Name:

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

Matrikelnummer:	*Punktzahl:*
Ergebnis:	

Freiversuch ☐ *F1* ☐ *F2* ☐ *F3* ☐

Klausur im WS 2015/16:

Die verschiedenen Programmierparadigmen von C++

Informatik Bachelorstudiengang. ☐ *Informatik Masterstudiengang* ☐

Hilfsmittel sind bis auf Computer, Handy etc. erlaubt !	Bitte Aufgabenblätter mit abgeben !
Austausch von Hilfsmitteln mit Kommilitonen ist **nicht** erlaubt !	
Anwesenheit von Handys, Smartphones etc. bei Klausurteilnehmern im Hörsaal nicht erlaubt.	
Sie sind vor Beginn der Klausur am Dozentenpult abzugeben!	

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.

Meistens kann die Lösung direkt auf dem Aufgabenblatt notiert werden. Extrablätter bitte mit Namen und/oder Matrikelnummer versehen.
Gehen Sie davon aus, dass `double` 8 Bytes sowie `int` und Zeiger jeweils 4 Bytes im Speicher belegen. Aufrufe von `new` sollen jeweils den nächsten zuhängenden ausreichend großen freien Speicherbereich ab den Adressen beginnend bei 7400 liefern. Die Speicherbelegung soll hier auf Stack und Heap jeweils von den tieferen zu den höheren Adressen verlaufen.

Geplante Punktevergabe

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

Punktziel	Sonderpunkte	erreicht
Laboraufgaben: 10 +10 P.		
A1: 15 P.		
A2: 20 P.		
A3: 39 P.		
A4: 26 P.		
Summe 100 P.		

Aufgabe 1 : Schleifen

ca. 15 Punkte

a.) (1,5 P.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert liefert die Ausgabe der Variablen `zaehler`?

```
int zaehler = 0;
for (int i=10; i < 23; ++i) {
   ++zaehler;
}
datei << zaehler << " runs\n";
```

(*— Lösung hier notieren. ——*)

b.) (1,5 P.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert liefert die Ausgabe der Variablen `zaehler`?

```
int zaehler = 0;
for (int i = 10; i < 23; i++) {
   zaehler++;
}
datei << zaehler << " runs\n";
```

(*— Lösung hier notieren. ——*)

c.) (1,5 P.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert liefert die Ausgabe der Variablen `zaehler`?

```
int zaehler = 0;
for (int i = 11; i > 8; --i) {
   zaehler++;
}
datei << zaehler << " runs\n";
```

(*— Lösung hier notieren. ——*)

d.) (1,5 P.) Wie oft wird die folgende Schleife durchlaufen, d.h. welchen Wert liefert die Ausgabe der Variablen `zaehler`?

```
int zaehler = 0;
for (int i = 100; i > 20; i = i - 10) {
   zaehler++;
}
datei << zaehler << " runs\n";
```

(*— Lösung hier notieren. ——*)

e.) (1,5 P.) Welche Ausgabe erzeugt das folgende Code-Fragment in `datei`?

```
vector<int> cont;
for (int i = 14; i < 18; ++i) {
   cont.push_back(i + 3);
}
for (unsigned int i = 0; i < cont.size(); ++i) {
   datei << cont[i] << " ";
}
```

(*— Lösung hier notieren. ——*)

f.) (1,5 P.) Welche Ausgabe erzeugt das folgende Code-Fragment in `datei`?

```
list<int> cont;
for (int i = 14; i < 18; ++i) {
   cont.push_back(i + 3);
}
list<int>::iterator iter = cont.begin();
while (iter != cont.end()) {
   datei << *iter << " ";
   ++iter;
}
```

(*— Lösung hier notieren. ——*)

g.) (2 P.) Welche Ausgabe erzeugt das folgende Code-Fragment in `datei`?

```
list<int> cont;
for (int i = 2; i < 14; ++i) {
   cont.push_back(i);
}
datei << *max_element(cont.begin(), cont.end());
datei << ", ";
datei << *find(cont.begin(), cont.end(), 12);
```

