OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG | INF | FAKULTÄT FÜR INFORMATIK

Institut für Simulation und Graphik
Dr. Christian Rössl

Magdeburg, 15.7.2013

Klausur zur Vorlesung „Grundlagen der C++-Programmierung“

Name, Vorname		Studiengang	Matrikelnummer
Zusatzblätter	Unterschriften unbenoteter Schein ☐	Student/in	Aufsicht

Tabelle bitte *nicht ausfüllen*!

	Aufgabe	max. Punkte	erreicht
1	Dereferenzierung	4	
2	Vererbung und virtuelle Methoden	10	
3	Typen	6	
4	Zeigerarithmetik	5	
5	Speicherverwaltung	9	
6	Smart-Pointer	5	
7	Verwendung von `const`	7	
8	Exceptions	9	
9	Kurzfragen	5	
		60 (+0)	

Bearbeitungszeit: 120 Minuten

Hinweise zur Klausurbearbeitung

- Überprüfen Sie die Klausur auf Vollständigkeit (10 nummerierte Blätter).
- Füllen Sie das Deckblatt aus!
- Beschriften Sie **alle** Blätter (Klausur, verteiltes Papier) mit Ihrem Namen.
- Legen Sie bitte alle für die Klausur benötigten Dinge, Stifte, Verpflegung und insbesondere Lichtbildausweis, auf Ihren Tisch. Schalten Sie Ihr **Mobiltelefon aus**!
- Hilfsmittel (Taschenrechner, Bücher, Skripten, Mobiltelefone, . . .) sind **nicht** zugelassen!
- Täuschungsversuche führen zum Nichtbestehen der Klausur!
- Verwenden Sie für Ihre Antworten den freien Platz nach den Aufgaben und ggf. die Rückseiten der Blätter. – Melden Sie sich, wenn Sie zusätzliche leere Blätter benötigen.
- **Schreiben Sie deutlich!** Unleserliche Passagen können nicht korrigiert werden.
- Benutzung von Bleistiften sowie roter und grüner Stiftfarbe ist untersagt.
- Beschränken Sie sich auf die geforderten Angaben und halten Sie Ihre Antworten kurz und präzise. Nicht geforderte Angaben ergeben keine zusätzlichen Punkte.

- **Zu C++** : Sie dürfen bei allen Antworten "**using namespace** `std;`" und **C++11** *voraussetzen*. Außerdem brauchen Sie (falls nicht explizit verlangt) *keine* Standard-**#include**-Dateien angeben.

Viel Erfolg!

Aufgabe 1: Dereferenzierung von Zeigern (4 Punkte)

Die folgenden Klassen beschreiben ein fiktives Raumschiff. (Die Konstruktoren sind nicht angegeben, Sie benötigen nur die Attribute.)

```
1  class Wing;
2  class Body;
3  class Laser;
4  class Cargo;
5
6  class Spaceship {
7  public:
8    Wing* left;
9    Body& body;
10   Wing* right;
11 };
12
13 class Wing {
14 public:
15   Spaceship* ship;
16   Laser&     laser;
17 };
18
19 class Body {
20 public:
21   Spaceship* ship;
22   Cargo*     cargo;
23 };
24
25 struct Cargo {
26   int data;
27 };
```

Nehmen Sie im folgenden an, dass alle Zeiger und Referenzen sinnvoll initialisiert sind. Gegeben ist eine Referenz

```
Wing& leftWing
```

auf einen linken Flügel.

(a) Geben Sie jeweils einen möglichst einfachen Ausdruck in Abhängigkeit von `leftWing` an, der folgendes referenziert bzw. ausdrückt:

- den `Laser` des *rechten* Flügels,
- die Ladung `data` des Raumschiffs,
- einen Zeiger auf den Körper `body` des Raumschiffs,

(b) Welche Typen haben die folgenden Ausdrücke?

- `&(leftWing.laser)`
- `&((&leftWing)[0].ship->right)`

Aufgabe 2: Vererbung und virtuelle Methoden (10 Punkte)

Gegeben ist das folgende Programm.

