Aufgabe 1

Objektorientierte Programmierung

Gegeben seien folgende Datensatzdefinitionen:

```
struct buecher{
  string autor;
  string titel;
  bool verfuegbar;
  int lfdNr;
  buecher * next;
};

struct nutzer{
  string name;
  string vorname;
  int lfdNr;
  nutzer * next;
};

struct ausleihe{
   int buchnr;
   int nutzernr;
   ausleihe * next;
};
```

Aus diesen Datensatzdefinitionen soll eine Klasse bibliothek aufgebaut werden, die folgende Komponenten enthält:

Zeiger auf den Anfang einer einfach verketteten Liste buchliste mit nullptr als Endekennzeichen.
Zeiger auf den Anfang einer einfach verketteten Liste nutzerliste mit nullptr als Endekennzeichen.
Zeiger auf den Anfang einer einfach verketteten Liste ausgeliehen mit nullptr als Endekennzeichen.

Diese Listen sollen nur von Methoden der Klasse bibliothek aus erreichbar sein.
Konstruktor ohne Parameter --> die Zeiger sollen mit nullptr initialisiert werden.
Konstruktor mit einem Parameter vom Typ C++-Zeichenkette -> der Parameter soll als Dateiname interpretiert werden. In der Datei steht die Anzahl der Bücher und folgend die Werte Autor und Titel für jedes Buch. Die Komponente verfuegbar jedes Buches soll den Wert true erhalten, die Komponente lfdNr soll mit dem Wert 1 beginnend durchnummeriert werden.
Destruktor --> der dynamisch allokierte Speicherplatz der drei Listen soll freigegeben werden
Methode neues_buch mit zwei Parametern (Autor, Titel) --> wenn noch kein Buch mit den übergebenen Parametern in der durch buchliste adressierten Liste existiert, soll ein neues Buch an das Ende der Liste eingefügt werden, die Komponente verfuegbar soll den Wert true erhalten, die Komponente lfdNr den um 1 erhöhten Wert des bisherigen letzten Elementes. Wenn das Buch bereits in der Liste enthalten ist, soll nichts geschehen. Das rufende Programm ist über den Erfolg des Einfügens zu informieren.
Methode buch_aussondern mit zwei Parametern (Autor, Titel) --> wenn ein Buch mit den übergebenen Parametern existiert und verfügbar ist, soll es aus der durch buchliste adressierten Liste entfernt werden. Das rufende Programm ist über den Erfolg des Aussonderns zu informieren.
Methode buecherliste --> der gesamte Buchbestand der Bibliothek ist wie folgt auszugeben: Autor (30 Zeichen), Titel (30 Zeichen) verfügbar (j/n) (1 Zeichen). Als Füllzeichen ist der Punkt '.' zu verwenden.

Methode buecherliste_verfuegbar --> der gesamte verfügbare Buchbestand der Bibliothek ist wie folgt auszugeben: Autor (30 Zeichen), Titel (30 Zeichen). Als Füllzeichen ist der Punkt '.' zu verwenden.
Methode buecherliste_to_file mit einem Parameter (Dateiname vom Typ C++-String) --> der gesamte Buchbestand der Bibliothek ist in eine Datei auszugeben. Das rufende Programm ist über den Erfolg der Ausgabe zu informieren.
Methode neuer_nutzer mit zwei Parametern (Name, Vorname) --> wenn noch kein Nutzer mit den übergebenen Parametern in der durch nutzerliste adressierten Liste existiert, soll ein neuer Nutzer an das Ende der Liste eingefügt werden, die Komponente lfdNr soll den um 1 erhöhten Wert des bisherigen letzten Elementes erhalten. Wenn der Nutzer bereits in der Liste enthalten ist, soll nichts geschehen. Das rufende Programm ist über den Erfolg des Einfügens zu informieren.
Methode nutzer_austragen mit zwei Parametern (Name, Vorname) --> wenn ein Nutzer mit den übergebenen Parametern existiert und der Nutzer keine Bücher ausgeliehen hat, soll er aus der durch nutzerliste adressierten Liste entfernt werden. Das rufende Programm ist über den Erfolg des Aussonderns zu informieren.
Methode nutzer_hat_buecher mit zwei Parametern (Name, Vorname) --> es sollen alle Bücher, die der Nutzer mit den übergebenen Parametern ausgeliehen hat, wie folgt ausgegeben werden: Autor (30 Zeichen), Titel (30 Zeichen). Als Füllzeichen ist der Punkt '.' zu verwenden. Wenn kein Nutzer existiert, soll ein entsprechender Hinweis ausgegeben werden.
Methode nutzer_leiht_aus mit vier Parametern (Name, Vorname, Autor, Titel) --> wenn ein Nutzer mit den Parametern Name, Vorname existiert und ein Buch mit den Parametern Autor, Titel existiert und verfügbar ist, soll ein neuer Eintrag mit den Werten lfdNr der Nutzerliste und lfdNr der Bücherliste an das Ende der durch ausgeliehen adressierten Liste angehängt werden. Der entsprechende Eintrag verfuegbar in der Bücherliste ist mit dem Wert false zu belegen. Das rufende Programm ist über den Erfolg des Ausleihens zu informieren.
Methode nutzer_gibt_zurueck mit vier Parametern (Name, Vorname, Autor, Titel) --> wenn ein Nutzer mit den Parametern Name, Vorname existiert und ein Buch mit den Parametern Autor, Titel existiert, soll ein Eintrag mit den Werten lfdNr der Nutzerliste und lfdNr der Bücherliste aus der durch ausgeliehen adressierten Liste gelöscht werden. Der entsprechende Eintrag verfuegbar in der Bücherliste ist mit dem Wert true zu belegen. Das rufende Programm ist über den Erfolg des Ausleihens zu informieren.

