SCHETT MATTHIAS

SEN-ÜBUNG 08

Inhaltsverzeichnis

Aufgabe 1

Lösungsidee

Der Shift Algorithmus erstellt einen Iterator der auf die count .te Stelle im Container zeigt und rotiert dann den Container mit std::rotate, anschließend werden die count letzten Stellen mit der Wert von val aufgefüllt. Der Iterator für die Mitte und die count letzte Stelle wird mittels std::advance erstellt. Der Code findet sich ab Anhang Aufgabe 1.

Testfälle

Listing 1: Ausgabe des Testreibers

```
1 Template Test with vector
2 Shift is done 3 times and filled up with 5
3 3, 4, 5, 6, 7, 8, 9, 5, 5, 5,
4 Test ende
5 Template Test with list
6 Shift is done 4 times and filled up with 6
7 4, 5, 6, 7, 8, 9, 6, 6, 6, 6,
8 Test ende
```

Aufgabe 2

Lösungsidee

Die Funktion checkSorting prüft die Sortierung in einem per Iteratoren gegebenen Bereich und gibt anschließend die vier in der Aufgabenstellung definierten Zustände zurück. Erreicht wird die Prüfung durch eine Iteration über den gegebenen Bereich und der Prüfung ob der Iterator größer oder kleiner ist als der nachfolgende, entsprechend werden Zähler erhöht. Nach der Iteration wird, falls ein Zähler gleich groß wie der gegebene Bereich ist, das Ergebnis Ascending, Descending oder EqualOrEmpty zurückgegebenen. Sollte kein Zähler das Kriterium erfüllen, dann kann davon ausgegangen werden, dass der Bereich nicht sortiert wird und daher wird UnSorted zurückgegeben. Es gibt auch eine Version mit Prädikat, bei dieser verhält es sich gleich nur statt dem Vergleich wird das Ergebnis des Prädikates, welches als bool vorliegen muss, verwendet. Der Code findet sich ab Anhang Aufgabe 2.

Testfälle

Listing 2: Ausgabe des Testreibers

```
1  Test with vector
2
3  Test with empty container: EmptyOrEqual
4  Test with ascending ordered container: Ascending
5  Test with custom predicate and ascending order container: Ascending
6  Test with random ordered container: UnSorted
7  Test with descending ordered container: Descending
8  Test with equal entries container: EmptyOrEqual
9
10
11 Test with std::list
12
13 Test with empty container: EmptyOrEqual
14 Test with ascending ordered container: Ascending
15 Test with custom predicate and ascending order container: Ascending
16 Test with random ordered container: UnSorted
17 Test with descending ordered container: Descending
18 Test with equal entries container: EmptyOrEqual
```

Aufgabe 3

Lösungsidee

Allgemein lässt sich die Laufzeikomplexität nach folgender Formel bestimmen:

$$T(n) = a * T(\frac{n}{b}) + f(n)$$

Wobei **a** für die Anzahl and rekursiven Aufrufen steht, $\frac{n}{b}$ für die Anzahl an Unterproblemen und **f(n)** für den Aufwand der Aufteilung des Problemes und zur Anschließenden Kombination der Teillösungen. Es gibt 3 Lösungsvarianten für diese Gleichung:

Fall	Wenn...	Dann..
1	$f(n) = \mathcal{O}(n^{\log_b(a-\epsilon)})$ für $\epsilon > 0$	$T(n) = \Theta(n^{\log_b(a)})$
2	$f(n) = \Theta(n^{\log_b(a)})$	$T(n) = \Theta(n^{\log_b(a)} * \log(n))$
3	$f(n) = \Omega(n^{\log_b(a-\epsilon)})$ für $\epsilon > 0$ und $a * f(\frac{n}{b}) \leq c * f(n)$ für $c < 1$	$T(n) = \Theta(f(n))$

Es wird nun für das erste und das zweite Beispiel der Lösungsweg skiziert für die restlichen Beispiele wird anschließend nur die Lösung angegeben.

Beispiel 1

Angabe: $T(n) = 3T(\frac{n}{2}) + n^2$

Daher ist $a = 3, b = 2, f(n) = n^2$ und der $\log_b(a) \approx 1.6$

$n^2 \in \Omega(n^{1.6+\epsilon})$

und $3 * (\frac{n}{2})^2 \leq cn^2$ berrechnet man nun den ersten Ausdruck erhält man

$\frac{3n^2}{4} \leq cn^2$ für c = 0.9 z.Bsp

Daher ist die Lösung vom Fall 3 $T(n) = \Theta(n^2)$

Beispiel 2

Angabe: $T(n) = 12T(\frac{n}{4}) + n$

Daher ist $a = 12, b = 4, f(n) = n$ und der $\log_b(a) \approx 1.8$

$n \in \mathcal{O}(n^{1.8-\epsilon})$

Daher ist die Lösung vom Fall 1 und $T(n) = \Theta(n^{1.8})$

Restliche Beispiele

Hier folgen nun die Lösungen der restlichen Beispiele:

3. $T(n) = \Theta(n)$ Fall 3
4. $T(n) = \Theta(n^{1.5})$ Fall 1
5. $T(n) = \Theta(n^2)$ Fall 3
6. $T(n) = \Theta(n^3 * \log(n))$ Fall 2

Anhang A
Aufgabe 1

Listing A.1: Implementierung der Shiftfunktion

```
1  ////////////////////////////////////
2  // Workfile    : TemplateShift.h
3  // Author      : Matthias Schett
4  // Date        : 19-05-2013
5  // Description : Shift function template
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #ifndef TEMPLATE_SHIFT
11 #define TEMPLATE_SHIFT
12
13 #include <algorithm>
14
15 //*********************************
16 // Method:    Shift
17 // FullName:  Shift
18 // Access:    public
19 // Returns:
20 // Qualifier:
21 // Parameter: TItor begin
22 // Parameter: TItor end
23 // Parameter: size_t count
24 // Parameter: TValue val
25 // Shift elements in container count places to the front
26 //*********************************
27 template <typename TItor, typename TValue>
28 void Shift(TItor begin, TItor end, size_t count, TValue val){
29
30     TItor middle = begin;
31
32     std::advance(middle, count);
33
34     std::rotate(begin, middle , end);
35
```

```
36      TItor it = end;
37      int countInt = 0 - count;
38      std::advance(it, countInt);
39      while(it != end){
40          *it = val;
41          std::advance(it, 1);
42      }
43
44  }
45
46
47  #endif
```

Listing A.2: Testtreiber

```
1   ////////////////////////////////////
2   // Workfile    : Main.cpp
3   // Author      : Matthias Schett
4   // Date        : 19-05-2013
5   // Description : Shift function template
6   // Remarks     : -
7   // Revision    : 0
8   ////////////////////////////////////
9
10  #include <iostream>
11  #include <vector>
12  #include <list>
13  #include <algorithm>
14  #include <iterator>
15  #include <fstream>
16  #include "TemplateShift.h"
17
18  using namespace std;
19
20  int main(){
21
22      ofstream oFile("OutputA1.txt");
23
24      std::ostream_iterator<int> out_it (oFile, ", ");
25
26      oFile << "Template Test with vector" << endl;
27
28      vector<int> myVec;
29
30      for (int i=0; i<10; i++) {
31          myVec.push_back (i);
32      }
33
34      size_t count = 3;
35      int val = 5;
```

```
36      oFile << "Shift is done " << count << " times and filled up with " << val << endl;
37      Shift(myVec.begin(), myVec.end(), count, val);
38
39      std::copy ( myVec.begin(), myVec.end(), out_it );
40
41      oFile << endl << "Test ende" << endl;
42
43      oFile << "Template Test with list" << endl;
44
45      list<int> myList;
46
47      for (int i=0; i<10; i++) {
48          myList.push_back (i);
49      }
50
51      count = 4;
52      val = 6;
53      oFile << "Shift is done " << count << " times and filled up with " << val << endl;
54      Shift(myList.begin(), myList.end(), count, val);
55
56      std::copy ( myList.begin(), myList.end(), out_it );
57
58      oFile << endl << "Test ende" << endl;
59
60      cin.get();
61      return 0;
62  }
```