(*— Lösung hier notieren. ——*)

h.) (4 P.)
Welche Ausgabe erzeugt das folgende Code-Fragment in `datei`?

```
void Print(int i) {
   datei << i << " ";
}
void funk8() {
   vector<int> cont;
   for (int i = 3; i < 6; ++i) {
      cont.push_back(i * i);
      cont.push_back(i);
   }
   for_each(cont.begin(), cont.end(), Print);
   datei << "\n";
   sort(++cont.begin(), cont.end()); // increasing
   for_each(cont.begin(), cont.end(), Print);
}
```

(*— Lösung hier notieren. ——*)

Aufgabe 2 : Instanzerzeugung

ca. 20 Punkte

Für diese Aufgabe und die folgende wird die folgende Klassendefinition verwendet.

```
class Task {
public:
   Task(int arg) {
      id = arg;
      datei << "+T" << id << " ";
      zeiten = nullptr;
      punkte = 2;
   }
   Task(int arg, int st, int copy){
      id = arg;
      datei << "+T" << id << " ";
      zeiten = new int[3];
      zeiten[0] = st;
      zeiten[1] = copy;
      zeiten[2] = -1;
      punkte = 0;
   }
   ~Task() {
      datei << "-T" << id << " ";
      delete[] zeiten;
      zeiten = nullptr;
   }
   int GetId() const {
      return id;
   }
 private:
   int id;
   //! Array, mit Startzeit, Kopierzeit, geplante Zeit
   int* zeiten;
   int punkte;
   friend Task Make1From3(Task t1,Task t2,Task t3);
};
```

Veranschaulichen Sie jeweils grafisch die Stack- **und/-oder** Heap-Speicherbelegung in **allen** folgenden Programmfragmenten nach Ausführung der mit `/* 1*/` usw. gekennzeichneten Programmzeilen.

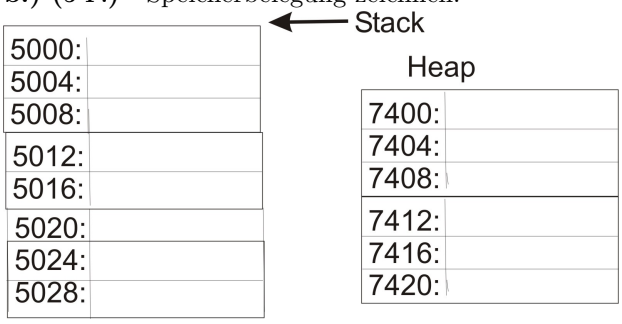
Beachten Sie bitte auch, dass bei fast allen Aufgaben, die Ausgabe des Funktionsaufrufs in den Stream `datei` zu notieren ist.

a.) (3 P.) Welche Ausgabe ergibt der Aufruf von `f10`?

```
void f10() {
   Task t1(8, 4, 3);
   Task t2(9); /*1*/
   }
```

(*— Lösung hier notieren. ——*)

b.) (5 P.) Speicherbelegung zeichnen:

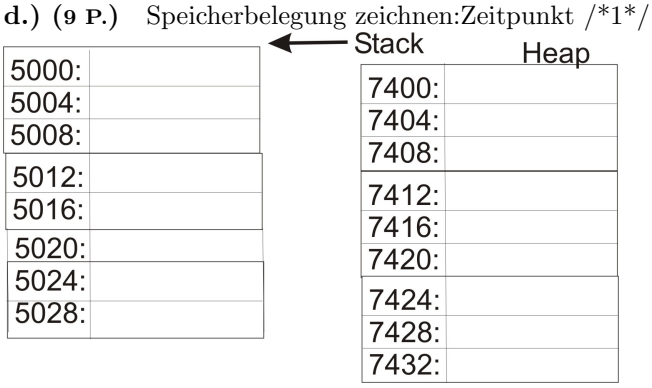


c.) (3 P.) Welche Ausgabe ergibt der Aufruf von `f11`?

```
void f11() {
   Task t1(8);
   Task* p2 = new Task(1, 2, 13); /*1*/
}
```