Methode zum Überladen des Ausgabeoperators << --> es sollen alle Nutzer und alle Bücher in der Standardausgabe ausgegeben werden. Die Bücher und Nutzer sollen als separate Tabellen (autor: Breite 30 Zeichen, titel: Breite 40 Zeichen, verfuegbar: Breite 1 Zeichen, name: Breite 20 Zeichen, vorname: Breite 20 Zeichen) ausgegeben werden.

Ihre Aufgaben innerhalb Ihres Teams lauten:

Geben Sie die Klassendefinition der Klasse bibliothek an. Member sollen wie beschrieben enthalten sein.

Implementieren Sie die Konstruktoren, den Destruktor sowie die Methoden buch_aussondern, buecherliste sowie die Methode zum Überladen des Ausgabeoperators.

Implementieren Sie eine Funktion main(), in der 2 Objekte der Klasse bibliothek (ohne Parameter, ein Parameter) erzeugt werden und alle in der Klassendefinition enthaltenen Methoden einmal aufgerufen werden.

Hinweis: Die Benutzung der Bibliothek fstream ist erlaubt.

Klassendefinition:

```
class bibliothek {
  private:
     buecher *buchliste;
     nutzer *nutzerliste;
     ausleihe * ausgeliehen;
  public:
     bibliothek();
     bibliothek(string dateiname);
     ~bibliothek();
     bool neues_buch(string autor, string titel);
     bool buch_aussondern(string autor, string titel);
     void buecherliste();
     void buecherliste_verfuegbar();
     bool buecherliste_to_file(string dateiname);
     bool neuer_nutzer(string name, string vorname);
     bool nutzer_austragen(string name, string vorname);
     void nutzer_hat_buecher(string name, string vorname);
     bool nutzer_leiht_aus(string name, string vorname, string autor, string titel);
     bool nutzer_gibt_zurueck(string name, string vorname, string autor, string titel);
     friend ostream & operator << (ostream & os, bibliothek & bibo);
};
```

Konstruktor ohne Parameter:

```
bibliothek::bibliothek()
{
    buchliste = nullptr;
    nutzerliste = nullptr;
    ausgeliehen = nullptr;
}
```

Konstruktor mit einem Parameter:

```
bibliothek::bibliothek(string dateiname)
 {
    nutzerliste = nullptr;       //in einer „neuen“ Bibliothek gibt es keine Nutzer
    ausgeliehen = nullptr;    //und es ist auch noch nichts ausgeliehen
    ifstream fin(dateiname);  //oder  ifstream fin(dateiname.c_str());
    if (!fin) {
       buchliste = nullptr;     //dann gibt es auch keinen Buchbestand;-)
     }
     else
        {
              int anz;
              fin >>anz;
               if (anz<1)  //keine Buecher in Datei
               {
                  buchliste = nullptr;
               }
                  else
                     {
                        buchliste = new buecher;
        ……
```

Konstruktor mit einem Parameter:

```
    ....
                        fin >> buchliste -> autor >> buchliste -> titel;
                        buchliste -> verfuegbar = true;
                        buchliste -> lfdNr = lfdNr_buecher;
                        buchliste -> next = nullptr;
                        buecher * pHelp = buchliste;
                        buecher * pNew;
                        for (int i=2; i<= anz; i++)   //das erste Buch wurde schon gelesen
                           {
                              pNew = new buecher;
                              fin >> pNew -> autor >> pNew -> titel;
                              pNew -> verfuegbar = true;
                              pNew -> lfdNr = lfdNr_buecher;
                              pNew -> next = nullptr;
                              pHelp -> next = pNew;
                              pHelp = pNew;
                           }
                        }
              fin.close();
          }
}
```

