technische universität dortmund

fakultät für informatik

Elias Kuthe
Roman Kalkreuth
Bernd Zey

Dortmund, den 22. November 2019

Praktikum zur Vorlesung Einführung in die Programmierung WS 19/20

Blatt 6

Es können 21 Punkte erreicht werden.

Allgemeine Hinweise

a) Bitte lesen Sie vor der Bearbeitung **alle** Aufgaben sorgfältig durch! Dies erspart Ihnen unnötige Arbeit und somit auch Zeit!

b) Die einzigen Header, die Sie zur Bearbeitung der Aufgaben verwenden dürfen, sind `iostream` und solche, die laut Aufgabenstellung explizit erlaubt werden.

c) Lassen Sie sich fertiggestellte Aufgaben bitte möglichst **frühzeitig** testieren. In der letzten halben Stunde vor Schluss wird nur noch **eine** Aufgabe testiert!

d) Wir akzeptieren ein Testat nur, wenn die Lösung eigenständig auf Anhieb erklärt werden kann. Andernfalls müssen wir die entsprechende Teilaufgabe mit 0 Punkten bewerten.

e) Globale Variablen sind zur Lösung der Aufgaben nicht erlaubt!

Aufgabe 1: EidP-Folge (13 Punkte)

In dieser Aufgabe betrachten wir die weitgehend unbekannte EidP-Folge, bei der sich das aktuelle Glied durch die gewichtete Addition der beiden vorherigen Glieder ergibt. Die allgemeine Definition der Folge lautet

$$\text{eidp}(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ \text{eidp}(n-1) + 3 \cdot \text{eidp}(n-2) & n > 2\,. \end{cases}$$

Die ersten 10 Glieder sind dann 1, 1, 4, 7, 19, 40, 97, 217, 508, 1159.

a) Legen Sie ein neues Projekt `Aufgabe_6_1` an und fügen Sie darin eine Quelltextdatei `eidp-folge.cpp` hinzu. Implementieren Sie eine rekursive Funktion `rekEidp`, die einen Eingabeparameter `n` vom Typ `unsigned int` erhält. Sie soll die `n`-te EidP-Zahl gemäß der oben angegebenen Vorschrift rekursiv berechnen und als `long long` zurückgeben.

_____ (3)

b) Fügen Sie eine nicht-rekursive (iterative) Funktion `itEidp` hinzu. Sie soll die gleiche Eingabe erhalten und dasselbe Ergebnis berechnen wie `rekEidp`. Allerdings soll sie dabei iterativ vorgehen und auf Rekursion komplett verzichten.

_____ (3)

c) Eine dritte Funktion `moivreBinetEidP` soll schließlich die `n`-te EidP-Zahl zurückgeben, indem die folgende Gleichung genutzt wird:

$$\text{eidp}(n) = \frac{1}{\sqrt{13}} \left[\left(\frac{1+\sqrt{13}}{2} \right)^n - \left(\frac{1-\sqrt{13}}{2} \right)^n \right]$$

Auch die Funktion `moivreBinetEidp` erhält einen Parameter vom Typ `unsigned int` und gibt das Ergebnis als `long long` zurück.
Zur Berechnung der Wurzeln und Potenzen können Sie mittels `#include <cmath>` die Bibliothek `cmath` am Anfang einer Quelldatei einbinden. Anschließend können Sie die Wurzel einer Variablen `x` mit `sqrt(x)` berechnen. Die `b`-te Potenz von `a` berechnen Sie mit `pow(a, b)`.

_____ (3)

d) Testen Sie die Funktionen `rekEidP`, `itEidP` und `moivreBinetEidP`, indem Sie sie von einer `main`-Funktion aus aufrufen und die Rückgabewerte ausgeben lassen. Tragen Sie die Ergebnisse für die angegebenen Werte von `n` in die folgende Tabelle ein und interpretieren Sie Ihre Resultate.

Argument	Ergebnis		
`n`	`rekEidP`	`itEidP`	`moivreBinetEidP`
9			
17			
22			
49			

_____ (1)

e) Erweitern Sie die Funktionen `rekEidP` und `itEidP` so, dass die Anzahl der Additionen gezählt wird. Verwenden Sie dazu eine Variable `count` vom Typ `unsigned int`, die in der Funktion `main` deklariert wird.
Dokumentieren Sie die Zählerwerte in der folgenden Tabelle und interpretieren Sie die Resultate:

Argument	Anzahl der Additionen	
`n`	`rekEidP`	`itEidP`
9		
17		
22		
49		

_____ (3)

Aufgabe 2: Ackermann-Funktion (8 Punkte)

Die Ackermann-Funktion ist eine 1926 von Wilhelm Ackermann gefundene, extrem schnell wachsende mathematische Funktion, mit deren Hilfe in der theoretischen Informatik Grenzen von Computer- und Berechnungsmodellen aufgezeigt werden können.
Die vereinfachte Definition der Funktion lautet wie folgt:

$$\begin{aligned} a(0,m) &= m+1 \\ a(n+1,0) &= a(n,1) \\ a(n+1,m+1) &= a(n,a(n+1,m)) \end{aligned}$$

a) Legen Sie ein neues Projekt `Aufgabe_6_2` an und fügen Sie eine leere C++-Quelldatei `ackermann.cpp` hinzu. Schreiben Sie eine rekursive Funktion `int ack(int n, int m)`, die die oben genannte Berechnungsvorschrift implementiert.

_____ (2)

b) Offenbar kann man die Ackermann-Funktion auch durch folgenden Algorithmus implementieren:

```
function ack2(n, m)
    while n != 0
        if m = 0
            m := 1
        else
            m := ack2(n, m - 1)
        n := n - 1
    return m + 1
```

Implementieren Sie auch diese Variante in C++.

_____ (3)

c) Schreiben Sie eine `main`-Funktion, in der beide Funktionen mit denselben Parametern aufgerufen werden. Tragen Sie in die unten stehende Tabelle die Ergebnisse ein. Sind die Ergebnisse identisch?

n	m	`ack(n, m)`	`ack2(n, m)`
1	52		
2	2		
2	7		
2	13		

Können Sie auch für $n = 3$ Ergebnisse berechnen?

HINWEIS: Laufende Programme können Sie mit der Tastenkombination `Strg + F2` abbrechen.

_____ (3)