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Animal {
6  public:
7    virtual ~Animal() { cout << "die" << endl; }
8    virtual void move() = 0;
9  };
10 class Fish : public Animal {
11 public:
12   ~Fish() { cout << "turn upside down" << endl; }
13   virtual void move() { cout << "swim" << endl; }
14 };
15 class Bird : public Animal {
16 public:
17   virtual void move() { cout << "fly" << endl; }
18 };
19 class Penguin : public Bird {
20 public:
21   ~Penguin() { cout << "freeze" << endl; }
22   virtual void move() { cout << "swim or walk" << endl; }
23 };
24
25 int main() {
26   Animal* tux=new Penguin;   tux->move();
27   Fish* nemo=new Fish;       nemo->move();
28   delete tux;
29   delete nemo;
30   return 0;
31 }
```

(a) Was bedeutet `move() = 0` (Zeile 8) in der Klasse `Animal`?

(b) Was gibt `main()` aus?

(c) Welche Zeiger liefern die folgenden Ausdrücke (nehmen Sie Auswertung in Zeile 27 an)
(1.) **dynamic_cast**`<Bird*>(tux)` und
(2.) **dynamic_cast**`<Fish*>(tux)` ?

Erklären Sie an diesem Beispiel *kurz* **dynamic_cast**`<>`.

(d) Wie ändert sich die Ausgabe, wenn Sie den Destruktor `Animal::~Animal()` (Zeile 7) *nicht* als **virtual** deklarieren?

Welche Variante ist vorzuziehen?

Aufgabe 3: Typen (6 Punkte)

(a) Definieren Sie die Typen von `x` in den folgenden Ausdrücken mit Hilfe von **typedef** als `XTYPE`. (Das heißt, falls **auto** steht, definieren Sie den entsprechenden Typen explizit.)

Beispiel: `int y; auto x=y;` $\Rightarrow$ *Antwort:* `typedef int XTYPE;`

- `int y; auto x=&y;`
- `int y[3]; auto x=y[1];`
- `int y[3]; auto x=&y[1];`
- `int* y; auto x=y;`
- `int* y; auto x=&y;`
- `int x[3];`
- `auto x="xyz";`
- `void x(int y) {}`

(b) Erzeugen Sie einen *Zeiger* `px` auf ein gegebenes Datum `x`. Geben Sie den Typ des Zeigers explizit an, das heißt, verwenden Sie dabei *weder* **typedef** (oder **using**) noch **auto**.

Beispiel: `int x;` $\Rightarrow$ *Antwort:* `int* px=&x;`

- `int* x;`
- `int& x=y;`
- `int x[2];`
- `void x(int y) {}`

Aufgabe 4: Zeigerarithmetik (5 Punkte)

Nehmen Sie an, die Tabelle zeigt einen Auszug des Hauptspeichers eines fiktiven Rechners (Adressen `0x00`,...,`0x3f`). Dabei soll gelten **sizeof**`(`**int**`)=4`, d.h. in jede Zeile der Tabelle „passt“ ein **int** Wert.

	0	1	2	3
0x3c				
0x38				
0x34				
0x30				
0x2c				
0x28				
0x24				
0x20				
0x1c				
0x18				
0x14				
0x10				
0x0c				
0x08				
0x04				
0x00				

Geben Sie die Belegung des Speichers nach Ausführung des folgenden Programmausschnitts an, indem Sie die entsprechenden Werte in die Tabelle eintragen. Tragen Sie dazu **char** Werte (d.h. „ein Byte pro Kästchen“) als Dezimalzahl ein.

Hinweis: Nehmen Sie an, dass alle Zeiger in den abgebildeten Speicherbereich zulässig sind. Außerdem ist der Wert von leeren Feldern *undefiniert*.

```
1  void* p=(void*) 0x11;
2
3  char* q=(char*) p;
4  *q=1;
5  *(q+4)=2;
6  ++q;
7  q[2]=3;
8
9  int* r=(int*) &q[6];
10 q=(char*) (r+2);
11 *q=4;
12 *((char*) &r[-2])=5;
13
14 *++q=6;
15 *q++=7;
16 q[0]=8;
```

Aufgabe 5: Speicherverwaltung: Was passiert wann? (9 Punkte)

Im folgenden Code-Beispiel soll die Funktion `meeting()` aufgerufen werden.

