Klausur OOP SS2002 (Gampp)

Lösung:

Aufgabe 1)

1. using namspace std; <-fehlt
2. virtual C() <- Konstruktor darf nicht virtuell sein
3. a=b <- b ist unbekannt u nicht definiert (Das es in der abgeleiteten Klasse exestiert ist nicht von Bedeutung)
4. C c = new C; <- new C gibt als Returnwert einen Pointer auf das Element C c ist aber kein Pointer -> richtig wäre C *c = new C;
5. b=0; <- Zuweisung darf nur in Methoden erfolgen -> b=0 steht aber außerhalb einer Methode
6. Class D: public C {…}; Semicolon bei Class D fehlt
7. void C::f(int b) {…} <- Prototyp der Funktion in C hat keinen Parameter
8. cout << x << this; Typ x ist nicht definiert in iostream und kann daher so nicht ausgegeben werden. Der Pointer this ist nur in Klassen erlaubt und muss sich auf etwas beziehen. Ausgabe von this ist theoretisch möglich (in Klassen) wenn er sich auf ein Datentyp bezieht der in iostream definiert ist.
9. return x.a; <- a ist protected daher kein Zugruff möglich

Aufgabe 2)

a) Ausgabe lautet:
   a. 4
   b. 9
   Begründung:
      1. A obj; ruft Konstruktor von A auf -> i=1
      2. obj.add(3); ruft A::add auf mit p=3 -> i=1+3 -> 4
      3. obj.access(); ruft A::access auf p=5 -> Aufruf von A::add mit p=5 -> i = 4+5 -> 9 (i ist immer noch 4!!)

b) Aufrufreihenfolge für Aufgabe b:
   1. A ()
   2. B (int_k)
   3. B::add(int p)
   4. A::access (int p=5)
   5. A::add(int p)

   Begründung:
   a) B obj(2) -> ruft erstmal Konstruktor von A auf dann von B
   b) obj.add(3) -> ruft Funktion von B auf da als letztes auf Class B zugegriffen worden ist.
   c) obj.access -> ruft Funktion von A auf -> Der Aufruf von add(p) befindet sich daher in Class A und damit wird unabhängig vom Typ auch die

add Funktion von Class A aufgerufen (anders wäre es wenn add Funktion virtuell wäre).

Aufrufreihenfolge für Aufgabe c:

1. A()
2. B (int _k)
3. B::add(int p)
4. A::access (int p)
5. B::add (int p)

Begründung:

Die add Funktion ist nun virtuell und daher wird die Funktion aufgerufen zu dem der Typ gehört. In dem Fall B und damit ruft auch die access Funktion B::add() auf. Das die access Funktion virtuell ist, ist nicht von belang da diese in B gar nicht vor kommt.

c) Ausgabe:

a) 7
b) 10

Begründung:

1. A *p = new B(2); -> A() -> i=1 -> B (2) -> k=2
2. p->add(4) -> B::add(4) -> i=1+4+2 -> 7
3. p->access(1) -> !!!p=1!!! -> add (1) –> i=7+1+2 -> 10

Aufgabe 3)

a) Ergänzung: C() { cout << i << ".000"; } <- Im Grunde genommen ein fauler Trick aber i ist ein Integer und hat daher immer .000 -> man könnte auch einen Cast nach double oder float machen und dann so -> cout.precision(3); cout.setf(ios::fixed); cout << (float) i;
b) Ergänzung void quadrat (double &p) {} -> Hier wird Referenzparameter benützt -> Pointer wäre nicht möglich da Übergabe nicht quadrat(&a) ist. Sonst wäre es dann void quadrat (double *p) { *p= *p * *p;}
c) static double div (double P) {…}

Dadurch wird class A nun nur noch als Namespace angesehen. Aber Achtung dadurch wird die Funktion nicht mehr als Teil der Klasse betrachtet (!Sie hat nur den NAMEN der Klasse). Dadurch ist es auch nicht möglich die A::add() Funktion static zu setzen weil add() auf die Variable i auf diese Art nicht mehr zugreifen kann. Die Funktion kann dann nur noch über ein erzeugtes Objekt auf i zugreifen wie z.B. obj.i das Objekt müsste dann an die Funktion übergeben werden.

Aufgabe 4)

a) Ersetzungen: template <typename typ> class Medium {
typ sig; -> (statt int sig;)
bool has_sig (typ _sig){
Medium <int> m; -> (statt Medium m;)
b) Ersetzungen:
Medium <string> m;
cout << m.has_sig("5");
c) da cin >> für sig verwendet wird können auch nur die Datentyp verwendet werden die in der Klasse iostream in den Methoden definiert sind. Man könnte natürlich den Operator >> überladen das ist hier aber nicht gefragt.

Aufgabe 5)

a)

Überschrieben = gleiche Signatur in verschiedenen (abgeleiteten) Klassen
Überladen = verschiedene Signatur
Signatur = Datentyp und Anzahl der Parameter (Prototyp ohne Returntyp)

Daher:

1. Weder noch -> Compilerfehler
2. Überschrieben (falls Klasse abgeleitet ist)
3. Überladen
4. Überladen (falls Klasse abgeleitet ist)

b)

Lösung nicht sicher. Ich vermute Funktion der Klasse ios und daher auch ios::

c)

a) eher weniger könnte man aber so sehen.
b) ein Hauptgrund
c) das ist damit möglich aber ob es ein Grund war wieso dies eingeführt wurde weiß nur der Erfinder selber.