# 第十二章　标准库

## 本章导读

设计和实现高效的数据结构是计算机科学的重要课题之一。各种数据结构(如列表、堆栈、队列、集合以及二叉树等)在编译器构造、计算机操作系统以及文件管理等领域都有广泛的应用。在C++中，数据结构还和迭代器紧密相关，理解数据结构有助于理解迭代器的本质，从而更好地理解算法、函数对象等，也有助于更好地设计数据结构。

本章讲述几种常用数据结构的C++实现。

## 学习目标：

1. 认识迭代器及其分类；
2. 认识数组类；
3. 认识链表类；
4. 认识容器适配器；

## 本章目录



## 第一节　常用库大全

C++标准库的所有头文件都没有扩展名。C++标准库的内容总共在50个标准头文件中定义，其中18个提供了C库的功能。 <cname>形式的标准头文件【 <complex>例外】其内容与ISO标准C包含的name.h头文件相同，但容纳了C++扩展的功能。在 <cname>形式标准的头文件中，与宏相关的名称在全局作用域中定义，其他名称在std命名空间中声明。在C++中还可以使用name.h形式的标准C库头文件名。

C++标准库的内容分为10类：

- C1.语言支持
- C2.输入/输出
- C3.诊断
- C4.一般工具
- C5.字符串
- C6.容器
- C7.迭代器支持
- C8.算法
- C9.数值操作
- C10.本地化

表12-1：C++标准库文件

| 库文件名称 | 功能描述 |
| --- | --- |
| C1 标准库中与语言支持功能相关的库文件 | |
| <cstddef> | 定义宏NULL和offsetof，以及其他标准类型size_t和ptrdiff_t。与对应的标准C头文件的区别是，NULL是C++空指针常量的补充定义，宏offsetof接受结构或者联合类型参数，只要他们没有成员指针类型的非静态成员即可。 |
| <limits> | 提供与基本数据类型相关的定义。例如，对于每个数值数据类型，它定义了可以表示出来的最大值和最小值以及二进制数字的位数。 |
| <climits> | 提供与基本整数数据类型相关的C样式定义。这些信息的C++样式定义在 <limits>中 |
| <cfloat> | 提供与基本浮点型数据类型相关的C样式定义。这些信息的C++样式定义在 <limits>中 |
| | 提供支持程序启动和终止的宏和函数。这个头文件还声明了许多其他杂项函数，例如搜索和排序函数，从字符串转换 |

| | |
|---|---|
| <cstdlib> | 为数值等函数。它与对应的标准C头文件 stdlib.h不同，定义了abort(void)。abort()函数还有额外的功能，它不为静态或自动对象调用析构函数，也不调用传给 atexit()函数的函数。它还定义了exit()函数的额外功能，可以释放静态对象，以注册的逆序调用用atexit()注册的函数。清除并关闭所有 打开的C流，把控制权返回给主机环境。 |
| <new> | 支持动态内存分配 |
| <typeinfo> | 支持变量在运行期间的类型标识 |
| <exception> | 支持异常处理，这是处理程序中可能发生的错误的一种方式 |
| <cstdarg> | 支持接受数量可变的参数的函数。即在调用函数时，可以给函数传送数量不等的数据项。它定义了宏va_arg、va_end、va_start以及va_list类型 |
| <csetjmp> | 为C样式的非本地跳跃提供函数。这些函数在C++中不常用 |
| <csignal> | 为中断处理提供C样式支持 |
| C2 支持流输入/输出的库文件 | |
| < iostream> | 支持标准流cin、cout、cerr和clog的输入和输出，它还支持多字节字符标准流wcin、wcout、wcerr和wclog。 |
| <iomanip> | 提供操纵程序，允许改变流的状态，从而改变输出的格式。 |
| <ios> | 定义iostream的基类 |
| <istream> | 为管理输出流缓存区的输入定义模板类 |
| <ostream> | 为管理输出流缓存区的输出定义模板类 |
| <sstream> | 支持字符串的流输入输出 |
| <fstream> | 支持文件的流输入输出 |
| <iosfwd> | 为输入输出对象提供向前的声明 |
| <streambuf> | 支持流输入和输出的缓存 |
| <cstdio> | 为标准流提供C样式的输入和输出 |
| <cwchar> | 支持多字节字符的C样式输入输出 |
| C3 与诊断功能相关的头文件库文件 | |
| <stdexcept> | 定义标准异常。异常是处理错误的方式 |
| <cassert> | 定义断言宏，用于检查运行期间的情形 |
| <cerrno> | 支持C样式的错误信息 |
| C4 定义工具函数的库文件 | |
| <utility> | 定义重载的关系运算符，简化关系运算符的写入，它还定义了pair类型，该类型是一种模板类型，可以存储一对值。这些功能在库的其他地方使用 |
| <functional> | 定义了许多函数对象类型和支持函数对象的功能，函数对象是支持operator()()函数调用运算符的任意对象 |
| <memory> | 给容器、管理内存的函数和auto_ptr模板类定义标准内存分配器 |
| <ctime> | 支持系统时钟函数 |
| C5 支持字符串处理的库文件 | |
| <string> | 为字符串类型提供支持和定义，包括单字节字符串(由char组成)的string和多字节字符串(由wchar_t组成) |
| <cctype> | 单字节字符类别 |
| <cwctype> | 多字节字符类别 |
| <cstring> | 为处理非空字节序列和内存块提供函数。这不同于对应的标准C库头文件，几个C样式字符串的一般C库函数被返回值为const和非const的函数对替代了 |
| <cwchar> | 为处理、执行I/O和转换多字节字符序列提供函数，这不同于对应的标准C库头文件，几个多字节C样式字符串操作的一般C库函数被返回值为const和非const的函数对替代了。 |
| <cstdlib> | 为把单字节字符串转换为数值、在多字节字符和多字节字符串之间转换提供函数 |
| C6 定义容器类的模板的库文件 | |

