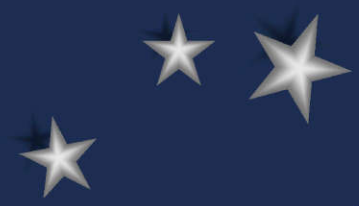
C++语言程序设计

# 第十章 C++标准模板库

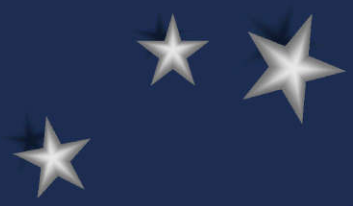
# 主要内容

- 泛型程序设计
- 与标准模板库有关的概念和术语
- C++标准模板库中的容器
- 迭代器
- 标准C++库中的算法
- 函数对象



# 泛型程序设计

- 将程序写得尽可能通用
- 将算法从特定的数据结构中抽象出来，成为通用的
- C++的模板为泛型程序设计奠定了关键的基础
- STL是泛型程序设计的一个范例
  - 容器(container)
  - 迭代器(iterator)
  - 算法（algorithms）
  - 函数对象（function object）



# 命名空间（namespace）

概念和术语

- 一个命名空间将不同的标识符集合在一个命名作用域（**named scope**）内
  - 为了解决命名冲突
  - 例如，声明一个命名空间NS：

```
namspace NS {
class File;
void Fun ();
}
```

则引用标识符的方式如下，

```
NS:: File obj;
NS:: Fun ();
```

- 没有声明命名空间的标识符都处于无名的命名空间中

# 命名空间（续）

概念和术语

- 可以用**using**来指定命名空间
  - 例如，经过以下声明：
    using NS::File;
    在当前作用域中就可以直接引用File
  - using namespace std;
    命名空间std中所有标识符都可直接引用
- 在新的**C++**标准程序库中，所有标识符都声明在命名空间**std**中，头文件都不使用扩展名

概念和术语

# 容器

- 容器类是容纳、包含一组元素或元素集合的对象。
- 异类容器类与同类容器类
- 顺序容器与关联容器
- 七种基本容器：
  - 向量（vector）、双端队列（deque）、列表（list）、集合（set）、多重集合（multiset）、映射（map）和多重映射（multimap）

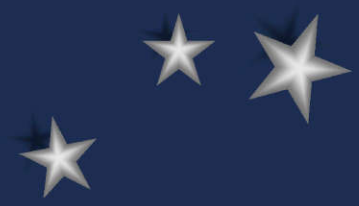
# 容器的接口

- 通用容器运算符
  - ==，!=，>，>=，<，<=，=
- 方法（函数）
  - 迭代方法
    - begin()，end()，rbegin()，rend()
  - 访问方法
    - size()，max_size()，swap()，empty()

# 适配器

概念和术语

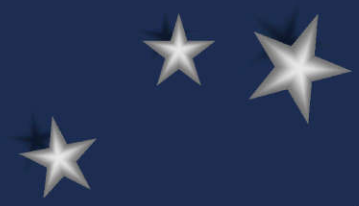
- 适配器是一种接口类
  - 为已有的类提供新的接口。
  - 目的是简化、约束、使之安全、隐藏或者改变被修改类提供的服务集合。
- 三种类型的适配器：
  - 容器适配器
    - 用来扩展7种基本容器，它们和顺序容器相结合构成栈、队列和优先队列容器
  - 迭代器适配器
  - 函数对象适配器。



# 迭代器

概念和术语

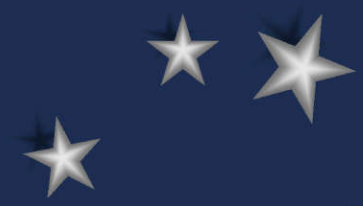
迭代器是面向对象版本的指针，它们提供了访问容器、序列中每个元素的方法。



# 算法

概念和术语

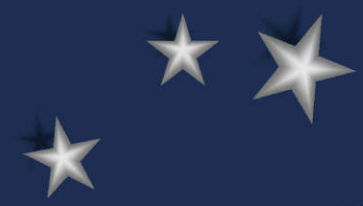
- C++标准模板库中包括70多个算法
  - 其中包括查找算法，排序算法，消除算法，记数算法，比较算法，变换算法，置换算法和容器管理等等。
- 这些算法的一个最重要的特性就是它们的统一性，并且可以广泛用于不同的对象和内置的数据类型。