(*— Lösung hier notieren. ——*)

d.) (9 P.) Speicherbelegung zeichnen:Zeitpunkt /*1*/



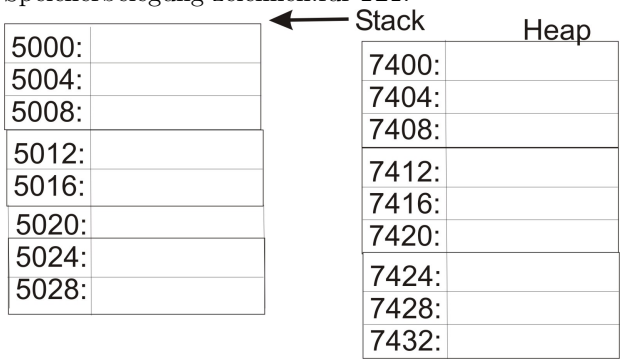
Aufgabe 3 : Flache und tiefe Kopie

ca. 39 Punkte

a.) (5 P.) Veranschaulichen Sie die Speicherbelegung zum Zeitpunkt /*1*/ nach Aufruf von `f21`.

```
void f21() {
   Task t1(8, 4, 3);
   Task t2(t1);       /*1*/
   datei << t2.GetId() << " ";
}
```

Speicherbelegung zeichnen:für `f21`:



b.) (5 P.) Erklären Sie mit Hilfe Ihrer Zeichnung, warum der Aufruf der Funktion `f21` zu einem Programmabsturz oder zumindest zu undefinierten Verhalten des Programms führt.

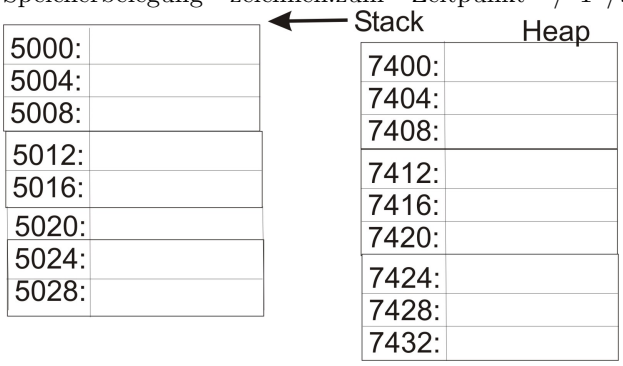
(*— Erklärung der Probleme: ——*)

c.) (5 P.) Erweitern Sie `Task` (mit oder ohne Ausgabe nach `datei`) direkt `inline` in der Klassenschnittstelle von `Task`, sodass die Funktion `f21` weiterhin syntaktisch korrekt ist und so, dass ihr Aufruf nun zu korrektem Verhalten führt.

(*— Erweiterung von `Task`: ——*)

d.) (4 P.) Veranschaulichen Sie nun nochmals die Speicherbelegung beim Aufruf von `f21` nach Ihrer Verbesserung.

Speicherbelegung zeichnen:zum Zeitpunkt /*1*/:



e.) (6 P.) Gehen Sie im Folgenden davon aus, dass Ihre Verbesserung aus der vorherigen Teilaufgabe für die Klasse `Task` umgesetzt ist und dass Ihre Verbesserung auch eine geeignete Ausgb in den Stream `datei` erzeugt [z.B. `+TCo id`).
Das folgende Listing zeigt die Funktion `Make1From3`

```
   /* Kombination von 3 Tasks zu einer indem die
    Attribute adddiert werden.*/
Task Make1From3(Task t1, Task t2, Task t3){
   datei << "Make1From3 called\n";
   Task hlp(0, -1, -1);
   hlp.punkte = t1.punkte + t2.punkte + t3.punkte;
   hlp.id = t1.id + t2.id + t3.id;
   hlp.zeiten[0] = t1.zeiten[0] +
      t2.zeiten[0] + t3.zeiten[0];
   hlp.zeiten[1] = t1.zeiten[1] +
      t2.zeiten[1] + t3.zeiten[1];
   return hlp;
}
```

`Make1From3` wird in der Funktion `f32` aufgerufen.

```
   Task t1(1, 1, 1);
   Task t2(2, 2, 2);
   Task t3(3, 3, 3);

   Task t4 = Make1From3(t1, t2, t3);
   datei << t4.GetId() << " ";
}
```

Wie viele `int`-Objekte werden durch Aufruf von `f32` bis zum Ende der Funktion auf dem Heap erzeugt, d.h. wie oft (multipliziert mit 3) wird der Code `new int[3];` ausgeführt.

