Name:

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

Matrikelnummer:	*Punktzahl:*
Ergebnis:	

Freiversuch ☐ *F1* ☐ *F2* ☐ *F3* ☐

Klausur im SS 2008 :

☐ ☐ **Programmierkonzepte**
Informatik III

Medieninformatik ☐ *Praktische Informatik* ☐ *Technische Informatik* ☐
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.
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* diente lediglich bei der Klausurerstellung dem Zweck der Ausgabeumlenkung.

Geplante Punktevergabe

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

Punktziel	Sonderpunkte	erreicht
A1: 24 (3+8+5+8) P.		
A2: 20 (6+ 1,5+1,5 +4+2+5) P.		
A3: 15 (2+2+4+3+2+2) P.		
A4: 41 (6+6+4+2+2+2 + **19**) 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:

a.) Legen Sie die Schnittstelle von `minMax` fest, d.h. die Deklaration der Funktion.
(*——- Lösung auf einem Extrablatt ———*)

b.) Spezifizieren Sie (z.B. mit Worten, noch **nicht** als C++-Code) mindestens drei wirklich verschiedene Tests.
(*——- Lösung auf einem Extrablatt ———*)

c.) Implementieren Sie nun einen der Tests als C++-Funktion.
(*——- Lösung auf einem Extrablatt ———*)

d.) 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 auf einem Extrablatt ———*)

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 auf einem Extrablatt ———*)

b.) Dürfte das Argument des Kopier-Konstruktors von `Stack` auch als Werteparameter übergeben werden?
(*——- Lösung auf einem Extrablatt ———*)

c.) Dürfte das Argument des Zuweisungsoperators von `Stack` auch als Werteparameter übergeben werden?
(*——- Extrablatt, warum es meist so nicht erfolgt ——— *)

d.) Welche Unterschiede gibt es zwischen Java und C++ bei der Belegung des Heap- und Stackspeichers mit Variablen?
(*——- Lösung auf einem Extrablatt ———*)

e.) Warum wurde das Schlüsselwort `const` bei der Methode `Empty` verwendet und bei der Methode `Push` von der Klasse `Stack` nicht?
(*——- Lösung auf einem Extrablatt ———*)

f.) Implementieren Sie den C++-Code für einen Test der Methode `Count`
(*——- Lösung auf einem Extrablatt ———*)

Aufgabe 3 : Textfragen, Team

ca. 15 (2+2+4+3+2+2) Punkte
a.) Nennen Sie zwei Basistechniken von Extreme Programming.
(*——- Lösung auf einem Extrablatt ———*)

b.) Nennen Sie die zentrale Basistechnik von Extreme Programming. Begünden Sie Ihre Antwort
(*——- Lösung auf einem Extrablatt ———*)

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 auf einem Extrablatt ———*)

d.) Erklären Sie an zwei Beispielen (2 genügen), woher das Wort **extreme** in dem Namen *Extreme Proramming* stammt.
(*——- Lösung auf einem Extrablatt ———*)

e.) Erklären Sie den Begriff des Truck Faktors.
(*——- Lösung auf einem Extrablatt ———*)

f.) Welche zwei Basistechniken von Extreme Programming tragen hauptsächlich zur Reduktion des Truck Faktors bei?
(*——- Lösung auf einem Extrablatt ———*)

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 Heap-Speicherbelegung 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 auf einem Extrablatt ———*)

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 auf einem Extrablatt ———*)

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 auf einem Extrablatt ———*)

```
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
}
```

(*——- Lösung auf einem Extrablatt ———*)

d.) In welcher Zeile der Funktion `funk3` wird der Defaultkonstruktor aufgerufen?
(*——- Lösung auf einem Extrablatt ———*)

e.) In welchen Zeilen der Funktion `funk3` wird der Kopierkonstruktor aufgerufen?
(*——- Lösung auf einem Extrablatt ———*)

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

(*——- Lösung auf einem Extrablatt ———*)

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

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.
(*——- Lösung auf einem Extrablatt ———*)

```
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
```