

Name:

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

Matrikelnummer:		Punktzahl:	
Ergebnis:			
Freiversuch	<input type="checkbox"/>	F1	<input type="checkbox"/>
		F2	<input type="checkbox"/>
		F3	<input type="checkbox"/>

Klausur im SS 2008 :


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

Medieninformatik   
Informatik B. Sc.

Praktische Informatik

Technische Informatik   
Technische Informatik B. Sc.

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

Die Lösungen sind auf separaten Blättern zu notieren.

Bitte notieren Sie auf **allen** Blättern Ihren Namen bzw. Ihre Matrikelnummer.

Auf eine absolut korrekte Anzahl der Blanks und Zeilenumbrüche braucht bei der Ausgabe nicht geachtet zu werden. Dafür werden keine Punkte abgezogen.

**Hinweis:** In den folgenden Programmen wird manchmal die globale Variable *datei* verwendet. Hierfür kann der Einfachheit halber die Variable *cout* angenommen werden. Die Variable *datei* diene lediglich bei der Klausurerstellung dem Zweck der Ausgabeumlenkung.

### Geplante Punktevergabe

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

Punktziel	Im einzelnen	Pkte
A1: 24 <small>(3+8+5+8)</small> P.		
A2: 20 <small>(6+ 1,5+1,5 +4+2+5)</small> P.		
A3: 15 <small>(2+2+4+3+2+2)</small> P.		
A4: 41 <small>(6+6+4+2+2+2 + 19)</small> P.		
Zu: 10 P.		
Summe 100 (+10) P.		

Den Code der im Folgenden in einigen Aufgaben verwendeten Klasse **Stack** finden Sie hier. Dies erleichtert Ihnen das Umblättern.

Am besten greifen Sie hier auf Ihr Wissen über die **Matrix**-Klasse aus der Vorlesung zurück.

#### Header-Datei der Klasse **Stack**

```
// einfache generische Stack-Klasse, wobei auf
// Fehlerüberprüfung verzichtet wird.
template <class T>
class Stack {
public:
    Stack (int siz=2);
    ~Stack ();
    Stack (const Stack& s);
    Stack& operator= (const Stack& s);
    void Push (T x);
    T Pop ();
    bool Empty() const;
    int Count () const;

private:
    // Attribute
    T* data; // Zeiger auf Datenbereich
    int n; // aktuelle Anzahl Elemente im Stack
    int size; // maximale Anzahl Elemente im Stack
};
```

#### Quellcode-Datei der Klasse **Stack**

```
template <class T>
Stack<T>::Stack (int siz) {
    datei << "+Stack " << siz << "\n";
    size=siz;
    data = new T[size];
    n = 0;
}

template <class T>
Stack<T>::~Stack () {
    datei << "-Stack " << size << "\n";
    delete [] data;
}

// x wird in den Stack eingetragen
template <class T>
void Stack<T>::Push (T x) {
    datei << " Push ";
    data[n++] = x;
}

// oberste Element vom Stack wird geliefert
template <class T>
T Stack<T>::Pop () {
    datei << " Pop ";
    return data[--n];
}

template <class TYP>
Stack<TYP>::Stack (const Stack<TYP>& s) {
    datei << "+StackCopy " << s.size << "\n";
    size=s.size;
    data = new TYP [size];
    n = s.n;
    for (int i=0; i<n; i++) {
        data[i] = s.data[i];
    }
}

template <class T>
Stack<T>& Stack<T>::operator= (const Stack<T>& s) {
    datei << " operator= " << s.size << "\n";
    if (this != &s) {
        delete [] data;
        size = s.size;
        data = new T[size];
        n = s.n;
        for (int i=0; i<n; i++) {
            data[i] = s.data[i];
        }
    }
}
```

**Aufgabe 1 : Testen**

ca. 24 (3+8+5+8) Punkte

Entwerfen Sie eine Funktion `minMax`, die aus zwei Eingaben ihr Produkt (Multiplikation), ihre Summe (Plus) und ihre Differenz (Minus) ermittelt und anschließend den kleinsten **und** den größten dieser drei Werte zurückliefert. Achtung: Die beiden Ergebnisse Maximum und Minimum sind gleichzeitig mit einem Aufruf zu liefern.