Destruktor:

```
bibliothek::~bibliothek()
{
    buecher * buchdel;
    nutzer * nutzerdel;
    ausleihe * ausleihedel;
    //alle drei Liste Loeschen
    while (buchliste)
      {
          buchdel = buchliste;
          buchliste = buchliste→next;
          delete buchdel;
       }
    while (nutzerliste)
      {
          nutzerdel = nutzerliste;
          nutzerliste = nutzerliste→next;
          delete nutzerdel;
       }
     while (ausgeliehen)
      {
          ausleihedel = ausgeliehen;
          ausgeliehen = ausgeliehen→next;
          delete ausleihedel;
       }
}
```

Methode buch_aussondern:

```
 bool bibliothek::buch_aussondern(string autor, string titel)
{
 buecher * buchdel, * buchHelp;
  if (buchliste == nullptr)
       return false;   //in leerer Liste kann nichts gelöscht werden
  if (buchliste -> autor == autor && buchliste -> titel == titel)      //Erstes Buch aus der Liste
                                                                       //ist zu loeschen
  {
     if (buchliste -> verfuegbar == true)  //gesuchtes Buch ist verfuegbar
     {
        buchdel = buchliste;
        buchliste = buchliste -> next;
        delete buchdel;
        return true;
     }
     else
      return false;   //gesuchtes Buch ist nicht verfuegbar
  }
  ....
```

Methode buch_aussondern:

```
  buchHelp = buchliste;
  while (buchHelp -> next != nullptr)
  {
     if (buchHelp -> next-> autor == autor && buchHelp -> next-> titel == titel)
     {
        if (buchHelp -> next -> verfuegbar == true)
        {
           buchdel = buchHelp -> next;
           buchHelp -> next = buchdel -> next;
           delete buchdel;
           return true;
        }
        else
           return false;  //siehe oben
     }
     buchHelp = buchHelp -> next;
  }
  return false;
}
```

Methode buecherliste:

```
 void bibliothek::buecherliste()
{
    buecher * buecherHelp;
    if (buchliste == nullptr)
        cout << "keine Buecher im Bestand" « endl;
     else
     {
         buecherHelp = buchliste;
         while (buecherHelp != nullptr)
         {
             cout << setfill('.') << setw(30) << buecherHelp -> autor;
             cout << setfill('.') << setw(30) << buecherHelp -> titel;
             if (buecherliste -> verfuegbar == true)
                cout << "j\n";
               else
                 cout << "n\n";
             buecherHelp = buecherHelp -> next;
         }
     }
}
```

Operatorfunktion <<:

```
 ostream & operator << (ostream & os, bibliothek & bibo)
{
   buecher * buecherHelp;
   nutzer * nutzerHelp;
   if (bibo.buchliste == nullptr)
      os << "keine Buecher im Bestand" << endl;
    else
    {
       buecherHelp = bibo.buchliste;
       os << "Buecher im Bestand\n";
       while (buecherHelp != nullptr)
       {
          os << setfill('.') << setw(30) << buecherHelp -> autor;
          //os.width(30);
          //os.fill('.');
          os << setfill('.') << setw(40) << buecherHelp -> titel;
          if (buecherHelp -> verfuegbar == true)
             os << "j\n";
            else
             os << "n\n";
          buecherHelp = buecherHelp -> next;
       }
    }
    .....
```

Operatorfunktion <<:

```
...

    if (bibo.nutzerliste == nullptr)
       os << "keine Nutzer registriert" << endl;
    else
    {
        nutzerHelp = bibo.nutzerliste;
        os << "registrierte Benutzer\n";
        while (buecherHelp != nullptr)
        {
            os << setfill('.') << setw(20) << nutzerHelp -> name;
            //os.width(30);
            //os.fill('.');
            os << setfill('.') << setw(20) << nutzerHelp -> vorname;
            buecherHelp = buecherHelp -> next;
        }
        os << endl;
    }
  return os;
}
```

Funktion main():

```
int main()
    {
       //Minimalversion
       string dateiname;
       cout << "Dateiname?\n";
       cin >> dateiname;
       bibliothek bibo1;
       bibliothek bibo2(dateiname);

       bibo1.neues_buch("Mueller","Dies und das");
       bibo1.neues_buch("Mueller","Dies und jenes");
       bibo2.buecherliste();
       bibo2.buecherliste_verfuegbar();
       bibo2.buch_aussondern("Goethe","Faust");
       bibo2.buecherliste();
       bibo2.neuer_nutzer("Otto","Normal");
       bibo2.nutzer_hat_buecher("Otto","Normal");
       bibo2.nutzer_leiht_aus("Otto","Normal","Goethe","Faust");
       bibo2.nutzer_gibt_zurueck("Otto","Normal","Goethe","Faust");
       bibo2.nutzer_austragen("Otto","Normal");
       cout << bibo2 << endl;
       return 0;
    }
```

Musterlösung:

https://www.tu-chemnitz.de/informatik/friz/Grundl-Inf/Aufgaben/beispiele/musterklausur_oop.cpp

Übungsaufgabe:

Implementieren Sie die Methoden, die bisher nur als Platzhalter existieren.

