Name:

Prof. Dr.-Ing. Hartmut Helmke
Ostfalia
Hochschule für angewandte Wissenschaften
Fachbereich Informatik

Matrikelnummer:	*Punktzahl:*
Ergebnis:	

Freiversuch ☐ *F1* ☐ *F2* ☐ *F3* ☐

Klausur im WS 2009/2010:

Programmierkonzepte — 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 !	

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 Programmfragmenten 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.

In vielen Fällen können Sie die Lösung direkt auf dem Aufgabenblatt notieren. Falls der Platz nicht ausreichen sollte, verweisen Sie **per Pfeil** auf die Extrablätter.
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: 29: 2+2 + (5* 5) P.		
A2: 6: 6*1 P.		
A3: 18: (1+2+2+6 +4,5+2,5) P.		
A4: 10: (2+1+2+4+1) P.		
A5: 31: (4*2 +8+6 +2 +4+3) P.		
A6: 6: (3+3) P.		
Sonderpunkte Labor		xxxx
Sonderpunkte Übungen		xxxx
Summe 100 Punkte		

Aufgabe 1 : Stack-Heapspeicher

ca. 29: 2+2 + (5* 5) Punkte

Gegeben sei die folgende einfache Deklaration der Klasse `Stack` sowie die Implementierung einiger ihrer Methoden.

```
class Stack
{
private:
   enum {MAX=3};
public:
   Stack(int gr=MAX) {max=gr; anz=0;}
   ~Stack() {}
   Stack(const Stack& s2);
   Stack& operator=(const Stack& s1);

   void Push(int w) {
      data[anz]=w; anz=anz+1;}
   int Pop() {--anz; return data[anz];}
   bool Empty() {return 0==anz;}
   int Count() {return anz;}
   bool Full()  {return anz==max;}

private:
   int data[MAX];
   int anz;
   int max;
};
```

a.) Welche Methoden der Klasse können als `const` vereinbart werden und welche nicht. **Lösung:**
Empty, Full und Count.

b.) Begründen Sie an einem Beispiel, warum eine bestimmte Methode als konstant vereinbart werden kann und eine andere nicht. **Lösung:**
Empty ändert keines der drei Attribute der Klasse. Dagegen wird durch die Methode Push auf jeden Fall das Attribut `anz` modifiziert.

Veranschaulichen Sie im Folgenden jeweils grafisch die Stack- und ggf. die Heap-Speicherbelegung des folgenden Programmausschnitts zu den Zeitpunkten /*1*/ und /*2*/. [1] Um Ihnen Schreibaufwand zu ersparen, ist bereits ein Teil der Zeichnung vorgegeben. Sie können aber auch eigene Zeichnungen verwenden. Es werden auch nicht jedes Mal alle Speicherstellen aus der vorgegebenen Zeichnung zur Veranschaulichung benötigt.

Im Anschluss geben Sie jeweils noch die Ausgabe des Funktionsaufrufs in die Datei `datei` an.

c.)

```
void funk1() {
   Stack s1;
   s1.Push(2);
   /* 1 */
   datei << s1.Pop();
   /*2*/
}
```

