Name:

Dr.-Ing. Hartmut Helmke
Fachhochschule
Braunschweig/Wolfenbüttel
Fachbereich Informatik

Matrikelnummer:	*Punktzahl:*
Ergebnis:	

Freiversuch ☐ *F1* ☐ *F2* ☐ *F3* ☐

Klausur im WS 2008/09 :

☐ **Programmierkonzepte — Lösungen —**
☐ **Informatik III — Lösungen —**

Informatik B. Sc. ☐ *Technische Informatik B. Sc.* ☐

Hilfsmittel sind bis auf Computer, Handy etc. erlaubt !	Bitte Aufgabenblätter mit abgeben !
Austausch von Hilfsmitteln mit Kommilitonen ist **nicht** erlaubt !	

Die Lösungen sind auf separaten Blättern zu notieren.
Bitte notieren Sie auf **allen** Blättern Ihren Namen bzw. Ihre Matrikelnummer.

Verwenden Sie für jede Aufgabe ein separates eigenes Lösungsblatt.
Bei der grafischen Veranschaulichung der Heap- und Stackspeicherbelegung verwenden Sie bitte für jeden geforderten Zeitpunkt eine eigene Zeichnung (nicht alles in eine Zeichnung).
Gehen Sie davon aus, dass `double` 8 Bytes sowie `int` und Zeiger jeweils 4 Bytes im Speicher belegen.

Geplante Punktevergabe

Planen Sie pro Punkt etwas mehr als eine Minute Aufwand ein.

Punktziel	Im einzelnen	Pkte
A1: 23 (2+3+3+1+3+2+3+6) P.		
A2: 23 (2+2+6+4+2+3+4) P.		
A3: 11 (2+3+2+4) P.		
A4: 21 (7+4+6+4) P.		
A5: 22 (8+3+3+8) P.		
Sonderpunkte Labor		XXXX
Sonderpunkte Übungen		XXXX
Summe		

Bei den folgenden Aufgaben werden Sie meistens auf die hier gegebene Schnittstelle und Implementierung einer einfach verketteten Liste zurückgreifen.

Header-Datei der Klasse `Liste` (Ausschnitt):

```
// Knoten mit Wert und Nachfolger
class Node {
public:
  Node() {key=4; nxt=0;}
  Node(int k, Node* n) {key=k; nxt=n;}
  int getValue() const {return key;}
  const Node* getNxt() const {return nxt;}
private:
  int key;    // Wert des Listenelements
  Node* nxt; // Zeiger auf nächstes Element
  friend class Liste;
};
// Die Liste mit Anfang und Ende
class Liste{
public:
   Liste();
   ~Liste();
   void insBeg(int val); // am Anfang
   void insEnd(int val); // am Ende
   const Node* getStart() const;
   int getCount() const;
private:
   void clean(); // Speicher aufräumen
   Node* start;
   Node* end;
   int   count;
};
```

Zum Schreiben von Tests dürfen Sie auf die folgende Funktion zurückgreifen:

```
/** Überpruefung, ob jedes Element in
der Liste li im Array exp vorkommt und
li genau count Elemente enthält. */
bool checkRes(int exp[], const Liste& li,
                                 int count) {
   const Node* start = li.getStart();
   int i=0;
   while (start != NULL){
      if (exp[i++]!=start->getValue()){
         return false;
      }
      start=start->getNxt();
   }
   return li.getCount() == count;
}
```

Quellcode-Datei der Klasse `Liste` (Ausschnitt):

```
#include <cstdlib> // wg. NULL
#include "Liste.h"

/** Konstruktor */
Liste::Liste() {
   start = NULL;
   end = NULL;
   count = 0;
}
/** Destruktor */
Liste::~Liste() { clean(); }

void Liste::clean() {
   Node* lauf = start;
   while (lauf!=NULL) {
      Node* hlp = lauf->nxt;
      // Freigabe des Listenknotens
      delete lauf;
      lauf = hlp;
   }
   count = 0;
   start = NULL; end = NULL;
}
const Node* Liste::getStart() const{
   return start;
}
int Liste::getCount() const{
   return count;
}
```

```
/** Ein Knoten wird erzeugt und hier wird
val abgelegt. Der Knoten wird als
Listenanfang eingetragen.
Ist es der erste Knoten der Liste,
muss auch das Ende belegt werden. */
void Liste::insBeg(int val) {
   Node* oldSt = start;
   start = new Node;
   start->key = val;
   start->nxt = oldSt;
   if (end == NULL) {
      end = start;
   }
   count++;
}
```