Aufgabe 2

Dynamische Datenstrukturen:

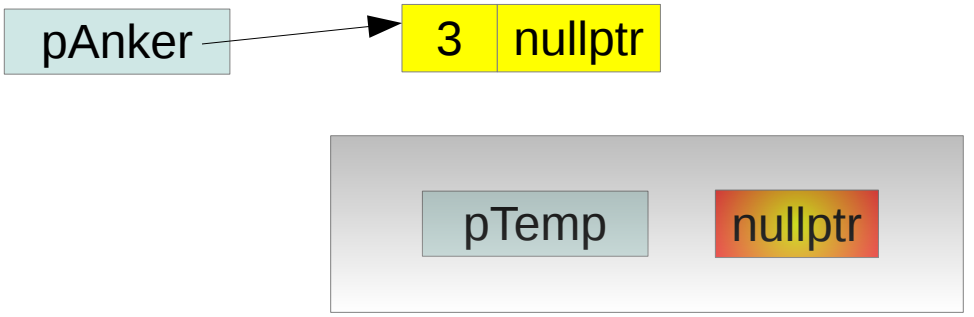
```
#include <iostream>

using namespace std;

struct lelem{
  int a;
  lelem * pNext;
};
int main()
{
  lelem * pTemp, * pAnker = new lelem;
  int i,j;
  cin >> j;
  pAnker -> a = j;
  pAnker -> pNext = nullptr;
  for ( i = 1; i < 10; i++)
    {
      pTemp = pAnker -> pNext;
      pAnker -> pNext = new lelem;
      cin >> j;
      pAnker -> pNext -> a = j % i;
      pAnker -> pNext -> pNext = pTemp;
    }
}
```

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.

```
#include <iostream>

using namespace std;

struct lelem{
  int a;
  lelem * pNext;
};
int main()
{
  lelem * pTemp, * pAnker = new lelem;
  int i,j;
  cin >> j;
  pAnker -> a = j;
  pAnker -> pNext = nullptr;
  for ( i = 1; i < 10; i++)
    {
      pTemp = pAnker -> pNext;
      pAnker -> pNext = new lelem;
      cin >> j;
      pAnker -> pNext -> a = j % i;
      pAnker -> pNext -> pNext = pTemp;
    }
}
```

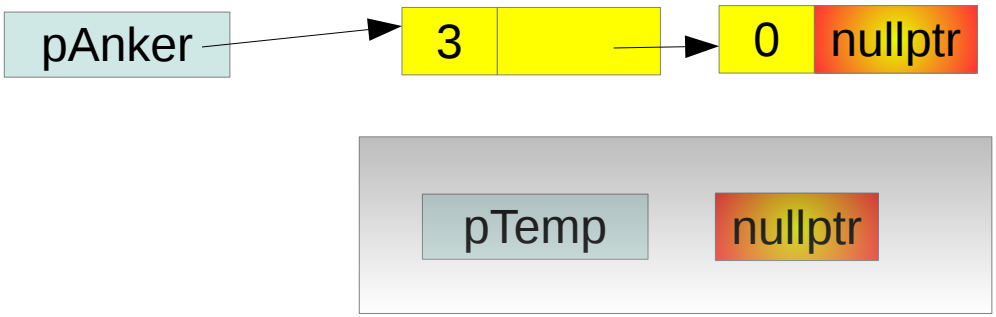
2. Dynamische Datenstrukturen:

```
#include <iostream>

using namespace std;

struct lelem{
  int a;
  lelem * pNext;
};
int main()
{
  lelem * pTemp, * pAnker = new lelem;
  int i,j;
  cin >> j;
  pAnker -> a = j;
  pAnker -> pNext = nullptr;
  for ( i = 1; i < 10; i++)
    {
      pTemp = pAnker -> pNext;
      pAnker -> pNext = new lelem;
      cin >> j;
      pAnker -> pNext -> a = j % i;
      pAnker -> pNext -> pNext = pTemp;
    }
}
```