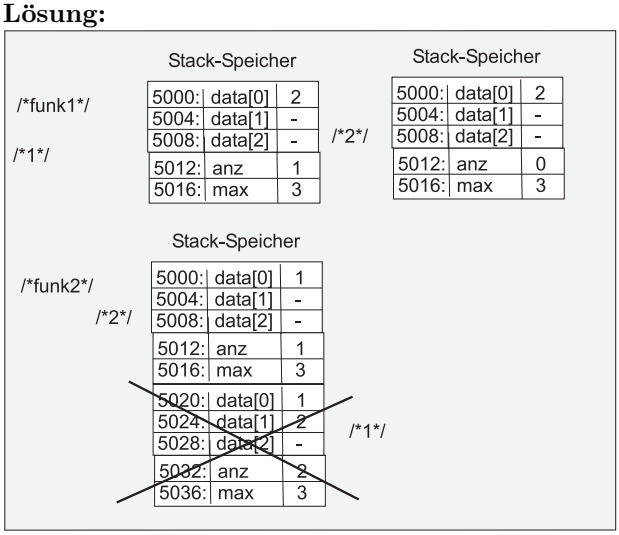
Lösung:

```
funk1: 2
```

d.)

```
void funk2b(Stack as) {
   as.Push(2);
   /* 1 */
}
void funk2() {
   Stack s;
   s.Push(1);
   funk2b(s);
   /* 2 */
   datei << s.Pop();
}
```

Lösung:



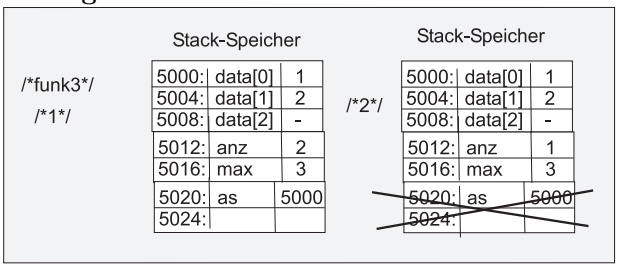
Lösung:

```
funk2: 1
```

e.)

```
void funk3b(Stack& as) {
   as.Push(2);
   /* 1 */
}
void funk3() {
   Stack s;
   s.Push(1);
   funk3b(s);
   datei << s.Pop();
   /* 2 */
}
```

Lösung:



Lösung:

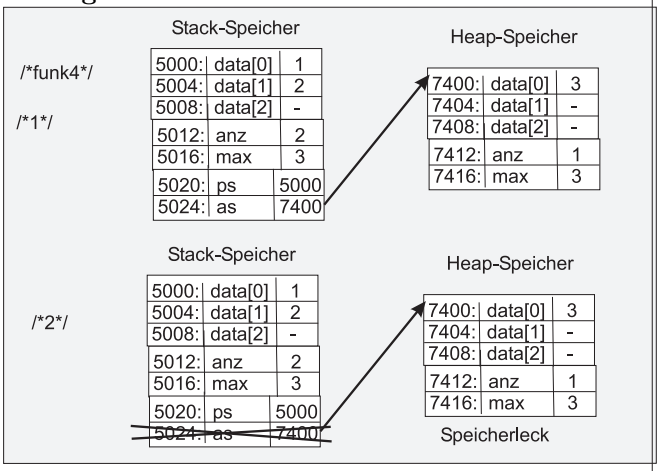
[1] Sie dürfen wie in der Vorlesung davon ausgehen, dass der Stackspeicher von den tiefen zu den hohen Adressen wächst.

```
funk3: 2
```

f.)

```
void funk4b(Stack* as) {
   as->Push(2);
   as = new Stack();
   as->Push(3);
   /* 1 */
}
void funk4() {
   Stack s;
   s.Push(1);
   Stack* ps=&s;
   funk4b(ps);
   datei << ps->Pop();
   /* 2 */
}
```

Lösung:



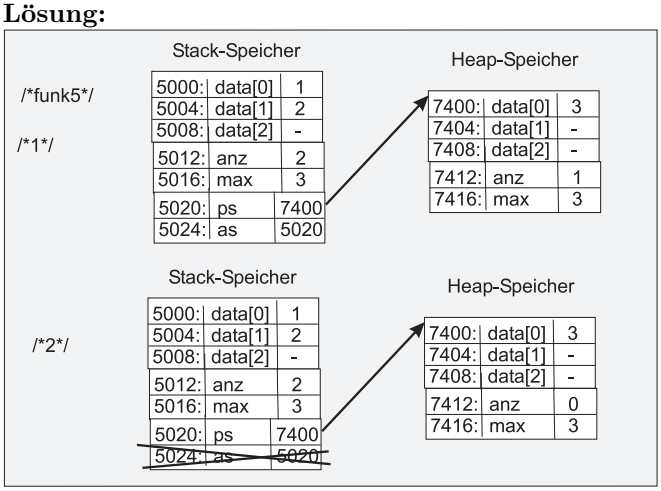
Lösung:

```
funk4: 2
```

g.)

```
void funk5b(Stack*& as) {
   as->Push(2);
   as = new Stack();
   as->Push(3);
   /* 1 */
}
void funk5() {
   Stack s;
   s.Push(1);
   Stack* ps=&s;
   funk5b(ps);
   datei << ps->Pop();
   /* 2 */
}
```