Aufgabe 1 : Textfragen, Team

ca. 23 (2+3+3+1+3+2+3+6) Punkte

a.) Nennen Sie zwei Basistechniken von Extreme Programming.

b.) Nennen Sie **die zentrale** Basistechnik von Extreme Programming. Begünden Sie Ihre Antwort.

Lösung:
Testen, fast alle anderen Techniken bauen auf dem Testen auf, ohne automatische Test könnte z.B. *gemeinsame Verantwortung* nicht funktionieren.

c.) Sie wissen, dass Sie die Implementierung einer aktuell noch nicht benötigten Funktion heute 500 Euro kosten wird. Wenn Sie die gleiche Funktion in drei Monaten implementieren, müssen Sie einiges an Ihrem Design ändern und die Implementierung wird dann wahrscheinlich 2.000 Euro kosten.
Warum könnte es trotzdem ratsam sein, die Funktion erst später zu implementieren?

Lösung:
Wenn Sie heute noch nicht mit Sicherheit wissen, dass Sie die Funktion mit Sicherheit brauchen, sondern dieses nur z.B. zu 20% gesichert ist, dann wäre die Implementierung unnötiger Aufwand. Der Erwartungswert der Kosten für die Implementierung heute wäre 1.500 Euro für die Implementierung in drei Monaten allerdings nur (20% * 2000 + 80% *0) 400 Euro.

d.) In der Planung von Softwareprojekten spielen vier Variablen eine Rolle: Kosten, Qualität und Zeit. Wie heißt die vierte Variable?

Lösung:
Umfang

e.) Klären Sie mit 2 - 3 Sätzen, welche Auswirkungen eine Verdopplung der Kosten (d.h. des zur Verfügung stehenden Projektbudgets) auf die Variable Zeit hat.

Hinweis: Hier gibt es nicht die eine richtige Antwort.

Lösung:
Wird das Budget frühzeitig erhöht, kann mehr Personal eingesetzt werden. Dies wird in der Regel die Projektlaufzeit verkürzen können, aber in der Regel darf nicht auf eine Halbierung gehofft werden.

f.) Erklären Sie mit einem Satz, was unter der *äußeren* Qualität der Software zu verstehen ist.

Lösung:
Die äußere Qualität bekommt der Kunde zu sehen (z.B. Aussehen der Benutzerschnittstelle, Performanz, Fehler).

g.) Sie haben Ihre Iteration in die Arbeitspakete A1, A2, A3 usw. eingeteilt. Nun wollen Sie den Aufwand der einzelnen Arbeitspakete (in idealen Tagen) zusammen mit Ihrem Team schätzen. Wie könnten Sie hierbei vorgehen?

Hinweis: Hier gibt es nicht die eine richtige Antwort.

Lösung:
Jeder aus dem Team kann eine Schätzung abgeben. Liegen die Schätzungen zu weit auseinander, müssen die Extremwerte Ihre Schätzungen begründen und es wird erneut geschätzt.

h.) Sie haben den Umfang der einzelnen Aufgaben (in idealen Tagen) geschätzt, deren Erledigung sich der Kunde für die nächste Iteration (die achte) gewünscht hat. Damit wissen Sie natürlich noch nicht, wie viel Sie wirklich erledigen werden. Wie gehen Sie vor, um dem Kunden einigermaßen verlässlich zu sagen, was Sie leisten können und was nicht? Geben Sie auch ein Zahlenbeispiel zur Veranschaulichung Ihrer Überlegungen an.