Gehen Sie hierbei in den folgenden Schritten vor:

- Legen Sie die Schnittstelle von `minMax` fest, d.h. die Deklaration der Funktion.
- Spezifizieren Sie (z.B. mit Worten, noch **nicht** als C++-Code) mindestens drei wirklich verschiedene Tests.

**Lösung:**

Sonderpunkt, wenn berücksichtigt ist, dass auch zwei gleiche Werte herauskommen können.

- Implementieren Sie nun einen der Tests als C++-Funktion.

**Lösung:**

Sonderpunkt, wenn Konstante bei `==` links steht.

- Beschreiben Sie nun (mit deutschen Worten) einen möglichen Algorithmus der Funktion `minMax` (Hoffentlich fällt Ihnen eine bessere Lösung ein als die Verwendung von vielen `if`-Anweisungen; Stichwort *Sortieren*).

Die Funktion selbst sollen Sie **nicht** implementieren, das ist die Aufgabe Ihrer Angestellten.

**Lösung:**

```
void minMax(int a, int b, int& amin, int& amax);

bool prodSumDiffTest(){
    /* diff und sum werden geliefert als min, max
    0, -1 --> sum: -1, diff +1, prod: 0
    min=-1, max =+1
    */
    int emin; int emax;
    minMax(0, -1, emin, emax);
    bool retValue = (-1 == emin) &&
        (1 == emax);
    /* diff und prod werden geliefert als min, max
    2, 4 --> sum: 6, diff -2, prod: 8
    min=-1, max =+1
    */
    minMax(2, 4, emin, emax);
    retValue = (-2 == emin)
        && (8 == emax) && retValue;
    /* sum und prod werden geliefert als min, max
    -2, -4 --> sum: -6, diff 2, prod: 8
    min=-6, max =+8
    */
    minMax(-2, -4, emin, emax);
    retValue = (-6 == emin)
        && (8 == emax) && retValue;
    return retValue;
}

/* Summe, Differenz und Produkt werden in ein Array
geschrieben .
Dies wird mit dem STL-Algorithmus sort sortiert und
anschließend ist das erste Element das Minimum und
das dritte des sortierten Arrays ist das Maximum */
void minMax(int a, int b, int& amin, int& amax)
{
    int arr[3] = {a+b, a-b, a*b};
    sort(&arr[0], &arr[3]);
    amin=arr[0];
    amax=arr[2];
}
```

**Aufgabe 2 : Textfragen, C++**

ca. 20 (6+ 1,5+1,5 +4+2+5) Punkte

a.) Welche Methoden etc. der Klasse `Stack` (siehe Anhang) gehören zur minimalen Standard-Schnittstelle? Geben Sie hierzu die entsprechenden Codezeilen der Header-Datei an.

**Lösung:**

Default-Konstruktor `Stack(int siz=2);`

Destruktor: `Stack ();`

Kopier-Konstruktor: `Stack (const Stack& s);`

Zuweisungsoperator:

`Stack& operator= (const Stack& s);`

b.) Dürfte das Argument des Kopier-Konstruktors von `Stack` auch als Werteparameter übergeben werden?

**Lösung:**

Nein, denn dann würde sich der Kopier-Konstruktor unendlich oft selbst aufrufen.

c.) Dürfte das Argument des Zuweisungsoperators von

Stack auch als Werteparameter übergeben werden?

**Lösung:**

Dies ist erlaubt. Es würde dann allerdings jeweils zunächst eine Kopie durch den Kopierkonstruktor angelegt, was zumindest ineffizient wäre.

d.) Welche Unterschiede gibt es zwischen Java und C++ bei der Belegung des Heap- und Stackspeichers mit Variablen?

**Lösung:**

Java legt benutzerdefinierte Datentypen grundsätzlich auf dem Heap ab und die Standardtypen auf dem Programmstack, C++ legt das der Benutzer selbst fest. Standardtypen sind im Heap und auf Programmstack möglich. Entsprechendes gilt für die benutzerdefinierten Typen.

