一个容器中的所有对象必须是同一种类型。

vector不是一种数据类型，而是一个类模板。

vector对象的定义和初始化：

// vector是一个类模板

vector<int> vector\_int; // 保存类型为int，容器为空，默认构造函数

vector<int> vector\_int2(10); // 容器中包含10个元素，默认初始化为0

vector<int> vector\_int3(10, -1); // 容器中包含10个元素，初始化为-1

安全的泛型编程：

在使用vector这种类模板进行泛型编程时，例如在使用for循环操作时，采用下面的方式：

for (vector<int>::size\_type i = 0; i != vector\_int3.size(); ++i)

{

}

首先尽量使用vector<int>::size\_type来定义迭代变量，其次判断迭代变量的大小时，要尽量使用”!=”而不是”<”，最后直接在for语句头中调用size()函数，而不是将size()的值保存成一个变量，然后再调用，原因是：vector容器size()的值有可能会动态改变。

可以通过下标的方式访问和修改vector容器已存在的元素，但不能通过下标的方式添加元素到vector容器中。

vector<int> vector\_int4; // 空容器

for (vector<int>::size\_type i = 0; i != 10; ++i)

{

vector\_int4[i] = i; // 运行时会导致程序崩溃

}

vector<int> vector\_int;

迭代器：

for (int i = 0; i != 10; ++i)

{

vector\_int.push\_back(i);

}

vector<int>::iterator iterator\_int; // 定义一个迭代器

for (iterator\_int = vector\_int.begin(); iterator\_int != vector\_int.end(); ++iterator\_int)

{

std::cout << \*iterator\_int << std::endl;

}

for (iterator\_int = vector\_int.begin(); iterator\_int != vector\_int.end(); ++iterator\_int)

{

\*iterator\_int = 10; // 通过迭代器修改容器元素的值

}

const迭代器：

vector<int>::const\_iterator const\_iterator\_int; // 定义一个const迭代器

const\_iterator\_int = vector\_int.begin();

// \*const\_iterator\_int = 80; // error：不能给常量赋值

++const\_iterator\_int;

const vector<int>::iterator iterator\_const\_int = vector\_int.begin();

\*iterator\_const\_int = 80;

// ++iterator\_const\_int; // error不能修改迭代器本身

二者的区别类似于指向常量的指针和指针常量的区别。

vector容器作为函数形参：考虑到非引用形参和引用形参的区别，如果vector容器作为函数形参，应该声明为引用形参，以避免函数调用时复制容器中的每个元素。在实际应用过程中，更倾向于向函数中传递容器的迭代器。

例：程序vector\_test2

void PrintValue(vector<int>::iterator begin, vector<int>::iterator end);

int main(int argc, char\* argv[])

{

vector<int> vector\_int;

for (int i = 0; i != 10; ++i)

{

vector\_int.push\_back(i);

}

PrintValue(vector\_int.begin(), vector\_int.end());

return 0;

}

void PrintValue(vector<int>::iterator begin, vector<int>::iterator end)

{

while (begin != end)

{

std::cout << \*begin << std::endl;

++begin;

}

}