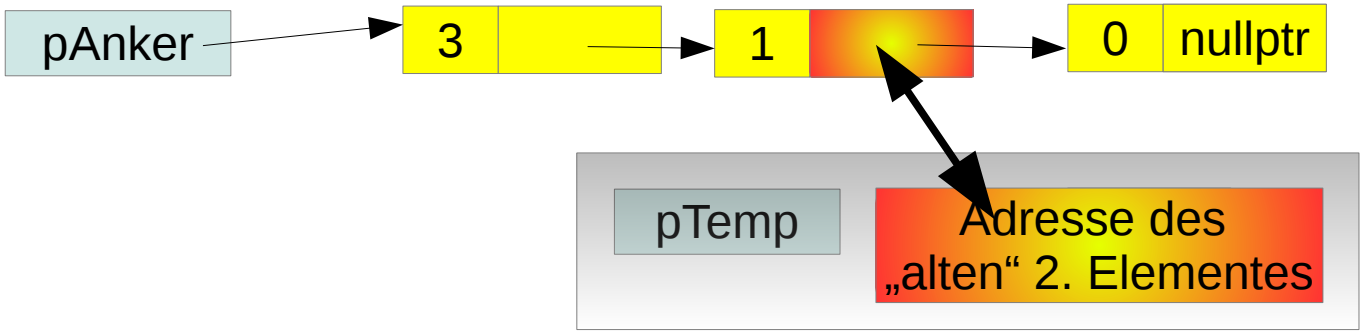
2. Dynamische Datenstrukturen:

```
#include <iostream>

using namespace std;

struct lelem{
  int a;
  lelem * pNext;
};
int main()
{
  lelem * pTemp, * pAnker = new lelem;
  int i,j;
  cin >> j;
  pAnker -> a = j;
  pAnker -> pNext = nullptr;
  for ( i = 1; i < 10; i++)
    {
      pTemp = pAnker -> pNext;
      pAnker -> pNext = new lelem;
      cin >> j;
      pAnker -> pNext -> a = j % i;
      pAnker -> pNext -> pNext = pTemp;
    }
}
```



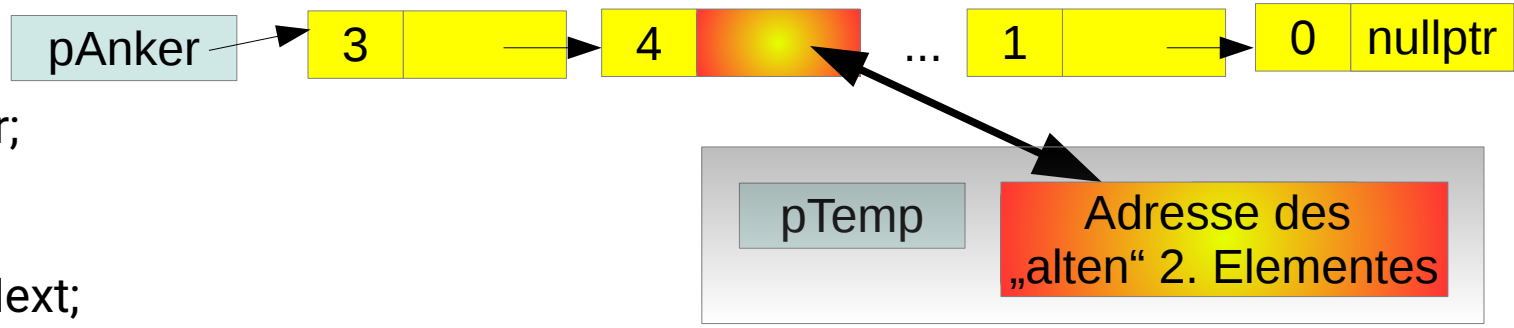
2. Dynamische Datenstrukturen:

```
#include <iostream>

using namespace std;

struct lelem{
  int a;
  lelem * pNext;
};
int main()
{
  lelem * pTemp, * pAnker = new lelem;
  int i,j;
  cin >> j;
  pAnker -> a = j;
  pAnker -> pNext = nullptr;
  for ( i = 1; i < 10; i++)
    {
      pTemp = pAnker -> pNext;
      pAnker -> pNext = new lelem;
      cin >> j;
      pAnker -> pNext -> a = j % i;
      pAnker -> pNext -> pNext = pTemp;
    }
}
```



2. Dynamische Datenstrukturen:

```
#include <iostream>

using namespace std;

struct lelem{
  int a;
  lelem * pNext;
};
int main()
{
  lelem * pTemp, * pAnker = new lelem;
  int i,j;
  cin >> j;
  pAnker -> a = j;
  pAnker -> pNext = nullptr;
  for ( i = 1; i < 10; i++)
    {
      pTemp = pAnker -> pNext;
      pAnker -> pNext = new lelem;
      cin >> j;
      pAnker -> pNext -> a = j % i;
      pAnker -> pNext -> pNext = pTemp;
    }
}
```



Datenwerte:

3 4 5 2 1 2 1 2 1 0

b) Ergänzen Sie das Programm um eine Funktion

int count_wert (int wert,lelem * pAnker1)

mit folgender Funktionalität:
In der Funktion **count_wert** wird die Anzahl der Elemente der Datenstruktur, die durch den Zeiger **pAnker1** adressiert wird, ermittelt, deren Datenwerte **a** einen Wert haben, der gleich ist wie der Parameter **wert**. Diese Anzahl wird der rufenden Funktion über den entsprechenden Rückgabewert mitgeteilt. Wenn der Funktion die leere Liste übergeben wurde, so soll die Funktion **count_wert** den Wert **-1** zurückgeben. Es ist nicht davon auszugehen, dass die Anzahl der Elemente in der Datenstruktur, die durch den Zeiger **pAnker1** adressiert wird, bekannt ist.