Anhang B
Aufgabe 2

Listing B.1: Implementierung der Sortierprüfung

```
1  ////////////////////////////////////
2  // Workfile    : GenericSortCheck.h
3  // Author      : Matthias Schett
4  // Date        : 19-05-2013
5  // Description : Generic Sort Checking
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include <algorithm>
11 #include <iterator>
12 #include <string>
13
14 enum SortType{
15
16     Ascending,
17     Descending,
18     UnSorted,
19     EmptyOrEqual,
20
21 };
22
23 //*********************************
24 // Method:    getDayName
25 // FullName:  getDayName
26 // Access:    public
27 // Returns:   const string &
28 // Qualifier:
29 // Parameter: enum SortType sortT
30 // Prints the value of the enum
31 //*********************************
32 const char* getSortType(enum SortType sortT) {
33     switch (sortT) {
34         case Ascending: return "Ascending";
35         case Descending: return "Descending";
```

```
36          case UnSorted: return "UnSorted";
37          case EmptyOrEqual: return "EmptyOrEqual";
38      }
39  }
40
41  //**********************************
42  // Method:    CheckSorting
43  // FullName:  CheckSorting
44  // Access:    public
45  // Returns:   SortType
46  // Qualifier:
47  // Parameter: TItor begin
48  // Parameter: TItor end
49  // Checks the sorting for a container
50  //**********************************
51  template <typename TItor>
52  SortType CheckSorting(TItor begin, TItor end){
53
54      if(begin == end){
55          return EmptyOrEqual;
56      }
57
58      int distance = std::distance(begin, end) - 1;
59      int ascCount = 0;
60      int descCount = 0;
61      int eqCount = 0;
62      for(TItor it = begin; it != end; ++it){
63          TItor it2 = it;
64          std::advance(it2, 1);
65          if( (it2 != end) && (*it < *it2) ){
66              ascCount++;
67          }else if( (it2 != end) && (*it > *it2) ){
68              descCount++;
69          } else if( (it2 != end) && (*it == *it2) ){
70              eqCount++;
71          }
72      }
73
74      if(ascCount == distance){
75          return Ascending;
76      } else if(descCount == distance){
77          return Descending;
78      } else if(eqCount == distance){
79          return EmptyOrEqual;
80      } else {
81          return UnSorted;
82      }
83
84  }
```

```
85
86  template <typename TItor, typename BinaryFunction>
87  SortType CheckSorting(TItor begin, TItor end, BinaryFunction f){
88
89      if(begin == end){
90          return EmptyOrEqual;
91      }
92
93      int distance = std::distance(begin, end) - 1;
94      int ascCount = 0;
95      int descCount = 0;
96      for(TItor it = begin; it != end; ++it){
97          TItor it2 = it;
98          std::advance(it2, 1);
99          if( (it2 != end) ) {
100             bool order = f(it, it2);
101             if(order){
102                 ascCount++
103             } else{
104                 descCount++;
105             }
106         }
107     }
108
109     if(ascCount == distance){
110         return Ascending;
111     } else if(descCount == distance){
112         return Descending;
113     } else {
114         return UnSorted;
115     }
116 }
```

Listing B.2: Testtreiber

```
1  ////////////////////////////////////
2  // Workfile    : Main.cpp
3  // Author      : Matthias Schett
4  // Date        : 19-05-2013
5  // Description : Generic Sort Checking
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include <iostream>
11 #include <vector>
12 #include <algorithm>
13 #include <functional>
14 #include <list>
15 #include <fstream>
```

```cpp
#include "GenericSortCheck.h"

using namespace std;

bool testSortPredicate(const int a, const int b){
    if(a == b){
        return true;
    } else if(a < b){
        return true;
    } else if(a > b){
        return false;
    }
}

int main(){

    ofstream file("OutputA2.txt");

    SortType sortT;
    vector<int> v;

    file << "Test with vector" << endl << endl;

    file << "Test with empty container: ";
    file << getSortType(CheckSorting(v.begin(), v.end())) << endl;

    for(int i = 0; i < 10; i++){
        v.push_back(i);
    }

    file << "Test with ascending ordered container: ";
    file << getSortType(CheckSorting(v.begin(), v.end())) << endl;

    file << "Test with custom predicate and ascending order container: ";
    file << getSortType(CheckSorting(v.begin(), v.end())) << endl;

    std::random_shuffle(v.begin(), v.end());

    file << "Test with random ordered container: ";
    file << getSortType(CheckSorting(v.begin(), v.end())) << endl;

    std::sort(v.begin(), v.end(), std::greater<int>());

    file << "Test with descending ordered container: ";
    file << getSortType(CheckSorting(v.begin(), v.end())) << endl;

    v.clear();
    for(int i = 0; i < 10; i++) {
        v.push_back(1);
```

```
65      }
66
67      file << "Test with equal entries container: ";
68      file << getSortType(CheckSorting(v.begin(), v.end())) << endl;
69
70      file << endl << endl << "Test with std::list" << endl << endl;
71
72      list<int> l;
73
74      file << "Test with empty container: ";
75      file << getSortType(CheckSorting(l.begin(), l.end())) << endl;
76
77      for(int i = 0; i < 10; i++){
78          l.push_back(i);
79      }
80
81      file << "Test with ascending ordered container: ";
82      file << getSortType(CheckSorting(l.begin(), l.end())) << endl;
83
84      file << "Test with custom predicate and ascending order container: ";
85      file << getSortType(CheckSorting(l.begin(), l.end())) << endl;
86
87      l.clear();
88      for(int i = 0; i < 10; i++){
89          if(i % 2 == 0){
90              l.push_back(i);
91          } else {
92              l.push_front(i);
93          }
94
95      }
96
97      file << "Test with random ordered container: ";
98      file << getSortType(CheckSorting(l.begin(), l.end())) << endl;
99
100     l.sort(std::greater<int>());
101
102     file << "Test with descending ordered container: ";
103     file << getSortType(CheckSorting(l.begin(), l.end())) << endl;
104
105     l.clear();
106     for(int i = 0; i < 10; i++) {
107         l.push_back(1);
108     }
109
110     file << "Test with equal entries container: ";
111     file << getSortType(CheckSorting(l.begin(), l.end())) << endl;
112
113     cin.get();
```

```
114     return 0;
115 }
```