"Informatik II"

Studiengänge B_BT, B_Me, B_Sy, B_SK

2017

Hinweis: In der vorliegenden Musterklausur werden beispielhaft mögliche Aufgentypen gezeigt. In der Klausur können ausdrücklich auch andere Konstellationen auftreten.

1. Objektorientierte Programmierung und dynamische Datenstrukturen (32 Punkte)

   Gegeben sei eine Datenstruktur mit folgendem Aufbau

```
struct lelem {
   int w;
   lelem * pNext;
};
```

   Basierend auf dieser Datenstruktur soll eine Klasse **liste** mit der folgenden Klassendefinition aufgebaut werden:

```
class liste {
     private:
        lelem * anchor;           // Zeiger auf den Beginn einer von
                                  // liste verwalteten einfach verketteten
                                  // Liste
     public:
        liste();                  // Konstruktor  zum Konstruieren
                                  // einer leeren Liste
        ~liste();                 // Destruktor, gibt den gesamten dynamisch
                                  // angeforderten Speicherplatz frei
        void einfuegen (int x);    // einfuegen eines  neuen Listenelementes
        bool loeschen (int x);    // loeschen eines  Listenelementes
        int count_wert (int wert, int & count)
                                  // Auswertemethode
        bool ausgabe_liste_datei(string  filename);
```

```
                // Ausgabe aller Datenelemente der Liste in Datei
        friend ostream & operator << (ostream & os, const liste & li);
};
```

Die Funktionalität der Methoden der Klasse **liste** ist wie folgt definiert:

- einfuegen:
  Die Methode fügt an das Ende der einfach verketteten Liste, die durch den privaten Member **anchor** adressiert wird, ein neues Listenelement mit dem Datenwert des Parameters x ein. Als Nachfolgewert soll die Konstante **NULL** eingetragen werden. Ist die Liste leer, soll ein neues erstes Listenelement eingefügt werden. 4 Punkte
- loeschen:
  Die Methode **loeschen** entfernt das erste Listenelement aus der einfach verketteten Liste, die durch den privaten Member **anchor** adressiert wird, dessen Datenwert **w** den gleichen Wert wie der Parameter **x** hat. Die Struktur der einfach verketten Liste soll erhalten bleiben.
  - false - Löschen ist missglückt, es existiert kein Listenelement mit den angegebenem Datenwert.
  - true - Löschen ist gelungen. 7 Punkte
- count_wert:
  In der Methode **count_wert** wird die Anzahl der Elemente der Datenstruktur, die durch den privaten Member **anchor** adressiert wird, ermittelt, deren Datenwerte **w** den gleichen Wert haben wie der Parameter **wert**. Diese Anzahl wird der rufenden Funktion über den Referenzparameter **count** mitgeteilt. Die Methode liefert als Rückgabewert die Anzahl der Elemente, die in der Datenstruktur, die über den privaten Member **anchor** adressiert wird, enthalten sind. Es ist nicht davon auszugehen, dass die Anzahl der Elemente in der Datenstruktur, die durch den privaten Member **anchor** adressiert wird, bekannt ist. 4 Punkte
- ausgabe_liste_datei:
  Die Datenelemente der einfach verketteten Liste sollen in eine Datei geschrieben werden, deren Name vom Nutzer anzugeben ist. Die Methode gibt einen Integer-Wert mit folgender Bedeutung zurück:
  - false - Ausgeben in Datei missglückt
  - true - Ausgeben in Datei gelungen 5 Punkte
- Friend-Operatorfunktion

  ```
  friend ostream & operator << (ostream & os, const liste & li)
  ```

  die die Datenelemente, die in einem Objekt der Klasse *liste* verwaltet werden, auf ein Objekt der Klasse ostream (in der Regel cout) ausgibt. (5 Punkte)