b) Ergänzen Sie das Programm um eine Funktion

```
int count_wert (int wert,lelem *  pAnker1)
{
   lelem * pH = pAnker1;
   int anz = 0;
   if (pH == nullptr)
      return -1;
   while (pH != nullptr)
      {
         if (pH->a == wert)
             anz++;
         pH = pH->pNext;
      }
    return anz;
}
```

c) Ergänzen Sie das Programm um eine Funktion

bool delete_elem(int value, lelem *& pAnchor1)

mit folgender Funktionalität:
In der Funktion **delete_elem** wird das **erste** Element aus der Datenstruktur, die durch den Zeiger **pAnchor1** adressiert wird, gelöscht, dessen Datenelement **a** gleich dem Parameter value ist. Zurückgegeben werden soll der Wert **true** wenn das Löschen erfolgreich war, der Wert **false** sonst (ein Element mit dem Datenwert **value** war nicht in der entsprechenden Datenstruktur enthalten). Es ist nicht davon auszugehen, dass die Anzahl der Elemente bekannt ist.

Sonderfälle:

I. Leere Liste
II. nur ein Element in der Liste
II. erstes Element wird gelöscht.

```
bool delete_elem(int value, lelem *&  pAnchor1)
{
  lelem * pDel, * pSearch;
  if (pAnchor1 == nullptr) // Sonderfall I
    return false;
  if (pAnchor1->pNext == nullptr) {// Sonderfall II
       if (pAnchor1->a == value){
             delete pAnchor1;
             pAnchor1 = nullptr;
             return true;
          }
          else
            return false;
   }
    if (pAnchor1->a==value){   // Sonderfall III
        pDel = pAnchor1;
        pAnchor1 = pAnchor1->pNext;
        delete pDel;
        return true;
    }

      .....
```

```
   ....
   pSearch = pAnchor1;
   while ((pSearch->pNext != nullptr)&&(pSearch->pNext->a!=value))
              //nicht letztes Element und Löschdatum nicht gefunden
    {
       pSearch = pSearch->pNext;
    }
   if (pSearch->pNext == nullptr)
     return false;
   if (pSearch->pNext->a==value)   //eigentlich überflüssig ;-)
    {
       pDel=pSearch->pNext;
       pSearch->pNext = pDel->pNext;
       delete pDel;
       return true;
    }
}
```

d) Demonstrieren Sie die Verwendung der Funktionen **count_wert** und **delete_elem** derart, dass bezüglich eines vom Nutzer einzugebenden Wert nur **maximal** ein Element in der durch den Zeiger **pAnker** adressierten Datenstruktur verbleibt, dessen Datenwert **a** gleich dem vom Nutzer eingegebenen Wert ist.

d) Demonstrieren Sie die Verwendung der Funktionen **count_wert** und **delete_elem** derart, dass bezüglich eines vom Nutzer einzugebenden Wert nur **maximal** ein Element in der durch den Zeiger **pAnker** adressierten Datenstruktur verbleibt, dessen Datenwert **a** gleich dem vom Nutzer eingegebenen Wert ist.

```
...
/*
   int w;
   cin >> w;
   while (count_wert(w,pAnker) > 1)
      delete_elem(w,pAnker);
*/
  int w, anz;
  cin >> w;
  anz = count_wert(w,pAnker);
  for (int i =1; i < anz; i++)
     delete_elem(w,pAnker);
```

e) Ergänzen Sie das Programm um eine Funktion list2array mit dem Prototyp

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen);
```

mit folgender Funktionalität:
In der Funktion **list2array** wird eine zweidimensionale Matrix dynamisch erzeugt, die 5 Spalten besitzen soll. Die Anzahl der Zeilen ergibt sich aus der Elementzahl der einfach verketteten Liste, deren Anfang der Funktion als Parameter **pAnker1** übergeben wird. Die ermittelte Anzahl wird der rufenden Funktion über den Referenzparameter **anz_zeilen** mitgeteilt. Wenn der Funktion eine leere Liste übergeben wird, so soll die zurückgegebene Matrix (Ergebnis des Rufes der Funktion list2array) ebenfalls durch den NULL-Zeiger repräsentiert werden. Wenn der Funktion eine nichtleere Liste übergeben wurde, so gibt die Funktion die Adresse einer dynamisch erzeugten Matrix mit anz_zeilen und 5 Spalten zurück. In der letzten Zeile eventuell überzählige Elemente sollen mit dem Wert -1 belegt werden. Demonstrieren Sie den Ruf der Funktion list2array, indem Sie die Elemente der neu erzeugten Matrix in der Funktion main() in Matrixform (anz_zeilen, 5 Spalten) auf dem Bildschirm ausgeben. Dabei soll jedes Matrixelement mit einer Breite von 5 Zeichen ausgegeben werden, als Füllzeichen ist das Zeichen ‘.‘ (der Punkt) zu verwenden.

e) Ergänzen Sie das Programm um eine Funktion list2array mit dem Prototyp

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen);
```