| | |
|---|---|
| <vector> | 定义vector序列模板，这是一个大小可以重新设置的数组类型，比普通数组更安全、更灵活 |
| <list> | 定义list序列模板，这是一个序列的链表，常常在任意位置插入和删除元素 |
| <deque> | 定义deque序列模板，支持在开始和结尾的高效插入和删除操作 |
| <queue> | 为队列(先进先出)数据结构定义序列适配器queue和priority_queue |
| <stack> | 为堆栈(后进先出)数据结构定义序列适配器stack |
| <map> | map是一个关联容器类型，允许根据键值是唯一的，且按照升序存储。multimap类似于map，但键不是唯一的。 |
| <set> | set是一个关联容器类型，用于以升序方式存储唯一值。multiset类似于set，但是值不必是唯一的。 |
| <bitset> | 为固定长度的位序列定义bitset模板，它可以看作固定长度的紧凑型bool数组 |
| C7 支持迭代器的库文件 | |
| <iterator> | 给迭代器提供定义和支持 |
| C8 有关算法的库文件 | |
| <algorithm> | 提供一组基于算法的函数，包括置换、排序、合并和搜索 |
| <cstdlib> | 声明C标准库函数bsearch()和qsort()，进行搜索和排序 |
| <ciso646> | 允许在代码中使用and代替&& |
| C9 有关数值操作的库文件 | |
| <complex> | 支持复杂数值的定义和操作 |
| <valarray> | 支持数值矢量的操作 |
| <numeric> | 在数值序列上定义一组一般数学操作，例如accumulate和inner_product |
| <cmath> | 这是C数学库，其中还附加了重载函数，以支持C++约定 |
| <cstdlib> | 提供的函数可以提取整数的绝对值，对整数进行取余数操作 |
| C10 有关本地化的库文件 | |
| <locale> | 提供的本地化包括字符类别、排序序列以及货币和日期表示。 |
| <clocale> | 对本地化提供C样式支持 |

## 第二节　正则表达式

例程12-1

```
第1行  #include<iostream>
第2行  #include<string>
第3行  #include<regex>
第4行  using namespace std;
第5行
第6行  int main(){
第7行          regex regA("<.*>.*</.*>");
第8行          bool found = regex_match(string("<em>Tsinghua</em><div>Pku</div>"), regA);
第9行          cout << found << endl;
第10行
第11行         string data = "XML tag: <div>北京大学</div><em>清华大学</em>双子星.";
第12行         cout << data << endl;
第13行
第14行         auto beg = data.cbegin();
第15行         auto end = data.cend();
```

```
第16行
第17行          smatch m;
第18行          for (; regex_search(beg, end, m, regex("<(.*)>([^<]*)</(\\1)>")); beg = m.suffix().first){
第19行                  cout << m.str() << endl;
第20行                  cout << m.str(1) << endl;
第21行                  cout << m.str(2) << endl;
第22行                  cout << endl << endl;
第23行          }
第24行
第25行
第26行          system("pause");
第27行          return 0;
第28行  }
```

