technische universität dortmund

fakultät für informatik

Andre Droschinsky
Roman Kalkreuth
Denis Kurz

Dortmund, den 29. November 2018

Praktikum zur Vorlesung Einführung in die Programmierung WS 18/19

Blatt 6

Es können 19 (+3 Bonus) Punkte erreicht werden.

Allgemeine Hinweise

1. Bitte lesen Sie vor der Bearbeitung **alle** Aufgaben sorgfältig durch! Dies erspart Ihnen unnötige Arbeit und somit auch Zeit!
2. Die einzigen Header, die Sie zur Bearbeitung der Aufgaben verwenden dürfen, sind `iostream` und solche, die laut Aufgabenstellung explizit erlaubt werden.
3. Lassen Sie sich fertiggestellte Aufgaben bitte möglichst **frühzeitig** testieren. In der letzten halben Stunde vor Schluss wird nur noch **eine** Aufgabe testiert!
4. Wir akzeptieren ein Testat nur, wenn die Lösung eigenständig auf Anhieb erklärt werden kann. Andernfalls müssen wir die entsprechende Teilaufgabe mit 0 Punkten bewerten.

Aufgabe 1: 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[1].
Die vereinfachte Definition der Funktion $a : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ 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_1` an und fügen Sie eine leere C++-Quelldatei `ackermann.cpp` hinzu. Schreiben Sie eine rekursive Funktion `int ackermann(int n, int m)`, die die oben genannte Berechnungsvorschrift implementiert.

_____ (3)

[1] https://de.wikipedia.org/wiki/Ackermannfunktion

b) Die Ackermann-Funktion lässt sich auch durch folgenden Algorithmus implementieren:

```
function ackermann2(n, m)
    while n != 0
        if m = 0
            m := 1
        else
            m := ackermann2(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	`ackermann(n, m)`	`ackermann2(n, m)`
0	20		
1	100		
2	30		
3	12		

Können Sie auch für $n = 4$ Ergebnisse berechnen?

_____ (2)

d) Bonusaufgabe. (Diese Aufgabe sollten Sie erst nach Aufgabe 2 bearbeiten.)
Schreiben Sie zunächst eine C++-Funktion `int ackermann_geschlossen(int n, int m)`, die immer -1 zurückgibt.
Programmieren Sie anschließend in der `main`-Funktion eine Ausgabe, die in vier Spalten die Werte n, m, `ackermann(n,m)` und `ackermann_geschlossen(n,m)` für alle $(n,m) \in \{0,1,2,3\} \times \{0,1,2,\ldots,10\}$ ausgibt.

Leiten Sie aus der Funktionsvorschrift und/oder der Ausgabe von `ackermann` eine geschlossene Form (nicht rekursiv oder iterativ) für die Ackermann-Funktion für jedes $n \in \{0,1,2,3\}$ her und notieren Sie diese 4 Funktionen auf dem Aufgabenblatt.

- `ackermann_geschlossen(0,m)=`
- `ackermann_geschlossen(1,m)=`
- `ackermann_geschlossen(2,m)=`
- `ackermann_geschlossen(3,m)=`

Fügen Sie anschließend in Ihre C++-Funktion `int ackermann_geschlossen(int n, int m)` ein `switch-case` für $n \in \{0,1,2,3\}$ ein, so dass das Ergebnis nach obiger Formel berechnet und zurückgegeben wird. Für andere Werte von n soll (weiterhin) -1 zurückgegeben werden.

_____ (3)

Aufgabe 2: Lindhauer-Folge (11 Punkte)

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

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

Wir beschränken uns hier auf den Spezialfall $a = 3$, $b = 1$. Die ersten 10 Glieder sind dann 1, 1, 4, 7, 19, 40, 97, 217, 508, 1159.

a) Legen Sie ein neues Projekt `Aufgabe_6_2` an und fügen Sie darin eine Quelltextdatei `lindhauer.cpp` hinzu. Implementieren Sie eine rekursive Funktion `rekLind`, die einen Eingabeparameter `n` vom Typ `unsigned int` erhält. Sie soll die `n`-te Lindhauer-Zahl gemäß der oben angegebenen Vorschrift rekursiv berechnen und als `long long` zurückgeben.

_____ (2)

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

_____ (3)

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

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

Auch die Funktion `moivreBinetLind` 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)`.

_____ (2)

d) Testen Sie die Funktionen `rekLind`, `itLind` und `moivreBinetLind`, 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`	`rekLind`	`itLind`	`moivreBinetLind`
8			
16			
24			
50			

_____

_____

_____

_____ (2)

e) Erweitern Sie die Funktionen `rekLind` und `itLind` 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. Verzichten Sie dabei vollständig auf globale Variablen.
Dokumentieren Sie die Zählerwerte in der folgenden Tabelle und interpretieren Sie die Resultate:

Argument	Anzahl der Additionen	
`n`	`rekLind`	`itLind`
8		
16		
24		
50		

EINSCHRÄNKUNG: Verwenden Sie keine globalen Variablen!

(2)