mit folgender Funktionalität:
In der Funktion **list2array** wird eine zweidimensionale Matrix dynamisch erzeugt, die 5 Spalten besitzen soll. Die Anzahl der Zeilen ergibt sich aus der Elementzahl der einfach verketteten Liste, deren Anfang der Funktion als Parameter **pAnker1** übergeben wird. Die ermittelte Anzahl wird der rufenden Funktion über den Referenzparameter **anz_zeilen** mitgeteilt. Wenn der Funktion eine leere Liste übergeben wird, so soll die zurückgegebene Matrix (Ergebnis des Rufes der Funktion list2array) ebenfalls durch den NULL-Zeiger repräsentiert werden.

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen)
{
  anz_zeilen = 0;              //wohldefinierter Wert
  int anz_elemente = 0;        //Anzahl der Elemente

  if (pAnker1 == nullptr)
    return nullptr;
```

e) Ergänzen Sie das Programm um eine Funktion list2array mit dem Prototyp

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen);
```

mit folgender Funktionalität:
In der Funktion **list2array** wird eine zweidimensionale Matrix dynamisch erzeugt, die 5 Spalten besitzen soll. Die Anzahl der Zeilen ergibt sich aus der Elementzahl der einfach verketteten Liste, deren Anfang der Funktion als Parameter **pAnker1** übergeben wird. Die ermittelte Anzahl wird der rufenden Funktion über den Referenzparameter **anz_zeilen** mitgeteilt. … Wenn der Funktion eine nichtleere Liste übergeben wurde, so gibt die Funktion die Adresse einer dynamisch erzeugten Matrix mit anz_zeilen und 5 Spalten zurück. In der letzten Zeile eventuell überzählige Elemente sollen mit dem Wert -1 belegt werden.

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen)
{
  anz_zeilen = 0;              //wohldefinierter Wert
  int anz_elemente = 0;        //Anzahl der Elemente
  …
  //Elemente zaehlen
  lelem * pHelp = pAnker1;
  while (pHelp != nullptr)
  {
    anz_elemente++;
    pHelp = pHelp->pNext;
  }
```

e) Ergänzen Sie das Programm um eine Funktion list2array mit dem Prototyp

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen);
```

mit folgender Funktionalität:
… Wenn der Funktion eine nichtleere Liste übergeben wurde, so gibt die Funktion die Adresse einer dynamisch erzeugten Matrix mit anz_zeilen und 5 Spalten zurück. In der letzten Zeile eventuell überzählige Elemente sollen mit dem Wert -1 belegt werden.

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen)
{
  anz_zeilen = 0;              //wohldefinierter Wert
  int anz_elemente = 0;        //Anzahl der Elemente
  …
  //Elemente zaehlen
  //Matrix anlegen
  int ** matrix;
  int i,j;
  anz_zeilen = anz_elemente / 5;
  if (anz_elemente % 5 != 0)   //“unvollstaendig“ gefuellte letzte Zeile
    anz_zeilen++;
  matrix = new int*[anz_zeilen];
  for(i=0;i<anz_zeilen;i++)
    matrix[i] = new int[5];
  //letzte Zeile mit -1 fuellen
  for (j=0;j<5;j++)
    matrix[anz_zeilen-1][j] = -1;
```

e) Ergänzen Sie das Programm um eine Funktion list2array mit dem Prototyp

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen);
```

mit folgender Funktionalität:
… Wenn der Funktion eine nichtleere Liste übergeben wurde, so gibt die Funktion die Adresse einer dynamisch erzeugten Matrix mit anz_zeilen und 5 Spalten zurück. In der letzten Zeile eventuell überzählige Elemente sollen mit dem Wert -1 belegt werden.

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen)
{
  anz_zeilen = 0;                //wohldefinierter Wert
  int anz_elemente = 0;      //Anzahl der Elemente
  …
  //Elemente zaehlen
  //Matrix anlegen
  //… und mit Werten belegen
  pHelp = pAnker1;
  for (i=0;i<anz_zeilen;i++)
     for (j=0;j<5 && pHelp != nullptr;j++) {
        matrix[i][j] = pHelp->a;
        pHelp = pHelp->pNext;
    }
  return matrix;
}
```

e) Ergänzen Sie das Programm um eine Funktion list2array mit dem Prototyp

```
int ** list2array (lelem *  pAnker1, int & anz_zeilen);
```

mit folgender Funktionalität:

… Demonstrieren Sie den Ruf der Funktion list2array, indem Sie die Elemente der neu erzeugten Matrix in der Funktion main() in Matrixform (anz_zeilen, 5 Spalten) auf dem Bildschirm ausgeben. Dabei soll jedes Matrixelement mit einer Breite von 5 Zeichen ausgegeben werden, als Füllzeichen ist das Zeichen ‘.‘ (der Punkt) zu verwenden.

