GP 2	Generative Programming	ST 17, Exercise 1

Deadline: 31.03.2017, 13:50

☐ **Gr. 1, J. Karder, MSc.**
☐ **Gr. 2, P. Fleck, MSc.**

Name ___________________________________ **Effort in h** _____

Points __________________________________ **Lecturer** ______

1. ... like Bunnies (2 + 2 Points)

Since your first programming exercises (in the long gone days of your bachelor studies) you know the famous Fibonacci sequence which is defined for all positive integers *n*:

$$fib(0) = 0$$
$$fib(1) = 1$$
$$fib(n) = fib(n-1) + fib(n-2)$$

Perhaps you also remember that the Fibonacci sequence is not only a well-known example of recursive functions, but is also of unbelievable practical use: Using the Fibonacci sequence you can compute the size of an (ideal) bunny population at time *t*.

Due to this immense importance of the Fibonacci sequence, we would like to focus on it in this first task. Anyhow, the computation of the sequence should now not be done at runtime but during compilation.

a) Write a C++ template metafunction *Fibonacci* which computes the *n*-th Fibonacci number at compile time. Demonstrate how the function can be used by computing *Fibonacci<40>*. For comparison, also implement a recursive function *fibonacci* which computes the Fibonacci numbers in the conventional way at runtime.

b) Extend your program so that it lists the first 40 Fibonacci numbers. Thereby the numbers should be computed on the one hand at compile time using template metafunctions (e.g., *DO* or *WHILE*) and on the other hand conventionally at runtime.

2. 1, 2, 3, 4, 5, 6, 7, … (6 + 6 Points)

a) Implement a class *Counter* which provides the methods *reset*, *increment* and *value* to access and manipulate its counter value. The data type and the initial value of the counter value should be configurable. Use a *traits class* to pass all required configuration information to the *Counter* (data type and initial value).

b) Furthermore, two additional optional configurations should be available for counters. On the one hand it should be possible to define an upper limit for the counter value, which must not be exceeded (*BoundedCounter*). On the other hand the step size of the counter should be configurable (*VarIncrementCounter*). Use the concept of *inheritance-based wrapper classes* to realize specializations of *Counter* for these two configurations.

   Test different configurations (e.g., *IntCounter*, *DoubleCounter*, *BoundedIntCounter*, *VarIncrementIntCounter*, *BoundedVarIncrementIntCounter*, …)

3. Counters off the Shelf (8 Points)

Similarly to the *LIST_GENERATOR* example which has been discussed in the lecture, implement your own generator *COUNTER_GENERATOR* to create suitable configurations for your *Counter* class automatically.

Thoroughly think about, how you can pass all the different values to the generator which are required in the configuration (initial value, upper bound, step size).

Test your generator extensively by creating and using different counter variants.

1 ... like Bunnies

1.1 Lösungsidee

Die Implementierungen der Aufgabe *like Buniies* wird in drei Dateien aufgeteilt.

- fibonacci.hpp
- statement.hpp
- main.cpp

Die Berechnung der Fibonacci-Folge wird über eine rekursive Funktion implementiert, die dazu verwendet wird die Berechnung zur Laufzeit zu durchzuführen. Es wird ein Template implementiert, welches die Fibonacci-Folge zur Compile-time berechnet. Es werden zwei partielle Ausprägungen des Templates implementiert und zwar für die Werte 0 und 1. 0 damit die Instanziierung der Templates ein definiertes Ende hat und zusätzlich eine partielle Ausprägung für 1, da die Fibonacci-Folge wie folgt definiert ist $fib(n) = fib(n-1) + fib(n-1)$.

Für die Berechnung der Fibonacci-Folge über DO-IF-Template-Metafunctions, die zur Compile-Time evaluiert werden, wird ein Template für die Berechnung der Fibonacci-Folge implementiert, sowie ein Template für die Condition, welche entscheidet wann die Berechnung fertig ist. Um das aktuelle n im Template zu speichern wird folgende Anweisung verwendet $enum\ \{\ current = n\ \}$. Das gespeicherte n als current wird von der *FibonacciCondition* benötigt um zu entscheiden, wann die Berechnung fertig ist.

Die DO-IF-Statement Templates werden in der Datei statement.hpp implementiert. Für das IF-Template wird zusätzlich ein partielles Template für den boolschen Wert false implementiert. Das DO-Template erwartet sich zwei Typparameter *Statement* und *Condition*, wobei *Statement* eine einfach verkettete Liste von *Statements* darstellt, wobei das nächste *Statement* über *Statement:NEXT* verfügbar ist.