Lösung:
Die Aufgaben sind nach Prioritäten aus Kundensicht zu ordnen. Mit der Technik des Wetters von gestern ist der Load Faktor aus den Schätzungen der letzten Iterationen und den erreichten Ergebnissen ermittelt. Es wird nun das Produkt aus Load Faktor und Anzahl der zur Verfügung stehenden idealen Tage für diese Iteration bestimmt. Damit ergibt sich, was in dieser Iteration möglich ist:
Beispiel:
Load Faktor: 0,6; 60 ideale Tage habe diese Iteration. Damit können wir dem Kunden Aufgaben im Umfang von 0,6 * 60 = 36 idealen Tagen versprechen. Wir versprechen ihm somit die Erledigung der ersten nach Prioritäten geordneten Aufgaben bis deren geschätzte Summe 36 Tage übersteigt.

Aufgabe 2 : Stack-Heapspeicher

ca. 23 (2+2+6+4+2+3+4) Punkte

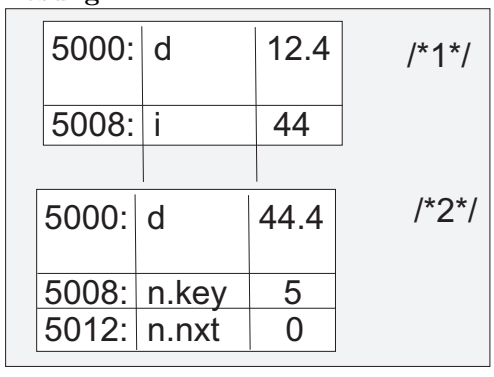
a.) Veranschaulichen Sie grafisch die Stack-Speicherbelegung des folgenden Funktionsaufrufs zum Zeitpunkt /* 1 */.

```
void heapStack1() {
   double d = 12.4;
   int i=44;
   /* 1 */
}
```

b.) Veranschaulichen Sie grafisch die Stack-Speicherbelegung des folgenden Funktionsaufrufs zum Zeitpunkt /* 2 */.

```
void heapStack2() {
   double d=44.4;
   Node n(5, NULL);
   /* 2 */
}
```

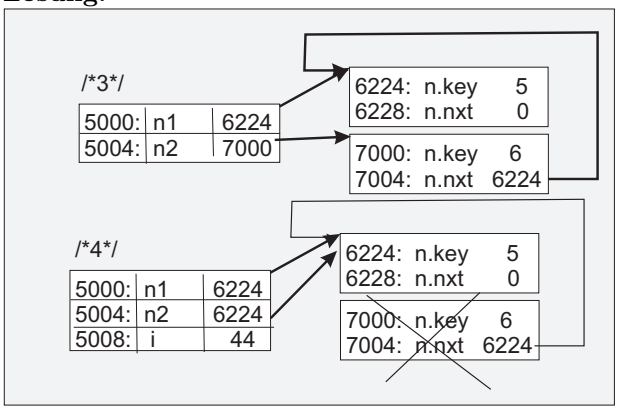
Lösung:



c.) Veranschaulichen Sie grafisch die Stack- und Heap-Speicherbelegung des folgenden Funktionsaufrufs zu den Zeitpunkten /* 3 */ und /* 4 */.

```
void heapStack3() {
   Node* n1 = new Node(5, NULL);
   Node* n2 = new Node(6, n1);
   /*3*/
   delete n2;
   n2 = n1;
   int i=44;
   /* 4 */
}
```

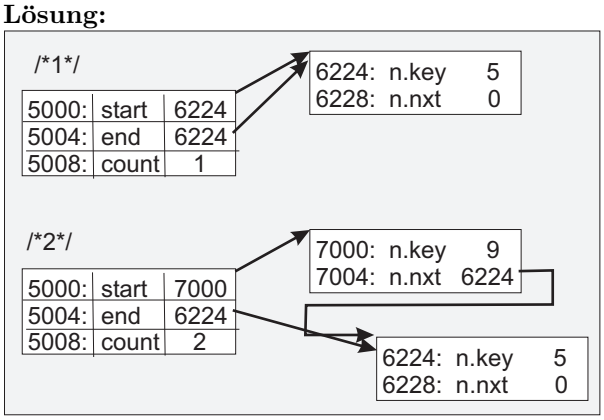
Lösung:



d.) Veranschaulichen Sie grafisch die Stack- und Heap-Speicherbelegung des folgenden Funktionsaufrufs zum Zeitpunkt /* 2 */ (Das ist nach der Anweisung der betreffenden Zeile).

```
void begAnw(){
   Liste lis;
   lis.insBeg(5);
   lis.insBeg(9); /* 2 */
}
```