Zusatzpunkt für: In C++ muss der Benutzer selbst für die Freigabe des Heapspeichers sorgen, in Java übernimmt diese Aufgabe der Garbage Collector.

e.) Warum wurde das Schlüsselwort `const` bei der Methode `Empty` verwendet und bei der Methode `Push` von der Klasse `Stack` nicht?

**Lösung:**

`Empty` verändert keine Attribute der Klasse, `Push` dagegen schon.

f.) Implementieren Sie den C++-Code für einen Test der Methode `Count`

**Lösung:**

```
bool testCount() {
    Stack<int> s1(4);
    s1.Push(1); s1.Push(2);
    return 2 == s1.Count();
}
bool testEmpty() {
    Stack<int> s1(4);
    Stack<int> s2(4);
    s2.Push(3);
    return s1.Empty() && (! s2.Empty());
}
```

**Aufgabe 3 : Textfragen, Team**

ca. 15 (2+2+4+3+2+2) Punkte

a.) Nennen Sie zwei Basistechniken von Extreme Programming.

b.) Nennen Sie die zentrale Basistechnik von Extreme Programming. Begründen 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 1.000 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 30% gesichert ist, dann wäre die Implementierung unnötiger Aufwand. Der Erwartungswert der Kosten für die Implementierung heute wäre 1.000 Euro für die Implementierung in drei Monaten allerdings nur  $(30\% * 2000 + 70\% * 0) = 600$  Euro.

d.) Erklären Sie an zwei Beispielen (2 genügen), woher das Wort **extreme** in dem Namen *Extreme Programming* stammt.

**Lösung:**

- Von Code-Reviews ist bekannt, dass sie ein sehr positives Kosten-Nutzen-Verhältnis aufweisen. Warum also nicht durch paarweises Programmieren, ständig Reviews durchführen?
- Wenn Testen zur Qualitätsverbesserung der Software beiträgt, warum soll dann nicht jeder Entwickler (durch Unit-Tests) und der Kunde (durch Akzeptanztests) ständig testen?
- Ein gutes Design der Software ist wichtig. Deshalb sollte jeder Entwickler durch Refactoring täglich zur Designverbesserung beitragen.
- Da einfache Systeme besser zu warten sind, sollte immer das einfachste Design gewählt werden, dass die gegenwärtigen Anforderungen erfüllt.
- Da Integration und Integrationstest wichtig sind, wird mehrmals am Tag integriert und getestet.
- Schnelle Iterationen sind anzustreben. Deshalb sollte eine Iteration sehr, sehr kurz sein – Stunden oder Tage, nicht Wochen oder gar Monate.

e.) Erklären Sie den Begriff des Truck Faktors.

**Lösung:**

Der Truck Faktor ist die Wahrscheinlichkeit, dass ein Projekt scheitert, wenn ein entscheidendes Team-Mitglied vom Truck überfahren wird.

f.) Welche zwei Basistechniken von Extreme Programming tragen hauptsächlich zur Reduktion des Truck Faktors bei?

**Lösung:**

Zwei Leute ein Bildschirm (auch *Programmieren in Paaren* genannt), gemeinsame Verantwortung und zum Teil auch: Testen, einfaches Design, Refactoring