```
int main()
{
   ….
   int ** ma;
   int zeilen;
   ma = list2array(pAnker,zeilen);
   if (ma!=nullptr)
    {
      for (i=0;i<zeilen;i++)
        {
          for (j=0;j<5;j++)
            cout << setw(5) << setfill(‘.‘) << ma[i][j];
          cout << ‘\n‘;
        }
    }
```

g) Ergänzen Sie das Hauptprogramm derart, dass der gesamte dynamisch angeforderte Speicherplatz freigegeben wird. Es ist nicht davon auszugehen, dass die Anzahl der Elemente bekannt ist.

g) Ergänzen Sie das Hauptprogramm derart, dass der gesamte dynamisch angeforderte Speicherplatz freigegeben wird. Es ist nicht davon auszugehen, dass die Anzahl der Elemente bekannt ist.

```
....
lelem * pDel = pAnker;
while (pDel != nullptr)
 {
   pAnker = pAnker->pNext;
   delete pDel;
   pDel = pAnker;
  }
```

Musterlösung:

https://www.tu-chemnitz.de/informatik/friz/Grundl-Inf/Aufgaben/beispiele/musterklausur_einf_temp.cpp

Objektorientierte Lösung:

Klassendefinition:

```
class liste {
	private:
		lelem * pAnchor;
		int count_wert (int wert);
		bool delete_elem(int wert);
	public:
		liste();
		~liste();
		void insert(int wert);
		int validate(int wert);  //nur ein Element mit dem Datenwert wert bleibt übrig
		void print();
		int ** liste2matrix(int &anz_zeilen);
}
```

Objektorientierte Lösung:

Definition ausgewählter Methoden:

insert bearbeitet 2 Fälle:
1. neues erstes Element der Liste, wenn liste leer war
2. Einfügen eines neuen zweiten Elementes

```
void liste::insert(int wert)
{
  if (pAnchor == nullptr)
  {
        pAnchor = new lelem;
        pAnchor->pNext = nullptr;
        pAnchor->a = wert;
  }
  else
  {
    lelem * pTemp = pAnchor -> pNext;
    pAnchor -> pNext = new lelem;
    pAnchor -> pNext -> a = wert;
    pAnchor -> pNext -> pNext = pTemp;
  }
}
```

Objektorientierte Lösung:

Definition ausgewählter Methoden:

validate:
nur ein Element mit dem Datenwert wert bleibt übrig
Die Methode gibt die Anzahl der gelöschten Elemente zurück

```
int liste::validate(int wert)  //nur ein Element mit dem Datenwert wert bleibt übrig
{
   int anz = count_wert(wert);
   int geloescht = 0;
   while (anz>1)
   {
     delete_elem(wert);
     anz--;
     geloescht++;
   }

   return geloescht;

}
```

Objektorientierte Lösung:

Verwendung:

```
int main()
{
  liste A;
  int ** ma;
  int zeilen;
  int i,j,w;
  cin >> j;
  for ( i = 1; i < 11; i++)
    {
          cin >> j;
          A.insert(j%i);
    }

  A.print();

....
```

Objektorientierte Lösung:

Verwendung:

…

```
ma = A.liste2matrix(zeilen);
  if (ma!=nullptr)
    {
      for (i=0;i<zeilen;i++)
        {
          for (j=0;j<5;j++)
            cout << setw(5) << setfill('.') << ma[i][j];
          cout << '\n';
        }
    }
  //Speicherplatz ma freigeben

   for (i=0;i<zeilen;i++)
     delete []ma[i];
   delete []ma;
```

….

Objektorientierte Lösung:

Verwendung:

```
...

  cout << "Welcher Wert soll untersucht werden?\n";
  cin >> w;
  cout << "Anz " <<  A.validate(w) << endl;

  A.print();

  return 0;
}
```

Musterlösung:

https://www.tu-chemnitz.de/informatik/friz/Grundl-Inf/Aufgaben/beispiele/musterklausur_einf_temp_oop.cpp

Klausur:

28.7.2025 14:00 Uhr

Allgemeine Hinweise unter

https://www.tu-chemnitz.de/informatik/friz/Grundl-Inf/Hinweise/hinweise/allgemeine_hinweise_24072024.php

Ich wünsche Ihnen weiterhin viel Erfolg!