Name:

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

Matrikelnummer:	*Punktzahl:*
Ergebnis:	

Freiversuch ☐ *F1* ☐ *F2* ☐ *F3* ☐

Klausur im WS 2008/09 :

☐ ☐ **Programmierkonzepte**
Informatik III

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	Sonderpunkte	erreicht
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.

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?

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

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.

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

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.

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.

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 */
}
```

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 */
}
```

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 */
}
```

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`.

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++.

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.

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;}
```

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.

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 */
}
```

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-Prográmmchen* kommt?

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.

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).

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    .....    */




};
```

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

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

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);
}
```