**Aufgabe 4 : Stack-Heapspeicher**

ca. 41 (6+6+4+2+2+2 + 19) Punkte

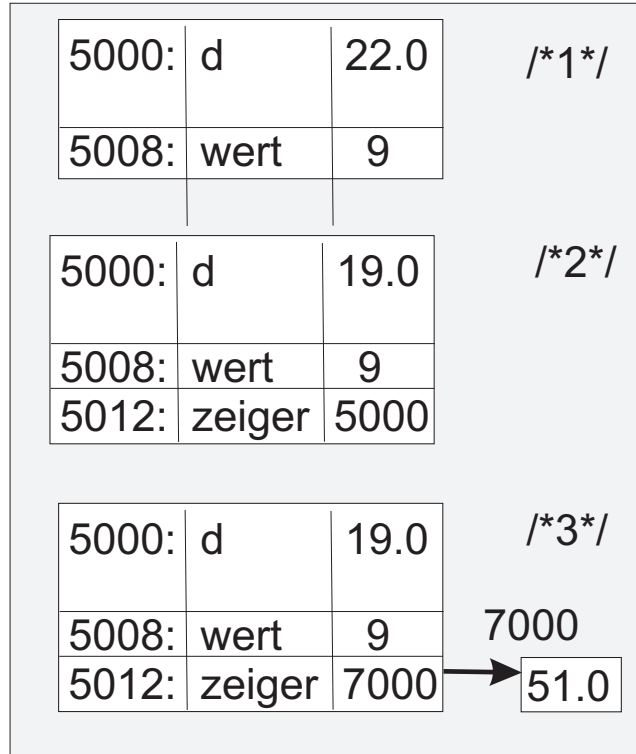
Bei der grafischen Veranschaulichung der Heap- und Stackspeicherbelegung verwenden Sie bitte für jeden Zeitpunkt eine eigene Zeichnung; nicht alles in eine Zeichnung.

a.) Veranschaulichen Sie grafisch die Stack- und Heapspeicherbelegung des folgenden Funktionsaufrufs zu den Zeitpunkten `/* 1 */`, `/* 2 */` und `/* 3 */`.

Beachten Sie, dass `double` 8 Byte belegt.

```
void funk1() {
    double d=22;
    int wert = 9; /* 1 */
    double* zeiger = &d;
    *zeiger = 19; /* 2 */
    zeiger = new double;
    *zeiger = 51; /* 3 */
}
```

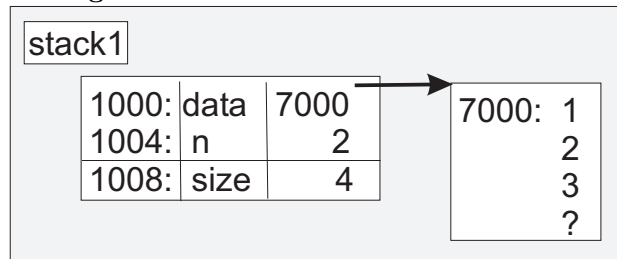
**Lösung:**



b.) Veranschaulichen Sie ganz entsprechend die Stack- und Heap-Speicherbelegung des folgenden Programmfragments nach Ausführung des mit /\* 1 \*/ gekennzeichneten Programmschrittes.

```
void funk2() {
    Stack<int> stack1(4);
    stack1.Push(1);
    stack1.Push(2);
    stack1.Push(3);
    stack1.Pop(); /* 1 */
}
```

**Lösung:**



c.) Zu welcher Ausgabe führt obiger Aufruf der Funktion funk2 in die Datei datei? **Hinweis:** In der ersten Zeile der Methoden führt jeweils der Aufruf von datei << ... zu einer Dateiausgabe. **Lösung:**

```
funk2
+Stack 4
Push Push Push Pop -Stack 4
```

```
void funk3() {
    Stack<int> stack1(4);
    stack1.Push(1);
    stack1.Push(2);
    stack1.Push(3);

    Stack<Stack<int> > big(3); /* 1 */
    datei << "###Vor Push\n";
    big.Push(stack1);
    stack1.Push(stack1.Pop() + 1);
    stack1.Pop();
    big.Push(stack1); /* 2 */
    datei << "###Nach Push\n";
    // big.Pop(); // Zusatzfrage
}
```

d.) In welcher Zeile der Funktion funk3 wird der Defaultkonstruktor aufgerufen?

**Lösung:**

Stack<Stack<int> > big(3); /\* 1 \*/  
Der Zeiger des Attributs data verweist auf ein Array mit drei Elementen vom Typ Stack<int>. Um diese Arrayelemente zu erzeugen, wird der Default-Konstruktor von Stack<int> aufgerufen.

e.) In welchen Zeilen der Funktion funk3 wird der Kopierkonstruktor aufgerufen?

