求，则有些编译系统将会返回空指针 NULL。）

operator

语法:

```
return-type class-name::operator#(parameter-list) {
                          ...}
      return-type operator#(parameter-list) {
                          ...}
```

用于重载函数.在上面语法中用特殊符(#)描述特征的操作将被重载.假如在一个类中,类名应当被指定.对于一元的操作, *parameter-list* 应当为空, 对于二元的操作,在 operator 右边的 *parameter-list* 应当包含操作数 (在 operand 左边的被当作 this 通过).

对于不属于重载函数的operator成员,在左边的操作数被作为第一个参数,在右边的操作数被当作第二个参数被通过.

private

属于私有类的数据只能被它的内部成员访问,除了 friend 使用.关键字 private 也能用来继承一个私有的基类,所有的公共和保护成员的基类可以变成私有的派生类.

protected

保护数据对于它们自己的类是私有的并且能被派生类继承.关键字 keyword 也能用于指定派生,所有的公共和保护成员的基类可以变成保护的派生类.

public

在类中的公共数据可以被任何人访问.关键字 public 也能用来指定派生,所有的公共和保护成员的基类可以变成保护的派生类.

reinterpret_cast

语法:

```
reinterpret_cast<type> (object);
```

把一种数据类型改变成另一种.它应当被用在两种不可调和的指针类型之间.

static_cast

语法:

```
static_cast<type> (object);
```

用来在两个不同类型之间进行强制转换,并且没有运行时间检查.

template

语法:

```
template <class data-type> return-type name( parameter-list ) {
                        statement-list;}
```

能用来创建一个对未知数据类型的操作的函数模板.这个通过用其它数据类型代替一个占位符 *data-type* 来实现. 例如:

```
template<class X> void genericSwap( X &a, X &b ) {
```

```
    X tmp;
    tmp = a;
    a = b;
    b = tmp;}
int main(void) {
    ...
    int num1 = 5;
    int num2 = 21;
    cout << "Before, num1 is " << num1 << " and num2 is " << num2 << endl;
    genericSwap( num1, num2 );
    cout << "After, num1 is " << num1 << " and num2 is " << num2 << endl;
    char c1 = 'a';
    char c2 = 'z';
    cout << "Before, c1 is " << c1 << " and c2 is " << c2 << endl;
    genericSwap( c1, c2 );
    cout << "After, c1 is " << c1 << " and c2 is " << c2 << endl;
    ...
    return( 0 );}
```

this

指向当前对象.所有属于一个 class 的函数成员都有一个 this 指向. (定义一个对象，系统要给该对象分配存储空间，如果该类包含了数据成员和成员函数，就应该分别给数据和函数的代码分配存储空间。但 C++的编译系统只用一段空间来存放各个共同的函数代码段，在调用各个对象的成员函数时，都调用这个公共的函数代码。因此，每个对象的存储空间都只是该对象数据成员所占用的存储空间，而不包括成员函数代码所占有的空间，函数代码是存储在对象空间之外的。每当创建一个对象时，系统就把 this 指针初始化为指向该对象，即 this 指针的值为当前调用成员函数的对象的起始地址。每当调用一个成员函数时，系统就把 this 指针作为一个隐含的参数传给该函数。不同的对象调用同一个成员函数时，C++编译器将根据成员函数的 this 指针所指向的对象来确定应该调用哪一个对象的数据成员。)

throw

语法:

```
        try {
    statement list;}
catch( typeA arg ) {
    statement list;}
catch( typeB arg ) {
```

```
    statement list;}
        ...
catch( typeN arg ) {
    statement list;}
```

在 C++体系下用来处理异常.同 try 和 catch 语句一起使用, C++ 处理异常的系统给程序一个比较可行的机制用于错误校正.当你通常在用 try 去执行一段有潜在错误的代码时.在代码的某一处,一个 **throw** 语句会被执行, 这将会从 try 的这一块跳转到 catch 的那一块中去.例如:

```
try {
  cout << "Before throwing exception" << endl;
  throw 42;
  cout << "Shouldn't ever see this" << endl;}
catch( int error ) {
  cout << "Error: caught exception " << error << endl;}
```

true

"true"是布尔型的值.

try

try 语句试图去执行由异常产生的代码.

typeid

语法:

```
typeid( object );
```

typeid 操作返回给一个 **type_info** 定义过的对象的那个对象的类型.

typename

能用来在中 template 描述一个未定义类型或者代替关键字 class.

using

用来在当前范围输入一个 namespace.

virtual

语法:

```
   virtual return-type name( parameter-list );
virtual return-type name( parameter-list ) = 0;
```

能用来创建虚函数,它通常不被派生类有限考虑.但是假如函数被作为一个纯的虚函数 (被=0 表示)时, 这种情况它一定被派生类有限考虑.

volatile

在描述变量时使用,阻止编译器优化那些以 volatile 修饰的变量,volatile 被用在一些变量能被意外方式改变的地方,例如:抛出中断,这些变量若无 volatile 可能会和编译器执行的优化 相冲突.

void

用来表示一个函数不返回任何值,或者普通变量能指向任何类型的数据. Void 也能用来声明一个空参数表.

wchar_t

用来声明字符变量的宽度.