Lösung:



e.) Passen Sie die obige Funktion begAnw an, sodass statt der Klasse `Liste` die STL-Schablonenklasse `list` verwendet wird. Die Funktionalität (2 Elemente werden in die Liste eingetragen) soll die gleiche bleiben.
Hinweis: Die STL-Schablonenklasse `list` besitzt u.a. die Methoden `push_back` und `push_front`.
Lösung:

```
void begAnw(){
   list<int> lis;
   lis.push_front(5);
   lis.push_front(9);
}
```

f.) Spezifizieren Sie für die Methode `Liste::insBeg` einen geeigneten Test (Spezifizieren bedeutet: Beschreibung mit deutschen Worten, ohne Code).
Hinweis: Sie können und sollten hierzu auf die Funktionalität der Funktion `checkRes` zurückgreifen (siehe Seite 2 der Aufgabenblätter).
g.) Implementieren Sie nun den zuvor spezifizierten Test für die Methode `Liste::insBeg` in C++.
Lösung:

```
/** Erstellung einer Liste aus 3, 5, 8.
Einfuegen von 1 am Anfang und Pruefung,
ob sich Liste 1, 3, 5, 8 ergibt */
bool testInsBeg2(){
   Liste lis;

   // Aufbau der Ausgangsliste
   lis.insBeg(8);
   lis.insBeg(5);
   lis.insBeg(3);
   int exp1[] = {3, 5, 8};
   bool res = checkRes(exp1, lis, 3);

   lis.insBeg(1);
   int exp2[] = {1, 3, 5, 8};
   res =checkRes(exp2, lis, 4) && res
      && lis.getCount() == 4;
   return res;
}
```

Aufgabe 3 : Listen-Klassen, Textfragen

ca. 11 (2+3+2+4) Punkte
a.) Geben Sie für die Header-Datei der Listenklasse einen passenden Include-Wächter an.
Lösung:

```
#ifndef __LISTE_HEADER
#define __LISTE_HEADER
...
#endif
```

b.) Erklären Sie, warum das Schlüsselwort `const` in der Header-Datei der Klasse `Node` an den folgenden Stellen **dreimal** erforderlich ist, d.h. drei Erklärungen sind gewünscht.

```
int getValue() const {return key;}
const Node* getNxt() const {return nxt;}
```

Lösung:
getValue() const:
Die Methode verändert kein Attribut der Klasse `Node`.
getNxt() const:
Die Methode verändert kein Attribut der Klasse `Node`.
const Node* getNxt():
Es wird ein Zeiger auf ein privates Attribut zurückgeliefert. Über diesen Zeiger könnte das Attribut (`nxt`) von außen verändert werden. Dies ist bei einer konstanten Methode aber verboten. Deshalb muss ein konstanter Zeiger zurückgeliefert werden, über den Inhalte nicht veränderbar sind.

c.) Welche Methoden müssen der Klasse `Node` noch hinzugefügt werden, damit sie eine minimale Standardschnittstelle besitzt?
d.) Implementieren Sie diese Methoden. Sie dürfen sie inline (d.h. direkt in der Header-Datei) implementieren.
Lösung:

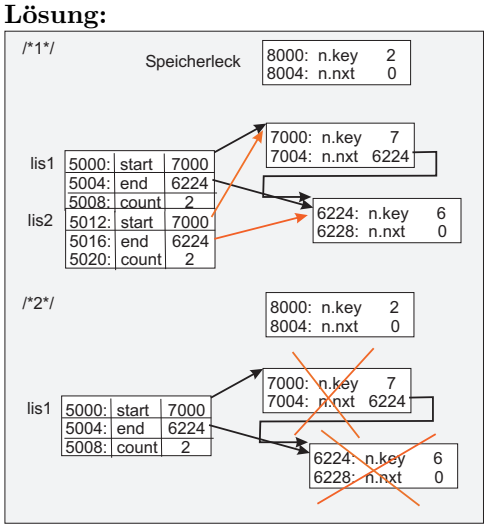
```
Destruktor: ~Node() {/*muss nichts machen*/}
Kopierkonstruktor:
Node(const Node& n2) {
  key = n2.key; nxt = n2.nxt;
}
Zuweisungsoperator:
Node& operator=(const Node& n2) {
  key = n2.key; nxt = n2.nxt; return *this;
}
```