```
1  #include <string>
2  #include <iostream>
3  #include <memory>
4
5  using namespace std;
6
7  struct Cookie {
8    string type;
9    Cookie() : type("Dough") {
10     cout << "Dough prepared" << endl;
11   }
12   Cookie(const string& type) : type(type) {
13     cout << type << " baked" << endl;
14   }
15   ~Cookie() {
16     cout << type << " vanished" << endl;
17   }
18 };
19
20 Cookie* mix(const Cookie* box1,int num1,
21             const Cookie* box2,int num2) {
22   Cookie* kmix=new Cookie[num1+num2];
23   for(int i=0;i<num1;++i) kmix[i]     =box1[i];
24   for(int i=0;i<num2;++i) kmix[i+num1]=box2[i];
25   return kmix;
26 }
27
28 bool meeting() {
29   Cookie box1[]={Cookie("Bisquit"),Cookie("Chocolate")};
30   Cookie box2[]={Cookie("Acacookie")};
31   Cookie* bowl=mix(box1,2,box2,1);
32   if (nobodyEatsCookies(bowl))
33     return false;
34   delete bowl;
35   return true;
36 }
```

(a) Was gibt ein Aufruf von `meeting()` aus, für den Fall, dass `nobodyEatsCookies(bowl)` (Zeile 32) **false** liefert?

(b) Geben Sie zu jeder Zeile der Ausgabe in (a) zusätzlich an, ob die ausgebende Instanz von `Cookie` auf dem *Stack* oder auf dem *Heap* liegt. (Also: „Zeigt **this** auf den *Stack/Heap*?“)

(c) Der angegebene Programmcode enthält zwei Fehler, die zu Speicherlecks führen.

Markieren Sie die Stellen im Quelltext, und geben Sie eine korrigierte Variante der jeweiligen Zeile an!

Aufgabe 6: Smart-Pointer (5 Punkte)

Der folgende Quelltext (abgewandelt von Aufgabe 5) soll so geändert werden, dass ausschließlich `unique_ptr<>` und `shared_ptr<>` zur Speicherverwaltung verwendet werden.

Füllen Sie dazu die Lücken, indem Sie `unique_ptr<···>` bzw. `shared_ptr<···>` oder wenn möglich Referenzen sinnvoll verwenden. (Der Einsatz „normaler“ Zeiger ist nicht erlaubt!)

Hinweis: Beginnen Sie ausgehend von der Definition von `nobodyEatsCookies()` (Zeile 16).

```
1  #include <string>
2  #include <iostream>
3  #include <memory>
4
5  using namespace std;
6
7  struct Cookie {
8    string type;
9    Cookie() : type("Dough") {cout << "Dough prepared\n";}
10   Cookie(_____________________) : type(type) {
11     cout << type << " baked\n";
12   }
13   ~Cookie() {cout << type << " vanished\n";}
14 };
15
16 bool nobodyEatsCookies(const unique_ptr<Cookie[]>& bowl) {
17   return false;
18 }
19
20 _____________ mix(_____________,int num1,
21                   _____________,int num2) {
22
23   _____________ kmix _____________;
24
25   for(int i=0;i<num1;++i) kmix[i]     =box1.get()[i];
26   for(int i=0;i<num2;++i) kmix[i+num1]=box2.get()[i];
27
28   return _____________;
29 }
30
31 bool meeting() {
32   _____________ box1(new Cookie("Double Chocolate"));
33   _____________ box2(new Cookie("Acacookie"));
34   _____________ bowl=mix(box1,1,box2,1);
35
36   if (nobodyEatsCookies(bowl))
37     return false;
38
39   cout << bowl[1].type << " is the best one."
40        << bowl1->type << " is good as well." << endl;
41
42   return true;
43 }
```

Aufgabe 7: Verwendung von `const` (7 Punkte)

(a) Füllen Sie die Lücken aus! Entscheiden Sie dazu, ob **const** stehen sollte (bzw. stehen muss) oder nicht. Beachten Sie, dass der Code in jedem Fall korrekt sein muss.

(b) Die Verwendung dieser *header*-Datei kann zu einem Fehler beim *Linken* führen. Warum? Was müssten Sie ändern, um den Fehler zu beseitigen?