## 第三节　array容器

例程12-2

```
第1行   #include<iostream>
第2行   #include<array>
第3行   using namespace std;
第4行
第5行   int main(){
第6行           array<int, 10> arrInt = { 1, 3, 4, 6, 4, 12, 14, 51, 31, 21 };
第7行           for (auto i : arrInt)
第8行                   cout << i <<" ";//输出:1 3 4 6 4 12 14 51 31 21
第9行
第10行          cout << arrInt.size() << endl;//输出：10
第11行          arrInt.fill(10);
第12行          for (auto i : arrInt)cout << i << " ";//输出10个10
第13行          int N = 0;
第14行          for (auto iter = arrInt.begin(); iter != arrInt.end(); ++iter)
第15行                  *iter = N++;
第16行
第17行          for (int i = 0; i < 10; i++)
第18行                  cout << arrInt[i] << " ";//输出：0 1 2 3 4 5 6 7 8 9
第19行
第20行          arrInt.assign(10);//每个成员的值改变为10
第21行          for (auto i : arrInt)cout << i << " ";//输出10个10
第22行
第23行          arrInt.at(1) = 99;//编号为1即第2个的元素值为99
第24行          cout <<"\n最后一个元素"<< arrInt.back() <<"第一个元素"<< arrInt.front() << endl;
第25行
第26行          const int *p = arrInt.data();//转换为指针
第27行          for (int i = 0; i < 10; i++)
第28行                  cout << *(p + i) << " ";
```

```
第29行
第30行          cout << arrInt.empty() << endl;//输出：0
第31行
第32行          return 0;
第33行  }
```

## 第四节　forward_list容器

## 第五节　bitset容器

## 第六节　无序容器

## 第七节　tuple

例程12-3

```
第1行  #include <iostream>
第2行  #include <tuple>
第3行  #include <functional>
第4行  int main(){
第5行          auto t1 = std::make_tuple(10, "Test", 3.14);
第6行          std::cout << "The value of t1 is "
第7行                  << "(" << std::get<0>(t1) << ", " << std::get<1>(t1)
第8行                  << ", " << std::get<2>(t1) << ")\n";//The value of t1 is (10, Test, 3.14)
第9行
第10行         int n = 1;
第11行         auto t2 = std::make_tuple(std::ref(n), n);//ref表示引用
第12行         n = 7;
第13行         std::cout << "The value of t2 is "<< "(" << std::get<0>(t2)
第14行                 << ", " << std::get<1>(t2) << ")\n";//        The value of t2 is(7, 1)
第15行
第16行         return 0;
第17行 }
```

例程12-4

```
第1行  #include <iostream>
第2行  #include <tuple>
第3行  using namespace std;
第4行
第5行  int main(){
第6行          int myint;
第7行          char mychar;
第8行          double myfloat;
第9行          tuple<int, double, char> myTuple;
第10行         myTuple = make_tuple(10, 2.6, 'a'); // packing values into tuple
第11行         get<0>(myTuple) = 99;
第12行         cout << get<0>(myTuple) << get<1>(myTuple) << get<2>(myTuple) << endl;
第13行         //tie (myint, ignore, mychar) = myTuple; // unpacking tuple into variables 【1】
第14行         tie(myint, myfloat, mychar) = myTuple;
```

```
第15行          cout << "myint contains: " << myint << endl;
第16行          cout << "mychar contains: " << mychar << endl;
第17行          cout << "myfloat contains: " << myfloat << endl;
第18行          get<0>(myTuple) = 100;//修改tuple的值
第19行          cout << "After assignment myint contains: " << get<0>(myTuple) << endl;
第20行          cout << myint << endl;
第21行
第22行          return 0;
第23行  }
```

例程12-5

```
第1行   #include <iostream>
第2行   #include <string>
第3行   #include <tuple>
第4行   using namespace std;
第5行
第6行   int main(){
第7行           tuple<double, string,string> tupleA(3.14, "pi","ABC");
第8行           tuple<int, char> tupleB(10, 'a');
第9行           auto myauto = tuple_cat(tupleA, tupleB);
第10行          cout << "myauto contains: " << endl;
第11行          cout << get<0>(myauto) << endl;
第12行          cout << get<1>(myauto) << endl;
第13行          cout << get<2>(myauto) << endl;
第14行          cout << get<3>(myauto) << endl;
第15行          cout << get<4>(myauto) << endl;
第16行
第17行          return 0;
第18行  }
```

例程12-6

```
第1行   #include <iostream>
第2行   using namespace std;
第3行   template<typename ...Args>
第4行   int getArgsNum(Args ...args){
第5行           return sizeof...(args);
第6行   }
第7行   int main(){
第8行           cout << getArgsNum(123, "ABC", 12.12);//输出：3
第9行
第10行          return 0;
第11行  }
```

例程12-7

```
第1行   #include <iostream>
第2行   using namespace std;
```

```
template<typename T>
void Print(T Vals){
        cout << Vals << endl;
}
template<typename Head,typename ...Tail>
void Print(Head h,Tail ... t){
        cout << h<<" ";
        Print(t...);
}
int main(){
        Print(1);
        Print(1,"XYZ");
        Print(1,"XYZ",12.12);

        return 0;
}
```

例程12-8

```
#include<iostream>
#include<cstdarg>
using namespace std;

int add_nums(int count, ...){
        int result = count;
        va_list args;
        va_start(args, count);
        for (int i = 0; i<count; ++i)
                result += va_arg(args, int);

        va_end(args);

        return result;
}

int main()
{
        cout << add_nums(4, 25, 25, 50, 50) << endl;

        return 0;
}
```