**3.1 指针的算术运算**

利用指针来提高函数的适用性

Template <typename T>

T\* find (const T\* array,int size ,T value );

以上适用于array 和vector，可以将结尾方式再变为末指针，以适用于更多的数据结构。

T\* find(const T\* begain, const T\* end,T value)

指针的算术运算：对于数组 ia[8] 需要明白 ia,和ia+8的地址含义。

C++对于指针打包的函数:begin()获取第一个数据首地址，end()获取最后元素的下一个地址。

**3.2 了解Iterator(泛型指针)**

**迭代器(iterator)是一中检查容器内元素并遍历元素的数据类型。**

在C++中可以由begin() 和end()函数来获取iterator。

(1) 每种容器类型都定义了自己的迭代器类型，如vector:

vector<int>::iterator iter;这条语句定义了一个名为iter的变量，它的数据类型是由vector<int>定义的iterator类型。

(2) 使用迭代器读取vector中的每一个元素：

vector<int> ivec(10,1);

for(vector<int>::iterator iter=ivec.begin();iter!=ivec.end();++iter)

{

\*iter=2; //使用 \* 访问迭代器所指向的元素

}

const\_iterator:

只能读取容器中的元素，而不能修改。

(3) iterator的算术操作：

iterator除了进行++,--操作，可以将iter+n,iter-n赋给一个新的iteraor对象。还可以使用一个iterator减去另外一个iterator.

const vector<int>::iterator newiter=ivec.begin();

vector<int>::iterator newiter2=ivec.end();

cout<<"\n"<<newiter2-newiter;

**3.3 所有容器公共操作**

标识符： == != =

函数：empty() 返回是否为空 size() 返回元素个数 clear() 删除所有元素

Begin()返回第一个元素的iterator

End()返回指向最后一个元素的下一个位置的itreator。

Insert() 将单一或者一个范围的元素插入到容器内。 -》多种重载

Erase() 将一个或者一个范围的容器删除 。-》多种重载

**3.4 使用顺序性容器**

**C++ 提供了三种顺序容器：vector list deque（双端队列）**对于三种容器的插入，取值的复杂度暂不做赘述了

产生容器对象的方法有五种（这里以vector）举例：

Vector< int> mvecto；

Vector< int> mvecto（32）；//固定大小

Vector< int> mvecto（32，2）；//大小和初值

通过一对iterator来实现和通过已有的容器实现

Int ia[8] = {0,1,2,3,4,5,6,7}

Vector< int> mvecto（ia，ia+8）；

Vector< int> mvecto2（mvecto）；

顺序容器都有的操作：pusn\_back() 填入 popback()删除

List 和 deque:push\_front() popfront() //vector 不能直接在首部加入

以上函数操作只能插入/移除对应元素，不能返回值，顺序容器通过 front() 和back() 读取首尾值。

**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。