Aufgabe 4 : Tiefes und flaches Kopieren

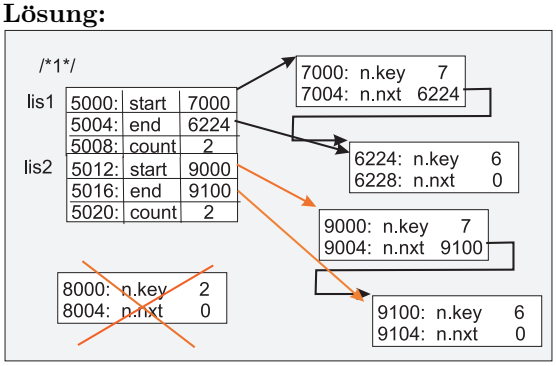
ca. 21 (7+4+6+4) Punkte
a.) Veranschaulichen Sie grafisch die Stack- und Heap-Speicherbelegung des folgenden Funktionsaufrufs zu den Zeitpunkten /*1*/ und /*2*/. Eine Zeichnung reicht, wenn aus ihr auch die Speicherbelegung zum Zeitpunkt /* 1 */ erkennbar ist.

```
void assignOperator() {
    Liste lis1;
    lis1.insBeg(7);
    lis1.insBeg(6);
    if (true) {
        Liste lis2;
        lis2.insBeg(2);
        lis2 = lis1; /* 1 */
    }
    /* 2 */
}
```

Lösung:



b.) Beschreiben Sie nun, warum obige Funktion beim Verlassen vermutlich zu einem Programmabsturz bzw. zu einem undefinierten Verhalten führen wird (Verwenden Sie dazu Ihre zuvor erstellte Stack- und Heapspeicherbelegung).
Hinweis: Welche Zeile in der Methode `Liste::clean` müsste auskommentiert werden, damit es zumindest zu keinem Absturz oder undefinierten Verhalten in diesem *Mini-Programmchen* kommt?
Lösung:
Der Speicherplatz der Listenknoten auf dem Heap, die zu lis1 gehören, sind bereits durch `delete lauf;` in der clean-Methode für `lis2` freigegeben worden. Beim Verlassen der Funktion `assignOperator` wird abermals durch Aufruf des Destruktors von `Liste` versucht, diesen Speicherplatz freizugeben, d.h. der gleiche Heap-Speicherplatz wird doppelt freigegeben.
c.) Wie würde die Speicherbelegung vom Heap- und Stackspeicher aussehen, wenn ein korrekt implementierter Zuweisungsoperator für die Klasse vorhanden wäre (Zeitpunkt /* 1 */ in der Funktion assignOperator)?
Achtung: Sie sollen hier nur die Speicherbelegung veranschaulichen.
Lösung:



d.) Implementieren Sie auch einen geeigneten Test für den Zuweisungsoperator.
Hinweis: Sie können und sollten hierzu auf die Funktionalität der Funktion `checkRes` zurückgreifen (siehe Seite 2 der Aufgabenblätter).
Lösung:

```
/** Eine Liste wird per
Zuweisungsoperator zugewiesen. Es wird geprüft,
ob die beiden Listen dann den Erwartungen
entsprechen und gleichen Inhalt haben.
*/
bool testAssign() {
   int exp1[] = {6, 7};
   Liste lis1;
   lis1.insBeg(7);
   lis1.insBeg(6);
   bool res = checkRes(exp1, lis1, 2);
   if (true) {
      Liste lis2;
      lis2 = lis1;
      res = checkRes(exp1, lis2, 2) && res;
   }
   res = checkRes(exp1, lis1, 2) && res;
   return res;
}
```

Aufgabe 5 : Templates

ca. 22 (8+3+3+8) Punkte
a.) Erweitern Sie die Klasse `Liste` zu einer Schablonenklasse, sodass nicht nur der Typ `int` in der