Lösung:



Lösung:

```
funk5: 3
```

Aufgabe 2 : Schleifen

ca. 6: 6*1 Punkte

a.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert hat die Variable `summe` am Ende der Schleife?

```
int summe = 0;
for (int i=13; i < 24; ++i) {
   summe++;
}
```

Lösung:

```
loop1: 11 Durchlaeufe
```

b.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert hat die Variable `summe` am Ende der Schleife?

```
int summe = 0;
for (int i=13; i < 31; i++) {
   summe++;
}
```

Lösung:

```
loop2: 18 Durchlaeufe
```

c.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert hat die Variable `summe` am Ende der Schleife?

```
int summe = 0;
for (int i=24; i > -14; i--) {
   summe++;
}
```

Lösung:

```
loop3: 38 Durchlaeufe
```

d.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert hat die Variable `summe` am Ende der Schleife?

```
int summe = 0;
for (int i=24; i > -24; i = i - 6) {
   summe++;
}
```

Lösung:

```
loop4: 8 Durchlaeufe
```

e.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert hat die Variable `summe` am Ende der Schleife?

```
int summe = 0;
int i=22;
while (i<58) {
   summe++;
   ++i;
}
```

Lösung:

```
loop5: 36 Durchlaeufe
```

f.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert hat die Variable `summe` am Ende der Schleife?

```
int summe = 0;
int i=19;
while (i<25) {
   summe++;
   i++;
}
```

Lösung:

```
loop6: 6 Durchlaeufe
```

Aufgabe 3 : Textfragen, Team

ca. 18: (1+2+2+6 +4,5+2,5) Punkte

a.) Welche Basistechnik von Extreme Programming ist ohne ein Versionsverwaltungssystem wie z.B. SVN nur mit ganz hohem Aufwand umsetzbar?

Lösung:

Gemeinsame Verantwortung, denn wie sollen ohne Versionsverwaltungssystem Änderungen am gleichen Code von verschiedenen Entwicklern zusammengeführt werden.

Auch „Fortlaufende Integration“ wird massiv durch ein Versionsverwaltungssystem unterstützt. „Pair Programming“ geht aber auch ohne, wenn es nur um das gemeinsame Programmieren geht.

b.) Ihr Boss entscheidet, dass Sie ab der nächsten Woche Extreme Programming als Prozessmodell zur Software-Entwicklung umsetzen werden. Ihr Team besteht aus zwei Personen: Heinz und Ihnen. Sie kommen jeden Tag erst um 13 Uhr zur Arbeit. Heinz geht aber immer schon um 12 Uhr nach Hause. Warum kann in diesem Team Extreme Programming nicht funktionieren, d.h. welche zwei Basistechniken von XP sind nicht anwendbar?

Lösung:

Programmieren in Paaren und das Planungsspiel. „Gemeinsame Verantwortung“ bedeutet ja erst mal nur, dass jeden jeden Code ändern darf. Dafür muss man nicht gleichzeitig am gleichen Ort zusammen kommen, sondern wird sogar durch diese verteilte Entwicklung erleichtert.

c.) Was bedeutet Refactoring?

