**2.1 如何编写函数**

返回类型 函数名 参数列表

{

函数体

}

**2.2 调用函数**

**传址和传值调用**

传值 利用局部变量复制参数，不改变原有值。不可返回局部变量的引用和地址

传址 reference/pointer 利用应用来传递参数，可以返回引用或指针

　　Int val = 1024;

　　Int \*pi = &val;

　　Int &rval = val：

注意:不管是何种方式，对于函数内部申请的局部变量，不可返回局部变量的引用和地址 ，只可以返回值，因为局部变量 地址/引用 过了函数周期无效。

**作用域及范围**

对于函数局部变量，用pointer和reference返回都会有问题，但是利用值返回就没有问题，因为返回的是副本。 他们的作用域是 **local scope**（局部变量）。如果定义在函数外部，他们的作用范围就是**file scope**（全局变量）。但是不建议用全局变量来参与函数的运算，最好可以改为参数的形式。

**动态内存管理**

上述定义 无论是local scope或者是file scope都是由系统自动管理分配。C++为程序员提供了可以动态分配的途径 为 new delete关键字 ，用法为：

New type; 或者 new type(initval\_value);

Ex: int \*pia = new int[24]; //由堆栈分配一块空间，并把这个空间的头部指针值赋给 pia。

delete [] pia; //数组的delete方法。

**2.3 提供默认参数值**

提供函数默认的参数值，可以作为函数中的判断依据，来处理函数的这个参数使用/不适用 两种情况的判断依据。

利用“参数传递”作为函数见间的沟通方式比 “将对象定义在file scope”更有利。

Ex: void bubble\_sort (vector <int> &vec,ofstream \* ofil = 0)

{ ……

//这里要用 pointer 不是reference，因为reference 不可为0

If（ofil == 0 ）{ ……}

……

}

以上便可以有两种调用方式 bubble\_sort( a); bubble\_sort(a，(ofstream \*) ofil);

默认参数定义有两个默认规则：

（1）默认参数一般放置在函数的最右边，即一参数是默认参数，它右边全为默认参数

（2）默认参数只需要指定一次，一般在于函数的声明处，包含在头文件中，不在函数的实现处。

**2.4 使用局部静态变量**

局部静态变量只会被初始化一次，运算之后研究保留计算值，其栈空间不会被优化。因此，可以在函数中可以返回局部静态变量的 reference 和pointer 。

**2.5声明inline 函数**

声明为inline函数，会要求编译器将函数在每一个调用的节点展开，适用于体积小 常被调用的函数。

**2.6 提供重载函数**

C++允许通过函数列表的不同来实现函数的重载，但是允许通过返回值的不同来实现。因为有函数情况的调用并没有返回值，便无法具体定义到具体的实现函数。

**2.7 定义并使用摸板函数**

要求增加三个display\_message() 函数，风别处理三种不同类型的vector 。

Void display\_message (const string& msg，const vector<int>& vec);

Void display\_message (const string& msg，const vector<double>& vec);

Void display\_message (const string& msg，const vector<string>& vec);

这三个函数的具体实现基本相同，只是对数据的处理方式不同，可以抽象为一个函数实现。C++为用户提供了 **function template（函数模板）**功能，用于剥离数据类型对于函数具体实现的影响。

Function template用关键字 ***template*** 开始，再用“<>”定义一个或多个标识符。

Ex: template <typename T> T表示一种已知的数据类型，在函数会根据具体调用的数据类型才会定义。由此可以重新定义上面的函数。

Void display\_message (const string& msg，const vector<**T**>& vec);

//在声明template参数时, 前缀关键字class和typename可以互换，但在使用模板参数T的内部类型名称即嵌套从属名称时只能用typename。

**2.8 函数指针带来更大的弹性**

函数指针的使用

**定义**：函数指针需要**指明函数的返回类型及参数列表**，并且函数指针名称的设定需要考虑到\*的优先级关系。

EX: const vector<int>\* (\* seq\_Ptr) (int);

**使用**：可以利用定义函数，定义函数指针数组，抽象成单个功能函数的形式来满足函数的抽象。

有返回六种不同队列的函数：

Const vector<int>\* fibon\_seq (int size);

Const vector<int>\* lucas\_seq (int size);

Const vector<int>\* pell\_seq (int size);

Const vector<int>\* triang\_seq (int size);

Const vector<int>\* square\_seq (int size);

Const vector<int>\* pent\_seq (int size);

需求：函数fibon\_elem() 可以获取某种数列中的某个值。如果没有函数指针的化就要类似上文提供五个不同的函数，但是有函数指针就可以利用函数指针访问上文6个函数，集成为一个函数。

Bool seq\_elem (int size,int& elem,const vector<int>\* (\*seq\_ptr)(int) )

{

Const vector<int>\* pseq = seq\_ptr( size);

If( !pseq ){ elem =0;return false;}

Elem = pesq[size-1];//取最后一个值了。

Return true;

}

调用： seq\_elem(5, value , fibon\_seq); //c++中函数名称可以人为是函数地址。