Listing 1: fibonacci.hpp

```cpp
#ifndef _fibonacci_h
#define _fibonacci_h

// function which calculates fibonacci number
int fibonacci(int n) {
  if (n <= 1) {
    return n;
  }
  else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}

// partial fibonacci template
template<int n>
struct Fibonacci {
  enum { RET = Fibonacci<n - 1>::RET + Fibonacci<n - 2>::RET };
};

// full fibonacci template for 0
template<>
struct Fibonacci<0> {
  enum { RET = 0 };
};

// full fibonacci template for 1
template<>
struct Fibonacci<1> {
  enum { RET = 1 };
};

// partial fibonacci condition template
template<int n>
struct FibonacciStatement {
  // here we remember the current set template value, otherwise will be lost
  enum { current = n };
  static void exec() {
    cout << "fibonacci<" << n << ">::RET = " << Fibonacci<n>::RET << endl;
  }
  typedef FibonacciStatement<n + 1> Next;
};

// partial fibonacci end condition statement template
template<int max>
struct FibonacciEndCondition {
  template<typename Statement>
  struct Code {
    enum { RET = Statement::current <= max };
  };
};

#endif
```

Listing 2: statement.hpp

```
#ifndef _statement_
#define _statement_

struct Stop {
  static void exec() {}
};

// If template
template<bool Condition, typename Then, typename Else>
struct IF {
  typedef Then RET;
};

// Then template
template<typename Then, typename Else>
struct IF<false, Then, Else> {
  typedef Else RET;
};

// Do statement
template<typename Statement, typename Condition>
struct DO {
  typedef typename Statement::Next NextStatement;
  static void exec() {
    Statement::exec();

    IF<Condition::Code<NextStatement>::RET,
      DO<NextStatement, Condition>,
      Stop
    >::RET::exec();
  }
};
#endif
```

Listing 3: main.cpp

```cpp
#include <iostream>
#include <chrono>
#include "fibonacci.hpp"
#include "statement.hpp"

using namespace std;

// templated function for measuirng
template<typename Func>
void measure(Func func) {
  auto start = chrono::high_resolution_clock::now();
  func();
  auto end = chrono::high_resolution_clock::now();

  auto duration = chrono::duration_cast<chrono::duration<double>> (end - start);
  cout << "duration: " << fixed << duration.count() << endl << "----------------------------------" << endl;
}

int main() {
  static const int n = 40;

  // measure recursive call at runtime
  measure([&]() { cout << "Recursive call:" << endl << "fibonacci(" << n << ") = " << fibonacci(n) << endl; });
  // measure compile time calculation
  measure([&]() { cout << "Compile time call:" << "fibonacci<" << n << ">::RET = " << Fibonacci<n>::RET << endl; } );

  // iterative recursive calls
  measure([&]() {
    for (int i = 0; i < n; i++) {
      measure([&]() {cout << "Iterative call: " << endl << "fibonacci(" << i << ") = " << fibonacci(i) << endl; });
    }
  });

  // measure compile time call with Do template
  cout << "Compile time call with Do - If statement: " << endl;
  measure([&]() {
    DO<FibonacciStatement<10>, FibonacciEndCondition<40>>::exec();
  });

  return 0;
}
```

```
Recursive call:
fibonacci(40) = 102334155
duration: 8.128902
----------------------------------
Compile time call:fibonacci<40>::RET = 102334155
duration: 0.001241
----------------------------------
Iterative call:
fibonacci(0) = 0
duration: 0.000714
----------------------------------
Iterative call:
fibonacci(1) = 1
duration: 0.000589
----------------------------------
Iterative call:
fibonacci(2) = 1
duration: 0.000497
----------------------------------
Iterative call:
fibonacci(3) = 2
duration: 0.000497
----------------------------------
Iterative call:
fibonacci(4) = 3
duration: 0.000497
----------------------------------
Iterative call:
fibonacci(5) = 5
duration: 0.000530
----------------------------------
Iterative call:
fibonacci(6) = 8
duration: 0.001458
----------------------------------
Iterative call:
fibonacci(7) = 13
duration: 0.001091
----------------------------------
Iterative call:
fibonacci(8) = 21
duration: 0.002063
----------------------------------
Iterative call:
fibonacci(9) = 34
duration: 0.001568
----------------------------------
Iterative call:
fibonacci(10) = 55
duration: 0.001613
----------------------------------
Iterative call:
fibonacci(11) = 89
duration: 0.001714
----------------------------------
Iterative call:
fibonacci(12) = 144
duration: 0.001108
----------------------------------
```