Lösung:

Verbessern des Designs von Code, nachdem er geschrieben wurde, ohne dessen Verhalten zu ändern.

d.) Sie haben zusammen im Team eine Netzplanungs-Software mit Extreme Programming als Prozessmodell entwickelt. Ihr Auftraggeber wünscht eine Erweiterung, deren Umfang mit einem Aufwand von 6 idealen Tagen grob abgeschätzt wurde. Die Erweiterung könnte z.B. sein, dass der Netzplan nun aus einer Datenbank eingelesen werden soll.

In welchen Schritten gehen Sie bei Extreme Programming vor, damit am Ende die neue Funktionalität erfolgreich *eingecheckt* werden kann? Gehen Sie insbesondere auf das Testen ein.

Lösung:

- Wir überlegen uns eine Funktion.
- Wir beschreiben die Funktion (zur Dokumentation) in natürlicher Sprache, (was man nicht beschreiben kann, kann man auch nicht bauen).
- Wir schreiben einen Test dafür
- Wir refaktorisieren den Code in Form
- Wir implementieren die Funktion
- Wir lassen die Test Suite laufen
- Wir refaktorisieren unseren Code
- Wir sind fertig, wenn alle Tests erfüllt sind
- Wir integrieren den Code in das Versionsverwaltungssystem

e.) Beschreiben Sie zunächst mit Worten einen Test, der die korrekte Funktionsweise von Push und Pop der folgenden Klassen `Stack` prüft (ca. 2 Drittel der Punkte). Anschließend implementieren Sie den Test in C++.

```
class Stack
{
private:
   enum {MAX=3};
public:
   Stack(int gr=MAX) {max=gr; anz=0;}
   ~Stack() {}
   Stack(const Stack& s2);
   Stack& operator=(const Stack& s1);

   void Push(int w) {
      data[anz]=w; anz=anz+1;}
   int Pop() {--anz; return data[anz];}
   bool Empty() {return 0==anz;}
   int Count() {return anz;}
   bool Full()  {return anz==max;}

private:
   int data[MAX];
   int anz;
   int max;
};
```

Aufgabe 4 : Interface / ADT

ca. 10: (2+1+2+4+1) Punkte
Gegeben sei die folgende einfache Deklaration der Klasse `Stack`.

```
class Stack
{
public:
   Stack(int gr);
   ~Stack();
   void Push(int w);
   int Pop();
    inline int Count();
   int anz;

private:
    inline bool Full();
   bool Empty();
   int* data;
   int max;
};
```

a.) Welche Methoden etc. und Attribute gehören zur Schnittstelle dieser oben angegebenen Klasse `Stack`?
Lösung:
Schnittstelle enthält alles aus dem public-Bereich: Konstruktor, Destruktor, Count, Push, Pop sowie Attribut anz.
b.) Welche Methoden etc. und/oder Attribute gehören bei einem guten Klassendesign eigentlich nicht zur Schnittstelle einer Klasse, d.h. was würden Sie aus der Schnittstelle dieser Klasse `Stack` entfernen?
Lösung:
Attribute dürfen nicht im public-Bereich implementiert werden, d.h. das Attribut anz gehört hier nicht hin.
c.) Welche Methoden etc. gehören zur minimalen Standardschnittstelle einer Klasse (nicht nur von `Stack`)?
Lösung:
Default-Konstruktor, Destruktor, Kopierkonstruktor und Zuweisungsoperator
d.) Die Klasse hat im Moment noch keine minimale Standardschnittstelle. Erweitern (nicht wiederholen) Sie die Schnittstelle (Implementierung nicht erforderlich), sodass die Klasse anschließend eine minimale Standardschnittstelle aufweist.
Lösung:
- Default-Konstruktor: Stack();
- Destruktor schon vorhanden
- Kopierkonstruktor: Stack(const Stack&)
- Zuweisungsoperator: Stack& operator=(Stack&)

e.) Warum können mit der bisherigen Schnittstelle (ohne minimale Standardschnittstelle) keine Arrays vom Typ `Stack` erzeugt werden?
Lösung:
`Stack` besitzt keinen Default-Konstruktor, der wird aber aufgerufen, wenn die einzelnen Elemente eines Arrays initialisiert werden.