**Lösung:**

Die beiden Aufrufe der Methode Push für die Instanz big führen den Kopierkonstruktor aus.

f.) Warum führt der auskommentierte Aufruf big.Pop(); zum Aufruf des Kopierkonstruktors von der Klasse Stack, wenn die Zeile wieder aktiviert wird?

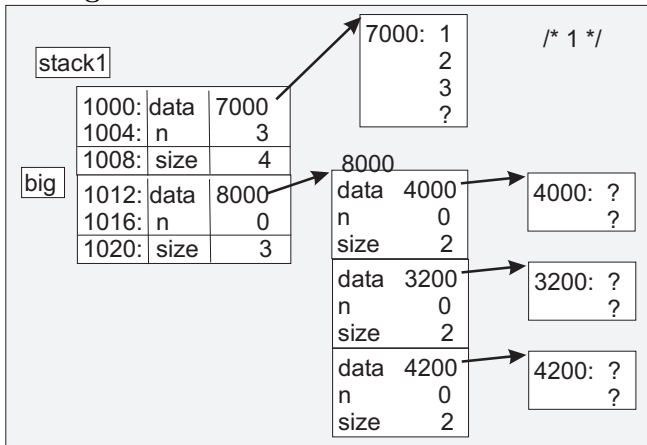
**Lösung:**

+StackCopy  
-Stack 4  
Pop liefert einen Wert vom Typ Stack<int> zurück, d.h. es wird eine Kopie angelegt, die anschließend wieder zerstört wird. Die vollständige Ausgabe wäre somit:

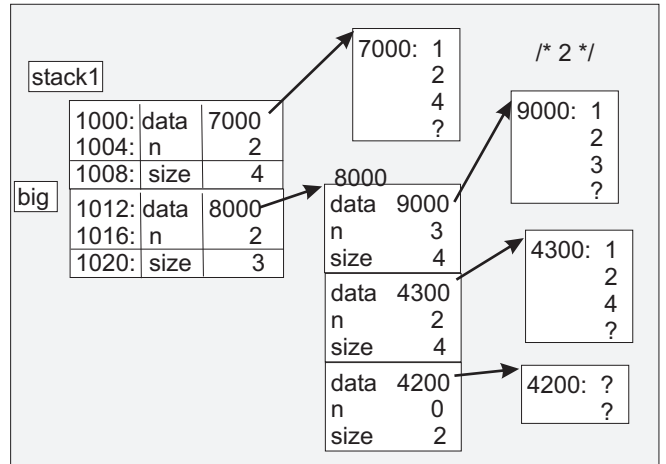
```
funk3 mit Pop
+Stack 4
  Push Push Push +Stack 3
+Stack 2
+Stack 2
+Stack 2
###Vor Push
+StackCopy 4
  Push operator= 4
-Stack 4
  Pop Push Pop +StackCopy 4
  Push operator= 4
-Stack 4
###Nach Push
  Pop +StackCopy 4
-Stack 4
-Stack 3
-Stack 2
-Stack 4
-Stack 4
-Stack 4
-Stack 4
+Stack 4
  Push Push Push Push Pop Pop Pop -Stack 4
```

g.) Veranschaulichen Sie die Stack- und Heap-Speicherbelegung des Aufrufs der Funktion `funk3` nach Ausführung der mit `/* 1*/` und `/* 2 */` gekennzeichneten Programmschritte.

**Lösung:**



**Lösung:**



**Aufgabe 5 : Zusatzaufgabe**

ca. 10 Punkte

Für diese Aufgabe gibt es Zusatzpunkte.

a.) Zu welcher Ausgabe führt der Aufruf der Funktion `funk3` in die Datei `datei`? Beachten Sie insbesondere den Aufruf des Default-Konstruktors und den Aufruf des Kopierkonstruktors.

```
funk3
+Stack 4
  Push Push Push +Stack 3
+Stack 2
+Stack 2
+Stack 2
###Vor Push
+StackCopy 4
  Push operator= 4
-Stack 4
  Pop Push Pop +StackCopy 4
  Push operator= 4
-Stack 4
###Nach Push
-Stack 3
-Stack 2
-Stack 4
-Stack 4
-Stack 4
```