```
1  #ifndef SPACESHIP_HH
2  #define SPACESHIP_HH
3
4  class Laser;
5
6  class SpaceShip {
7  public:
8    SpaceShip(float startHealth, ______ Laser& laser)
9      : totalHealth(startHealth), health(startHealth),
         laser(laser) {}
10   void applyDamage(float damage) ______ {
11     health-=damage;
12     if (health<=0) throw "BIG EXPLOSION";
13   }
14   float getRelativeHealth() ______ {
15     return health/totalHealth;
16   }
17   const Laser& getLaser() ______ { return laser; }
18 private:
19   ______ float totalHealth;
20   ______ float health;
21   ______ Laser& laser;
22 };
23
24 class Laser {
25 public:
26   Laser() : shotCount(0) {}
27   void shootLaser(SpaceShip& enemyShip) ______ {
28     enemyShip.applyDamage(DAMAGE_PER_SHOT);
29     ++shotCount;
30   }
31 private:
32   ______ int shotCount;
33   ______ static ______ float DAMAGE_PER_SHOT;
34 };
35
36 ______ float Laser::DAMAGE_PER_SHOT = 90.01f;
37
38 #endif // SPACESHIP_HH
```

Aufgabe 8: Exceptions (9 Punkte)

Welche Ausgabe erzeugt das folgende Programm?

```
#include <iostream>

using namespace std;

struct Scope {
  int id;

  Scope(int p) : id(p) { cout << "create " << id << endl; }
  ~Scope() { cout << "destroy "<< id << endl; }
};

void f(int n) {
  cout << "enter f, n=" << n << endl;
  Scope scope(n);
  if (n==0)
    throw 42;
  f(n-1);
  cout << "leave f, n=" << n << endl;
}

int main() {

  try {
    Scope scope(99);
    f(2);
  } catch (int e) {
    cout << "caught " << e << endl;
  }

  return 0;
}
```

Aufgabe 9: Kurzfragen zu allen Themen (5 Punkte)

Beantworten Sie die folgenden Fragen bzw. Aufgaben entweder kurz in Stichworten oder ggf. mit einem kurzen Code-Fragment.

(a) Was versteht man unter *include guard*? Geben Sie zur Erklärung ein kurzes Beispiel an.

(b) Nennen Sie zwei verschiedene Möglichkeiten, Programmbibliotheken zu realisieren.

(c) Ist `bool is_cookie(){ return true; }` eine Deklaration, Definition oder Initialisierung?

(d) Welche Werte haben die Variablen `a`, `b` und `c` in folgendem Code-Fragment?

```
{ static int a;
  const int  b = 1;
  int        c;
}
```

(e) Nennen Sie einen Nachteil von Makros!

(f) Ordnen Sie folgende Werte aufsteigend nach ihrer Größe. Verwenden Sie dazu die Relationen $<, \leqslant, =$.

sizeof(int), **sizeof(short)**, **sizeof(char)**, **sizeof(long)**, **sizeof(unsigned)**

(g) Welche Sichtbarkeit haben `method` und `attribute` in

```
class A {
  void method();
  int  attribute;
};
```

(h) Was kann in `template< ? X> struct A;` für `?` eingesetzt werden? Nennen Sie zwei Möglichkeiten!

(i) Was ist falsch? Warum? (kurze Begründung)

```
int* p=new int[4];
int  diff=p-((int*) nullptr);
```

(j) Geben Sie einen alternativen Ausdruck für `p>q` an, wobei beide Zeiger `q`,`p` vom Typ **int*** sind. Es soll weiterhin Die Relation $>$ verwendet werden. (Hinweis: Zeigerarithmetik)