Aufgabe 5 : Konstruktor/Destruktor

ca. 31: (4*2 +8+6 +2 +4+3) Punkte
Gegeben sei die folgende einfache Deklaration der Klasse `Stack`.

```
class Stack
{
public:
   Stack(int gr) {datei<<"+S " << gr;
      max=gr; anz=0; data= new int[max];}

   ~Stack() {datei << "-S " << max;
     //  delete [] data;
   }
   void Push(int w);
   int Pop();

private:
   int* data;
   int anz;
   int max;
};
```

a.) Welche Ausgabe in `datei` liefert der `funk1`-Aufruf? Vergessen Sie nicht den String Ende bei der Ausgabe!

```
void funk1(){
   Stack st1(4);
   Stack st2(3);
   datei                    << " Ende ";
}
```

Lösung:

```
funk1: +S 4+S 3 Ende -S 3-S 4
```

b.) Welche Ausgabe liefert der `funk2`-Aufruf?

```
void funk2(){
   Stack* ps1 = new Stack(4);
   Stack st2(3);
   datei                    << " Ende ";
}
```

Lösung:

```
funk2: +S 4+S 3 Ende -S 3
```

c.) Welche Ausgabe in `datei` liefert der `funk3`-Aufruf?

```
void funk3(){
   Stack st1(3);
   Stack& ps = st1;
   datei << " Ende ";
}
```

Lösung:

```
funk3: +S 3 Ende -S 3
```

d.) Welche Ausgabe in `datei` liefert der `funk4`-Aufruf?

```
void help(Stack as){
   Stack x(3);
   datei << " help ";
}
void funk4(){
   Stack st1(2);
   help(st1);
   datei << " Ende ";
}
```

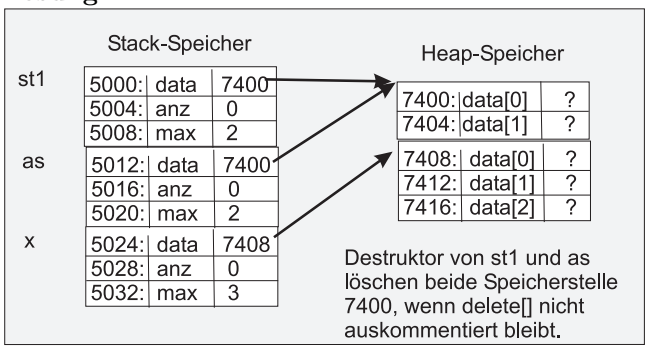
Lösung:

```
funk4: +S 2+S 3 help -S 3-S 2 Ende -S 2
```

e.) Im Destruktor von `Stack` ist der `delete`-Aufruf auskommentiert. Veranschaulichen Sie die Speicherbelegung im Stack- und im Heapspeicher bei Ausführung von `funk4`, sodass daran erkennbar wird, dass in der vorliegenden Implementierung von `Stack` der Aufruf auskommentiert bleiben muss. Ihre Zeichnung kann auch durch etwas Text zur Erklärung des Problems ergänzt werden. [a]

[a] Sie brauchen wahrscheinlich nur einen kleinen Teil der bereits vorgegebenen Speicherstellen aus der Zeichnung zu verwenden.

Lösung:



Zwei Zeiger verweisen auf die gleiche Adresse im Heap. Sowohl der Destruktor von as als der von st1 würde diesen Bereich löschen.

f.) Ist der `delete`-Aufruf auskommentiert, verbleibt ein Speicherleck. Ist er nicht auskommentiert, kann es einen Programmabsturz geben. Welche *Funktionalität* fehlt dieser Implementierung der Klasse `Stack`, damit sie auch bei Ausführung von `funk4` korrekt arbeitet. Implementieren Sie diese fehlende *Funktionalität*.