Geben Sie hierzu die Ausgabe des Aufrufs von `f32` in den Stream `datei` an und beginnen Sie jeweils eine neue Zeile in Ihrer Ausgabe, sobald `new int[3];` ausgeführt wird. Ergänzen Sie dann jeweils den Text „`3 int mit new`“.

Hier dürfen Sie (müssen aber nicht) zur Erklärung Ihrer Überlegungen weiteren Text ergänzen. Ihre Ausgabe könnte z.B. wie folgt aussehen:

```
+T1  +T7
   3 int mit new // t1 mit 1,4,8 aufgerufen
-T7 +T11
   3 int mit new // Kopierkonstruktor
           //wird aufgerufen
1 operator= // abcde jfj
  3 int mit new
   +TCopy7
  3 int mit new
...
```

Geben Sie Ihre Ausgaben am besten auf einem Extrablatt an.

f.) (5 P.) Sie haben gesehen, dass eine Menge teilweise *unnötiger* `int`-Instanzen auf dem Heap erzeugt wurden.
Wie erreicht man allein durch Anpassung der Schnittstelle von `Make1From3`, dass weniger `int`-Instanzen auf dem Heap erzeugt werden.? Sie dürfen davon ausgehen, dass `Make1From3` mit geänderte Schnittstelle dann auch `friend` der Klasse `Task` ist.
(*— Lösung hier notieren. ——*)

g.) (2 P.) Wie viele int-Instanzen werden nun insgesamt noch erzeugt?
(*— Lösung hier notieren. ——*)

h.) (5 P.) Erweitern Sie nun zusätzlich die Klassenschnittstelle von `Task` um einen Kopierkonstruktur mit Move-Semantik (die *Dinger* mit `&&`-Parametern sind gemeint), sodass bei Aufruf von `f32` noch weniger `int`-Instanzen auf dem Heap erzeugt werden. Sie dürfen wiederum direkt in der Klassenschnittstelle von `Task` in der Header-Datei implementieren.
(*— Lösung hier notieren. ——*)

i.) (2 P.) Wie viele int-Instanzen werden nun insgesamt noch erzeugt?
(*— Lösung hier notieren. ——*)

Aufgabe 4 : Shared Pointer

ca. 26 Punkte

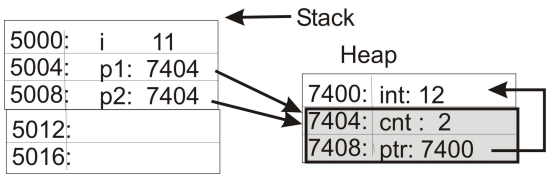
a.) (3 P.) Welche Ausgabe ergibt der Aufruf von `f12`?

```
void f12() {
   shared_ptr<Task> p1(new Task(5, 6, 7)); /*1*/
   datei << p1->GetId() << " ";
}
```

(*— Lösung hier notieren. ——*)

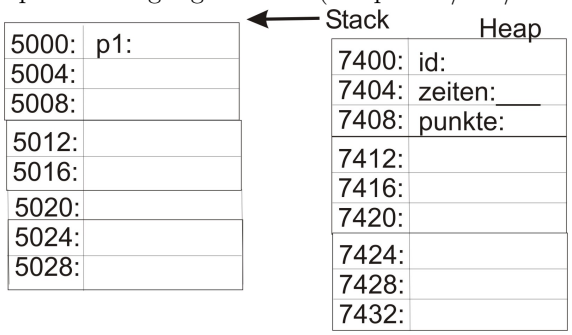
b.) (10 P.) Veranschaulichen Sie nun grafisch wiederum die Speicherbelegung zum Zeitpunkt /*1*/. Als Hilfe die Speicherbelegung bei Aufruf von `hilfe` zum Zeitpunkt /*2*/ kann die folgenden Grafik dienen:

```
void hilfe () {
   int i(11);
   shared_ptr<int> p1(new int(12));
   shared_ptr<int> p2(p1); /*2*/
}
```



Als Hilfe sind schon einige Einträge vorgegeben.

