10-1

容器：可逆容器、随机访问容器

顺序容器：向量、双端队列、列表

关联容器：集合、多重集合、映射、多重映射

迭代器：

输入迭代器、输出迭代器：前向迭代器

单向迭代器：双向迭代器：随机访问迭代器

算法：查找、排序、消除、计数、比较、变换、置换、容器管理

vector：随机访问迭代器

deque：随机访问迭代器

list：双向迭代器

set、multiset、map、multimap：双向迭代器

10-2

[s+1,s+4) s[1],s[2],s[3]

[s+4,s+5) 合法

[s+4,s+4), [s+4,s+3) 不合法

10-3

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
  vector<int>s;
  while(s.size()<100){
    s.push_back(1);
    cout<<s.capacity()<<" ";
  }
}
```

capacity：1 2 4 8 16 32 64 128 ……

10-4

（1）vector （2）deque （3）list

11-1

流是一种抽象，负责在数据的生产者和数据的消费者之间建立联系，并管理数据的流动

流的提取：读操作

流的插入：写操作

I/O流在c++中实现数据的流动，进行信息交换，同时编译时对类型严格检查，更安全

11-2

cout：标准输出流

cerr：标准错误输出流，没有缓冲，发送给它的内容立即被输出

clog：类似于cerr，但是有缓冲，缓冲区满时被输出

用 > 对标准输出重定向

2> 对标准错误输出重定向

12-1

异常：函数之间有明确的分工和复杂的调用关系，发现错误的函数往往不具备处理错误的能力，就会引发异常

异常处理：设计程序时，要充分考虑到各种意外情况，并给予恰当的处理

12-2

c++的异常处理机制使异常的引发和处理不必在同一函数中，这样底层的函数可以着重解决具体问题，不必过多地考虑对异常的处理，而上层调用者可以在适当的位置设计对不同异常的处理

12-3

```
#include <iostream>
using namespace std;
int divide(int x, int y) {
    if (y == 0)
        throw x;
    return x / y;
}
int main() {
    try {
        cout << "5 / 2 = " << divide(5, 2) << endl;
        cout << "8 / 0 = " << divide(8, 0) << endl;
        cout << "7 / 1 = " << divide(7, 1) << endl;
    } catch (int e) {
        cout << e << " is divided by zero!" << endl;
    }
    cout << "That is ok." << endl;
    return 0;
}
```

程序通过正常的顺序执行到达try语句，然后执行try块内的保护段。

如果在保护段执行期间没有引起异常，那么跟在try块后的catch子句就不执行。程序从try块后跟随的最后一个catch子句后面的语句继续执行下去。

若有异常则通过throw操作创建一个异常对象并抛掷。将可能抛出异常的程序段嵌在try块之中。catch子句按其在try块后出现的顺序被检查。匹配的catch子句将捕获并处理异常（或继续抛掷异常）。

如果匹配的处理器未找到，则运行库函数terminate将被自动调用，默认功能为终止程序。

如果找到匹配的catch，执行子句中的语句后，try和catch块即执行完毕。

如上述例子，在divide(8, 0) 处发现除零异常，被catch（int e）捕捉，执行其语句，输出" is divided by zero!"，该try-catch语句块执行完毕，不会继续执行cout << "7 / 1 = " << divide(7, 1) << endl;语句。