# 顺序容器

容器

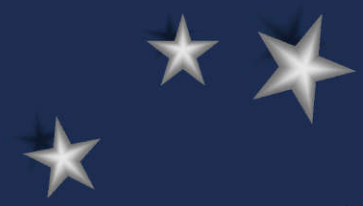
- 顺序容器的接口
  - 插入方法
    - push_front()，push_back()，insert()，运算符“=”
  - 删除方法
    - pop()，erase()，clear()
  - 迭代访问方法
    - 使用迭代器
  - 其他顺序容器访问方法（不修改访问方法）
    - front()，back()，下标[]运算符



# 顺序容器——向量

容器

- 向量属于顺序容器，用于容纳不定长线性序列（即线性群体），提供对序列的快速随机访问（也称直接访问）
- 向量是动态结构，它的大小不固定，可以在程序运行时增加或减少。
- 例10-1
  - 求范围2~N中的质数，N在程序运行时由键盘输入。



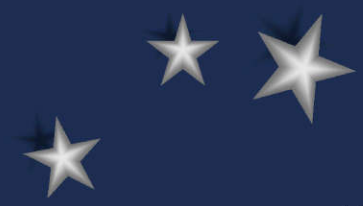
```
//10_1.cpp
#include <iostream>
#include <iomanip>
#include <vector> //包含向量容器头文件
using namespace std ;
int main()
{
  vector<int>  A(10);
  int n;
  int primecount = 0, i, j;
  cout<<"Enter a value>=2 as upper limit: ";
  cin >> n;
  A[primecount++] = 2;
```

```
    for(i = 3; i < n; i++)
    { if (primecount == A.size())
       A.resize(primecount + 10);
      if (i % 2 == 0)
       continue;
      j = 3;
      while (j <= i/2 && i % j != 0)
        j += 2;
      if (j > i/2) A[primecount++] = i;
    }
    for (i = 0; i<primecount; i++)//输出质数
    { cout<<setw(5)<<A[i];
      if ((i+1) % 10 == 0) //每输出10个数换行一次
        cout << endl;
    }
    cout<<endl;
}
```

# 顺序容器——双端队列

容器

- 双端队列是一种放松了访问权限的队列。元素可以从队列的两端入队和出队，也支持通过下标操作符“[]”进行直接访问。
- 例10-2
  - 使用双端队列容器保存双精度数值序列



# 顺序容器——列表

容器

- 列表主要用于存放双向链表，可以从任意一端开始遍历。列表还提供了拼接（splicing）操作，将一个序列中的元素从插入到另一个序列中。
- 例10-3 改写例9-7
  - 从键盘输入10个整数，用这些整数值作为结点数据，生成一个链表，按顺序输出链表中结点的数值。然后从键盘输入一个待查找整数，在链表中查找该整数，若找到则删除该整数所在的结点（如果出现多次，全部删除），然后输出删除结点以后的链表。在程序结束之前清空链表。

```
//10_3.cpp
#include <iostream>
#include <list>
using namespace std ;

int main()
{
 list<int> Link;    //构造一个列表用于存放整数链表
 int i, key, item;
 for(i=0;i < 10;i++)// 输入10个整数依次向表头插入
 {
   cin>>item;
   Link.push_front(item);
 }
 cout<<"List: "; // 输出链表
```

```
 list<int>::iterator p=Link.begin();
 while(p!=Link.end())//输出各节点数据，直到链表尾
 { cout <<*p << "  ";
   p++;  //使P指向下一个节点
 }
 cout << endl;
 cout << "请输入一个需要删除的整数: ";
 cin >> key;
 Link.remove(key);
 cout << "List: "; // 输出链表
 p=Link.begin();  // 使P重新指向表头
 while(p!=Link.end())
 { cout <<*p << "  ";
   p++; // 使P指向下一个节点
 }
 cout << endl;
}
```