Abbildung 1: Test Teil 1

```
----------------------------------
Iterative call:
fibonacci(13) = 233
duration: 0.001514
----------------------------------
Iterative call:
fibonacci(14) = 377
duration: 0.002782
----------------------------------
Iterative call:
fibonacci(15) = 610
duration: 0.000934
----------------------------------
Iterative call:
fibonacci(16) = 987
duration: 0.001254
----------------------------------
Iterative call:
fibonacci(17) = 1597
duration: 0.001154
----------------------------------
Iterative call:
fibonacci(18) = 2584
duration: 0.001057
----------------------------------
Iterative call:
fibonacci(19) = 4181
duration: 0.001435
----------------------------------
Iterative call:
fibonacci(20) = 6765
duration: 0.002333
----------------------------------
Iterative call:
fibonacci(21) = 10946
duration: 0.001917
----------------------------------
Iterative call:
fibonacci(22) = 17711
duration: 0.002126
----------------------------------
Iterative call:
fibonacci(23) = 28657
duration: 0.003417
----------------------------------
Iterative call:
fibonacci(24) = 46368
duration: 0.005488
----------------------------------
Iterative call:
fibonacci(25) = 75025
duration: 0.007235
----------------------------------
Iterative call:
fibonacci(26) = 121393
duration: 0.011695
----------------------------------
Iterative call:
fibonacci(27) = 196418
duration: 0.017702
----------------------------------
```

Abbildung 2: Test Teil 2

```
----------------------------------
Iterative call:
fibonacci(27) = 196418
duration: 0.017702
----------------------------------
Iterative call:
fibonacci(28) = 317811
duration: 0.026852
----------------------------------
Iterative call:
fibonacci(29) = 514229
duration: 0.041560
----------------------------------
Iterative call:
fibonacci(30) = 832040
duration: 0.066955
----------------------------------
Iterative call:
fibonacci(31) = 1346269
duration: 0.104698
----------------------------------
Iterative call:
fibonacci(32) = 2178309
duration: 0.167805
----------------------------------
Iterative call:
fibonacci(33) = 3524578
duration: 0.274100
----------------------------------
Iterative call:
fibonacci(34) = 5702887
duration: 0.436546
----------------------------------
Iterative call:
fibonacci(35) = 9227465
duration: 0.713934
----------------------------------
Iterative call:
fibonacci(36) = 14930352
duration: 1.189259
----------------------------------
Iterative call:
fibonacci(37) = 24157817
duration: 1.894353
----------------------------------
Iterative call:
fibonacci(38) = 39088169
duration: 3.030520
----------------------------------
Iterative call:
fibonacci(39) = 63245986
duration: 4.836759
----------------------------------
duration: 12.925221
----------------------------------
```

Abbildung 3: Test Teil 3

```
Compile time call with Do - If statement:
fibonacci<10>::RET = 55
fibonacci<11>::RET = 89
fibonacci<12>::RET = 144
fibonacci<13>::RET = 233
fibonacci<14>::RET = 377
fibonacci<15>::RET = 610
fibonacci<16>::RET = 987
fibonacci<17>::RET = 1597
fibonacci<18>::RET = 2584
fibonacci<19>::RET = 4181
fibonacci<20>::RET = 6765
fibonacci<21>::RET = 10946
fibonacci<22>::RET = 17711
fibonacci<23>::RET = 28657
fibonacci<24>::RET = 46368
fibonacci<25>::RET = 75025
fibonacci<26>::RET = 121393
fibonacci<27>::RET = 196418
fibonacci<28>::RET = 317811
fibonacci<29>::RET = 514229
fibonacci<30>::RET = 832040
fibonacci<31>::RET = 1346269
fibonacci<32>::RET = 2178309
fibonacci<33>::RET = 3524578
fibonacci<34>::RET = 5702887
fibonacci<35>::RET = 9227465
fibonacci<36>::RET = 14930352
fibonacci<37>::RET = 24157817
fibonacci<38>::RET = 39088169
fibonacci<39>::RET = 63245986
fibonacci<40>::RET = 102334155
duration: 0.026492
----------------------------------
```

Abbildung 4: Test Teil 4

2 1, 2, 3, 4, 5, 6, ...