Speicherbelegung zeichnen:(Zeitpunkt /*1*/ bei `f12`):

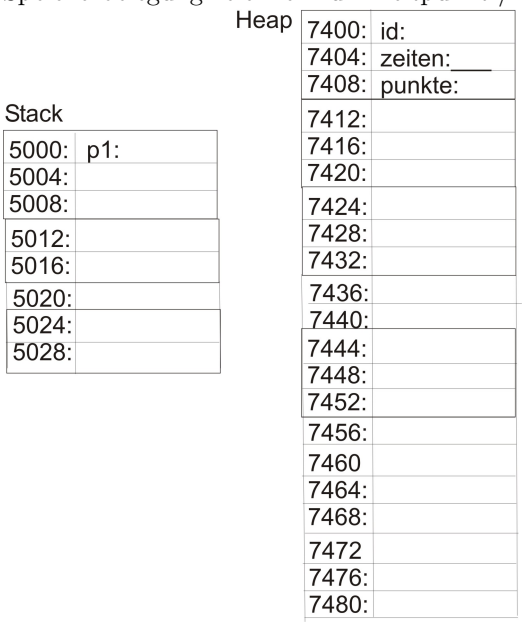


c.) (10 P.) Sie sehen hier die Funktion `f13`. Wie sehen die Speicherbelegungen zum Zeitpunkt /*2*/ und /*4*/ aus? Kennzeichnen Sie grafisch oder textlich, welche Speicherbereiche auf dem Heap bereits wieder freigegeben sind. Es bietet sich an, auch die Belegungen zu den Zeitpunkten /*1*/ und /*3*/ grob zu veranschaulichen (auch wenn es dafür keine Punkte gibt).

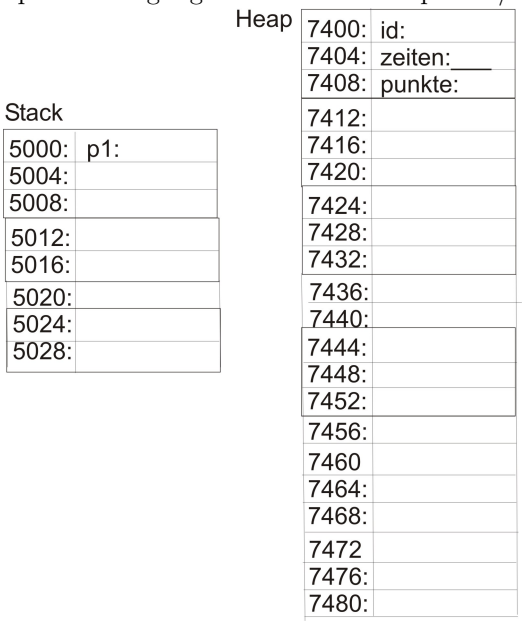
```
void f13() {
   shared_ptr<Task> p1 (new Task(1,2,3));
   shared_ptr<Task> p2; /*1*/
   if (p1->GetId() < 100) {
      shared_ptr<Task> p3(new Task(12));
      /*2*/
      p2 = p3; /*3*/
      p1 = p3; /*4*/
   }
   datei << p1->GetId() << " ";
}
```

Ganz wenige Einträge sind zur Orientierung, welches Objekt wohl zuerst erzeugt wird, sind schon vorgegeben.

Speicherbelegung zeichnen:zum Zeitpunkt /*2*/



Speicherbelegung zeichnen:zum Zeitpunkt /*4*/



d.) (3 P.) Mit diesen Vorüberlegungen sollte Ihnen nun die Ausgabe des Funktionsaufrufs von `f13` in den Stream `datei` einfach gelingen. Wie sieht die Ausgabe aus?

(*— Lösung hier notieren. ——*)