Implementieren Sie den Konstruktor (1 Punkt), den Destruktor (2 Punkte) sowie die angegebenen Methoden der Klasse **liste**. Schreiben Sie ein Rahmenprogramm, das **alle** oben beschriebenen Methoden benutzt. Dabei soll eine Liste mit mindestens 20 Elementen aufgebaut werden, wobei jeweils der Nutzer zur Eingabe eines Datenwertes aufgefordert werden soll. Die Liste selbst wird durch den Ruf der Methode **einfuegen** aufgebaut. Demonstrieren Sie dann die Verwendung der Methode **count_wert** im Hauptprogramm zur Bestimmung und Anzeige des prozentualen Anteils von Elementen mit einem bestimmten Wert an der Gesamtanzahl aller Elemente der Datenstruktur, die durch den privaten Member **anchor** adressiert wird. (4 Punkte)

2. Algorithmierung (12 Punkte)

   Ein Primzahlpalindrom ist eine Primzahl, deren Ziffern von vorn und von hinten gelesen die gleiche Zahl ergeben, analog zum Palindrom, das von vorn und von hinten gelesen das gleiche Wort ergibt.
   Wenn **p** eine Zahl ist und x_i die Ziffer x an der Stelle i der Zahl p, so gilt
   $p = x_{n-1}x_{n-2}\ldots x_1x_0 = x_0x_1\ldots x_{n-2}x_{n-1}$

   Beispiele für Primzahlpalindrome: 2, 3, 5, 7, 11, 101

   Entwerfen und implementieren Sie ein C++-Programm, das alle Primzahlpalindrome im Intervall von 2 bis 1000 ermittelt und ausgibt. Die Ausgabe soll so erfolgen, dass jede Zahl linksbündig mit einer Breite von 6 Zeichen dargestellt werden soll.

   Hinweis: Eine Primzahl ist nur durch 1 und durch sich selbst ohne Rest teilbar. Die kleinste Primzahl ist die 2.

3. Dynamische Datenstrukturen (16 Punkte)

   Gegeben sei folgendes C++-Programm:

→ Bitte wenden!

```
#include <iostream>
using namespace std;
struct lelem{
  int a;
  lelem * pNext;
};
int main()
{
  lelem * pNeu, * pAnker = new lelem;
  int i,j;
  cin >> pAnker -> a;
  pAnker -> pNext = pAnker;
  for ( i = 1; i < 10; i++)
    {
      pNeu = new lelem;
      cin >> j;
      pNeu -> a = j % i;
      pNeu -> pNext = pAnker;
      pAnker= pNeu;
    }
  return 0; }
```

a. Stellen Sie die Datenstruktur grafisch dar, die in diesem Programm aufgebaut wird (inklusive aller vorkommenden Zeiger nach Abarbeitung des Programms) bei folgender Eingabefolge: 3 5 7 11 13 17 19 23 29 31. Die Reihenfolge des Aufbaus der dynamischen Datenstruktur muss erkennbar sein. 6 Punkte

b. Ergänzen Sie das Programm um eine Funktion

```
bool alarm (int & anzahl, int wert,p *  pAnker1)
```

mit folgender Funktionalität:
In der Funktion **alarm** wird die Anzahl der a-Instanzen (Datenelemente) der Datenstruktur, die durch den Parameter **pAnker1** adressiert wird, die den Wert **wert** haben, ermittelt. Die Anzahl wird der rufenden Funktion über den Referenzparameter **anzahl** übermittelt. Die Funktion liefert als Rückgabewert den Wert **true**, wenn mehr als die Hälfte der a-Instanzen der Elemente der durch den Parameter **pAnker1** adressierten Datenstruktur den Wert **wert** besitzen, sonst den Wert **false**. Es ist nicht davon auszugehen, dass die Anzahl der Elemente bekannt ist. 5 Punkte

c. Testen Sie den Ruf der Funktion **alarm** so, dass der Nutzer aufgefordert wird, eine Zahl einzugeben und er dann davon informiert wird, ob diese Zahl in mehr als 50 % der a-Instanzen der durch den Zeiger **pAnker** adressierten Datenstruktur enthalten ist oder nicht. Ergänzen Sie das Programm so, dass der gesamte dynamisch allokierte Speicherplatz wieder freigegeben wird 5 Punkte