Klasse gespeichert werden kann. Sie dürfen hierzu direkt im folgenden Code der Header-Datei ergänzen (Quellcode-Datei heute nicht erforderlich).

```
template <typename T>
class Node {
public:
  Node(): key(), nxt(0) {}
  /* 1    .....    */









private:
  template <typename T> friend class Liste;
  /*2    .....    */









};

template <typename T>
class Liste{
public:
    Liste();
    Liste(const Liste& li2);
    Liste<T>& operator=(const Liste<T>& li2);
    ~Liste();
    /* 3    .....    */









private:
    void clean(); // Speicher aufräumen
    void copy(const Liste& li2); // li2 kopieren
    /* 4    .....    */









};
```

Lösung:

```
// Knoten mit Wert und Nachfolger
template <typename T>
class Node {
public:
  Node(): key(), nxt(0) {}
  Node(const T& k, Node* n) {key=k; nxt=n;}
  T getValue() const {return key;}
  const Node* getNxt() const {return nxt;}
private:
  T key;   // Wert des Listenelements
  Node* nxt; // Zeiger auf nächstes Element
  template <typename T> friend class Liste;
};
```

```
// Die Liste mit Anfang und Ende
template <typename T>
class Liste{
public:
    Liste();
    Liste(const Liste& li2);
    Liste<T>& operator=(const Liste<T>& li2);
    ~Liste();
    void insBeg(const T& val); // am Anfang
    void insEnd(const T& val); // am Ende
    const Node<T>* getStart() const;
    int getCount() const;
private:
    void clean(); // Speicher aufräumen
    void copy(const Liste& li2); // li2 kopieren
    Node<T>* start;
    Node<T>* end;
    int    count;
};

#include "Liste.cpp"
```

b.) Welche Voraussetzungen muss ein Datentyp `T` erfüllen, damit Elemente von ihm in dieser Schablonenklasse `Liste<>` gespeichert werden können?

Lösung:
Sie muss über eine minimale Standardschnittstelle verfügen, also über
Standardkonstruktor:
wg. `Node(): key(), nxt(0) {}`
über Destruktor
Methode `Liste<T>::clean`: `delete lauf;`
über Zuweisungsoperator
wg. `Node(const T& k, Node* n) {key=k; ...}`
über Kopierkonstruktor
wg. `T getValue() const {return key;}`

c.) Implementieren Sie eine kurze Anwendungsfunktion für die Schablonenklasse Liste, in der `double`-Werte in einer Liste gespeichert werden.

Lösung:

```
/** Eine Liste mit double-Werten wird erstellt.
Es werden zwei Werte in die double-Liste
eingetragen.
*/
void doubleApplication() {
    Liste<double> lis;
    lis.insBeg(3);
    lis.insEnd(29);
}
```

d.) Implementieren Sie einen Test für die Methoden `Liste::insBeg` , sodass Listen verwendet werden, die selbst als Elemente wieder Listen vom Typ `Liste<int>` enthalten.
Erweitern Sie hierzu den im folgenden angegebenen Codeausschnitt aus der Funktion `testInsBegListList` in geeigneter Weise an den Stellen /*1*/ (Beschreibung, des *Was* und *wie*) und /*2*/.

```
/* 1  Beschreibung */
bool  testInsBegListList (){
   /* 2
   zu Prüfendes aufbauen */

   /* Prüfen, ob big als erstes
   Element die Liste <3, 5, 8> enthält.*/
   int exp[] = {3, 5, 8};
   return  checkRes(exp,
      big.getStart()->getValue(), 3);
}
```

Lösung:

```
/** Anlegen einer leeren Liste vom Typ
Liste<Liste<int> > mit Namen big.
In diese Liste wird eine Liste eingetragen,
die selbst die int-Werte 3, 5, 8 enthält.
Anschließend wird aus big das erste
Element gelesen. Es wird geprüft, ob
diese gelesenen Liste die Werte
3, 5, 8 enthält.
*/
bool  testInsBegListList (){
    Liste<int> hlp;
    hlp.insEnd(3);
    hlp.insEnd(5);
    hlp.insEnd(8);

    Liste<Liste<int> > big;
    big.insBeg(hlp);

    /* Prüfen, ob big als erstes
    Element die Liste <3,5,8> enthält.*/
    int exp[] = {3, 5, 8};
    return  checkRes(exp,
       big.getStart()->getValue(), 3);
}
```