Labb 6 Algoritmer

Uppgift: Använd STL (Standard Template Library) för att implementera några algoritmer genom att använda de färdiga algoritmerna och containerna i STL.

Genomgående så använder vi klassen C som element att sortera den är definierad som:

```
template <int N> struct C {
    int value;   //det är denna som används
    int a[N];    //bara en placeholder för att det ska ta tid att
kopiera ett C objekt.
}
```

Observera att man med iterators inte kan ta bort ett element i en container utan att t.ex. ”remove” sorterar om containern så att det som ska bort kommer sist.

Uppgift 1:

Ett problem med att algoritm-biblioteket använder sig av iteratorer och inte själva containers är att en iterator inte kan ta bort element i en container. Gör ett program som:

1. Skapar en container med slumpvisa tal (C-objekt) eller bestämda tal med slumpvis ordning. (random_shuffle)
2. Skriver ut containern.
3. Tar bort alla jämna tal med hjälp av STL (använd ”remove_if” och ”erase”).
4. Skriver ut containern.

Uppgift 2A:

Visa att det räcker med forward iteratorer för att t.ex. Sortera en container. Gör en function:

```
template <class ForwardIterator>
void ForwardSort(ForwardIterator begin, ForwardIterator end);
```

Den ska bara använda forward iterator funktionerna (dvs. man kan göra *it, ++it, it1!=it2 och inte så mycket mera) och sortera containerna den används för.

Uppgift 2B:

Använd ForwardSort för att sortera en container med C-objekt.

Uppgift 3a:

Gör ett program som:

1. Skapar en container med slumpvisa tal (C-objekt) eller bestämda tal med slumpvis ordning.
2. Skriver ut containern.
3. Sorterar den baklänges mha. reverse_iterator.
4. Skriver ut containern.

Uppgift 3b:

Gör ett program som:

1. Skapar en container med slumpvisa tal (C-objekt) eller bestämda tal med slumpvis ordning.
2. Skriver ut containern.
3. Sorterar den baklänges genom att skicka med en lambda funktion som parameter.
4. Skriver ut containern.

Uppgift 4:

Gör ett program som:

1. Skapar en container med pekare till slumpvisa heltal (C-objekt) eller bestämda heltal med slumpvis ordning.
2. Skriver ut containern – det är C objekten som ska skrivas ut.
3. Sorterar den med std::sort genom att skicka med en lambda funktion som parameter – det är C objekten som ska sorteras.
4. Skriver ut containern – det är C objekten som ska skrivas ut.

Uppgift 5:

Testa hastigheten på era String objekt jämfört med std::string. Använd HiResTimer.h och .cpp för att mäta tiden. Nedan finns exempel på program som ni kan använda. Ni ska skapa ett lämpligt antal strängar (10000 t.ex.) av lämplig längd (0-100 t.ex.) genom att göra upprepade push_back på era strängar.
Använd `GenerateStrings` (se sist) för att mäta tiden, gör det för fyra kombinationer och skriv ner resultatet. De fyra kombinationerna kombinationerna av följande två varianter:

1. Er String klass.
2. std:string

resp.

a. I debug configuration och ”Start Without Debugging”
b. I release configuration och ”Start Without Debugging”

De senare styr ni med Build/Configuration Manager och välj överst mellan debug och release.

Försök förklara resultatet som blir lite olika beroende på hur ni implementerat er String klass.

Några Tips:

Random numbers: C++ bibliotek är krångligt, det gamla C varianten enkel:

```
#include <cstdlib>
int r = rand();
```

och nu är r ett random positivt heltal (0 <= r <= RAND_MAX == 32767)

Fylla en container med något:

```
    iota(v.begin(), v.end(), 101);
```

fyller v med 101, 102, 103 etc.

```
random_shuffle(v.begin(), v.end());
```

"blandar" v

```
for (int i = 0; i < N; ++i){
    list.push_front(C(rand()));
}
```

Lägger till N random tal i "list"

Lambdafunktioner är anonyma funktioner t.ex.:

```
[](int x){return i%10==7; }
```
//ger true för alla tal som har 7 som sista siffra.

Generate Strings:

```
#include "HiResTimer.h"

#include "StringItt.h"  //Include String without iterators

char GenRandLetter() {
    return ('a' + (rand()*('z' - 'a' + 1) / RAND_MAX));
}

const int NofStrings = 10000;
const int MaxLengthofString = 100;

template<class str>
double GenerateStrings(vector<str>& VectorOfString) {
    HiResTimer timer;
    VectorOfString.resize(0);
    VectorOfString.shrink_to_fit();
    VectorOfString.resize(NofStrings);
    timer.Start();
    for (unsigned i = 0; i < VectorOfString.size(); ++i) {
        int length = MaxLengthofString* rand() / RAND_MAX;
        for (int j = 0; j < length; ++j)
            VectorOfString[i].push_back(GenRandLetter());
    }
    timer.Stop();
    return timer.GetDeltaTime();
}

void Test5() {
    vector<string> mystrings;
    vector<String> myStrings;
    auto timeS = GenerateStrings(myStrings);
    auto times = GenerateStrings(mystrings);
    cout << string("time with my String   ") << timeS << endl;
    cout << string("time with std::string ") << times << endl;
    cin.get();
}
```