# 容器适配器

容器

- 容器适配器是用来扩展7种基本容器的
- 栈容器
  - 使用适配器与一种基础容器相结合来实现
  - 例10-4：应用标准库中的deque顺序容器生成一个整数栈stack。
- 队列容器
  - 使用适配器与一种基础容器相结合来实现的先进先出数据结构。
  - 例10-5：应用标准库中的deque顺序容器生成一个整数标准队列queue。

# 什么是迭代器

迭代器

- 迭代器是面向对象版本的指针
  - 指针可以指向内存中的一个地址
  - 迭代器可以指向容器中的一个位置
- STL的每一个容器类模版中，都定义了一组对应的迭代器类。使用迭代器，算法函数可以访问容器中指定位置的元素，而无需关心元素的具体类型。

# 迭代器的类型

迭代器

- 输入迭代器
  - 可以用来从序列中读取数据
- 输出迭代器
  - 允许向序列中写入数据
- 前向迭代器
  - 既是输入迭代器又是输出迭代器，并且可以对序列进行单向的遍历
- 双向迭代器
  - 与前向迭代器相似，但是在两个方向上都可以对数据遍历
- 随机访问迭代器
  - 也是双向迭代器，但能够在序列中的任意两个位置之间进行跳转。

# 迭代器适配器

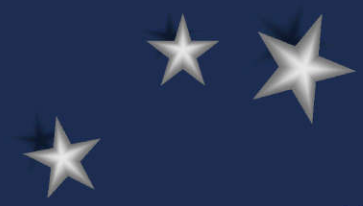
迭代器

- 迭代器适配器是用来扩展（或调整）迭代器功能的类。它本身也被称为迭代器，只是这种迭代器是通过改变另一个迭代器而得到的
- 逆向迭代器
  - 通过重新定义递增运算和递减运算，使其行为正好倒置
- 插入型迭代器
  - 将赋值操作转换为插入操作。通过这种迭代器，算法可以执行插入行为而不是覆盖行为
- 例10-6
  - 应用逆向迭代器和后插迭代器来操作向量容器中的元素

# 迭代器相关的辅助函数

迭代器

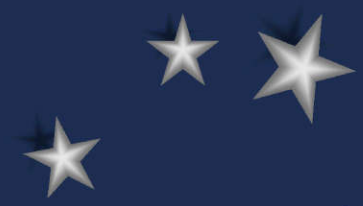
- advance()函数
  - 将迭代器的位置增加，增加的幅度由参数决定
- distance()函数
  - 返回迭代器之间的距离
- 函数iter_swap()
  - 交换两个迭代器所指向的元素值
- 例10-7
  - 用三个迭代器辅助函数来操作列表容器中的元素。



# 标准C++库中的算法

算法

- 算法本身是一种函数模板
- 不可变序列算法（non-mutating algorithms）
  - 不直接修改所操作的容器内容的算法
- 可变序列算法（mutating algorithms）
  - 可以修改它们所操作的容器的元素。
- 排序相关算法
- 数值算法



# 算法应用举例

算法

- 例10-9
  - 应用不可变序列算法对数据序列进行分析
- 例10-10
  - 以可变序列算法对数据序列进行复制，生成，删除，替换，倒序，旋转等可变性操作。
- 例10-11
  - 应用排序相关算法对序列进行各项操作
- 例10-12
  - 应用数值算法对数据序列进行操作

# 函数对象

- 一个行为类似函数的对象，它可以没有参数，也可以带有若干参数，其功能是获取一个值，或者改变操作的状态。
- 任何普通的函数和任何重载了调用运算符operator()的类的对象都满足函数对象的特征
- STL中也定义了一些标准的函数对象，如果以功能划分，可以分为算术运算、关系运算、逻辑运算三大类。为了调用这些标准函数对象，需要包含头文件<functional>。

# 小结与复习建议

- 主要内容
  - 泛型程序设计、与标准模板库有关的概念和术语、C++标准模板库中的容器、迭代器、标准C++库中的算法、函数对象
- 达到的目标
  - 初步了解泛型程序设计的概念，学会C++标准模板库（STL）的使用方法
- 实验任务
  - 实验十