Lösung:

Der Kopierkonstruktor fehlt.

```
Stack::Stack(const Stack& s) {
   max=s.max;
   anz=s.anz;
   data = new int [max];
   for (int i=0; i < anz; ++i) {
      data[i] = s.data[i];
   }
}
```

g.) Welche Ausgabe in `datei` liefert der `funk5`-Aufruf?

```
void funk5(){
   Stack* st1[3];
   st1[0]=NULL;
   datei << " Ende ";
```

Lösung:

```
funk5:  Ende
```

h.) Implementieren Sie die Klasse `AutoPtrStack`, die automatisch nicht mehr benötigten Heap-Speicher frei gibt, sodass z.B. der folgende Funktionsaufruf von `funk6` die Ausgabe `+S 3+S 4 Ende -S 4-S 3` in `datei` erzeugen würde. Hier muss keine vollständige Implementierung der Klasse `AutoPtrStack` für alle möglichen Anwendungsfälle implementiert werden, sondern es reicht aus, wenn die oben genannte Ausgabe geliefert wird.

Tipp Nutzen Sie aus, dass C++ automatisch den Destruktor aufruft, wenn eine Variable ihren Gültigkeitsbereich verlässt.

```
void funk6(){
   AutoPtrStack pa(new Stack(3));
   AutoPtrStack pb(new Stack(4));
   datei << " Ende ";
}
```

Lösung:

```
class AutoPtrStack {
   Stack* mp;
public:
   AutoPtrStack(Stack* p): mp(p) {;}
   ~AutoPtrStack() {delete mp;}
};
```

i.) Erweitern Sie die Klasse `AutoPtrStack` nun noch zu einer generischen Klasse `AutoPtr`$< T >$, die

unabhängig vom übergebenen Zeigertyp ist, sodass z.B. das folgende Programmfragment möglich ist und wiederum die Ausgabe `+S 3+S 4 Ende -S 4-S 3` in `datei` erzeugt werden würde.

```
void funk7(){
   AutoPtr<Stack> pa(new Stack(3));
   AutoPtr<Stack> pb(new Stack(4));
   AutoPtr<double> pd(new double(11));
   datei << " Ende ";
}
```

Lösung:

```
template <typename T>
class AutoPtr {
   T* mp;
public:
   AutoPtr(T* p): mp(p) {;}
   ~AutoPtr() {delete mp;}
};
```

Aufgabe 6 : STL

ca. 6: (3+3) Punkte

a.) Welche Ausgabe erzeugt der Aufruf von *funk1* in `datei`?

```
void Print(int i){
   datei << i << " ";
}

void funk1(){
    list <int> li1;
   for (int i=1; i<7; ++i) {
      li1.push_back(i);
   }
    for_each(li1.begin(), li1.end(),Print);
    datei << endl;

    list <int>::iterator iter =
       find(li1.begin(), li1.end(),4);
    for_each(++iter, li1.end(),Print);
    datei << endl;

   *iter = 61;
    for_each(li1.begin(), li1.end(),Print);
    datei << endl;
}
```

Lösung:

```
funk1
1 2 3 4 5 6
5 6
1 2 3 4 61 6
```

b.) Welche Ausgabe erzeugt der Aufruf von *funk2* in `datei`?

```
void funk2(){
    list <int> li1;
   for (int i=1; i<4; ++i) {
      li1.push_back(i);
   }
    li1.push_back(2);
    li1.push_back(2);
    li1.push_back(7);
    for_each(li1.begin(), li1.end(),Print);
    datei << endl;


    list <int>::iterator iter =
       max_element(li1.begin(), li1.end());
    datei << *iter << endl;

    iter = remove(li1.begin(), li1.end(), 2);
    for_each(li1.begin(), li1.end(),Print);
    --iter;
    datei << "\niter= " << *iter << endl;

    li1.erase(iter, li1.end());
    datei << "Nach erase: ";
    for_each(li1.begin(), li1.end(),Print);
    datei << endl;
}
```

Lösung:

```
1 2 3 2 2 7
7
1 3 7 2 2 7
iter= 7
Nach erase: 1 3
```