STL(标准模板库)

一. STL编程

1. 何谓 STL?

   STL(standard template library) 标准模板库：

   STL 是C++标准库的一部分，STL 中提供了众多在计算机应用领域使用频繁的

   一些功能算法，这些功能是以类模板和函数模板的形成提供；

   学习网站：

   1. www.cplusplus.com
   2. www.cppreference.com

2. STL 六大关键组件

   2.1 分配器（alloctor）

   2.2 容器 （container） ***

   2.3 迭代器（iterator） ***

   2.4 算法 （algorithm） ***

   2.5 仿函数（functor）

   2.6 适配器（adaptor）

3. 容器：

简单理解：数据结构的实现(类模板)

分类：

3.1 顺序容器：

vector ***

list ***

deque

3.2 关联容器：

set

multiset

map ***

multimap

3.3 容器适配器：

stack

queue

prority_queue

4. 顺序容器：

4.1 vector(矢量/向量)

4.2 list(双向链表)

5. 关联容器：

5.1 map(映射)

6. 迭代器(iterator)：

作用： 用于访问容器中的元素；

特点： 迭代器，类似于一个指针,用于指向容器中某个位置上的元素；

换句话说，迭代器使用时，可以像指针一样使用：

本质： 是 iterator 类对象

不同的容器需要不同的迭代器

使用： 由于不同的容器需要不同的迭代器，所以我们使用迭代器时，需要为不同的容器实例化iterator迭代器对象；

格式：容器<类型实参>:: iterator it;

例子： vector<int>:: iterator it;

引出问题： 如何让迭代器指向容器中的某位置上的元素?

容器<类型实参>:: begin();

容器<类型实参>:: end();

容器<类型实参>:: rbegin();

容器<类型实参>:: rend();

分类： 反向迭代器 reverse_iterator

只读迭代器 const_iterator

随机迭代器 iterator

7. 谓词(Predicate)

返回值为bool 型的普通函数或者是返回值为bool 重载了函数调用运算符()成员函数。

一元谓词(Unary predicate)： 参数只有一个的谓词

二元谓词(Binary predicate)： 参数只有两个的谓词

```
bool  SmallerThanFive(int a)
{
    return  a < 5;
}

class Cond
{
 public:
     bool operator()(int a)
     {
         return  a < 5;
     }
};
```

8. 仿函数(functor 函数对象)

实现了函数调用运算符() 的类的对象，成为仿函数或者函数对象，

该对象行为类似于函数，

```
class Cond
{
 public:
     bool operator()(int a)
     {
```

```
        return  a < 5;
    }
};
Cond  cond;
```

C++ STL库中内置的仿函数(函数对象)：

被声明在 <functional> 头文件中

例子：

```
template <class T> struct plus : binary_function <T,T,T>
{
    T operator() (const T& x, const T& y) const {return x+y;}
};
```

8. 适配器(adaptor)

作用：　　将一个接口转换为所期待的另一个接口；

例子：　　可以将一个二元函数转换为一个一元函数

　　内部实现方法：　　参数绑定

　　怎么实现参数绑定：　bind1st /　bind2st

常用适配器分类：

取反适配器：　　not1 / not2

函数对象适配器：　　bind1st / bind2st

函数指针适配器：　　ptr_fun

成员函数适配器：　　mem_fun / mem_fun_ref

容器适配器：　　stack ，queue prority_queue;

9. 分配器 (alloctor)

作用：C++提供高级的内存管理方式

思想：将内存分配与对象构造分离的；

STL ：template <class T> class allocator;

分配器使用步骤：

1. 实例化 alloctor 的对象 alloc；
2. 内存分配， pointer * p = alloc.allocate(num);
3. 在申请的内存空间上进行数据(类对象)构造

   alloc.construct (p, val);
4. 对象析构

   alloc.destroy(p);
5. 回收内存：

   alloc.deallocate(p，num);

不重要 适配器

函数对象适配器 C++

```cpp
#include<iostream>
#include<vector>
#include<algorithm>
#include<functional>
using namespace std;

void show(int val)
{
    cout << val << '\t';
}
int main()
{
    int a[] = { 1,5,9,13,6,34 };
    vector<int> vt(a, a + sizeof(a) / sizeof(a[0]));
    for_each(vt.begin(), vt.end(), ptr_fun(show));
    cout << endl;

    /*函数对象适配器：实现原理：参数绑定：绑定一个参数，使得二元谓词成为一元谓词*/
    vector<int>::iterator fit = remove_if(vt.begin(), vt.end(), bind2nd(greater<int>(),5));
    for_each(vt.begin(), fit, show);
    cout << endl;
}
```

类成员适配器 C++

```cpp
#include<iostream>
#include<vector>
#include<list>
#include<algorithm>
#include<functional>
using namespace std;

class Simple
{
    int x;
public:
    Simple(int a):x(a){}
    void Print()
    {
        cout << x << endl;
    }
};
int main()
{
    list<Simple*> l;
    l.push_back(new Simple(1));
    l.push_back(new Simple(3));
    l.push_back(new Simple(5));
    l.push_back(new Simple(7));

    /*转化成员函数为函数对象*/
    for_each(l.begin(), l.end(), mem_fun(&Simple::Print));

    return 0;

}
```

容器适配器 C++

```cpp
#include<vector>
#include<stack>
#include<iostream>
using namespace std;

int main()
{
    int a[] = { 4,7,8,3,2,1 };
    vector<int> vt(a, a + sizeof(a) / sizeof(a[0]));
    stack<int, vector<int>> stk(vt);
    while (!stk.empty())
    {
        cout << stk.top() << '\t';
        stk.pop();
    }
    cout << endl;
    return 0;
}
```

不重要 分配器

分配基础数据类型 C++

```cpp

#include<iostream>
#include<memory>
using namespace std;

int main()
{
    allocator<int> alloc;
    int* p = alloc.allocate(3);

    alloc.construct(p, 100);
    alloc.construct(p+1, 200);
    alloc.construct(p+2, 300);

    alloc.destroy(p);
    alloc.construct(p, 888);
    for (int i = 0; i < 3; i++)
    {
        cout << p[i] << '\t';
    }
    cout << endl;

    alloc.deallocate(p, 3);


    return 0;
}
```