Fachhochschule Ingolstadt SS 2002
Studiengang Informatik

Prüfung Software-Entwicklung 2

Aufgabensteller: Dr. B.Glavina, Dr. S.Hahndel, Dr. J.Schweiger, Dr. M.Rudolph
Prüfungsdauer: 90 Minuten
Hilfsmittel: eigenhändig geschriebene Notizen (max. vier Seiten DIN A4)

Aufgabe 1 (Listen in C; etwa 20%)

Gegeben ist folgende Deklaration für den Typ `person`

```
struct person {
     int nummer;
     struct person *next;
};
```

a) Erweitern Sie den Typ so, dass man darin zusätzlich auch noch Folgendes speichern kann:
- einen (maximal 50 Zeichen langen) Namen,
- einen Zeiger auf eine (irgendwo sonst gespeicherte) Zeichenkette, die eine Zugehörigkeit zu einer Gruppe bezeichnet, und
- eine Datumsangabe, die das Eintrittsdatum der Person beschreibt.

Führen Sie für die Datumsangabe einen eigenen Typ `date` ein.

b) Schreiben Sie eine C-Funktion `print_date`, die eine Datumsangabe als Parameter bekommt und im Format tag-monat-jahr ausdruckt, wobei
- tag numerisch ohne führende Nullen ist
- monat die ersten drei Zeichen des Monatsnamens sind
- jahr numerisch mit vier Stellen ist

Beispiele: 18-Jul-2002, 1-Sep-1999.

c) Schreiben Sie eine C-Prozedur `mirror_list`, die eine als transienter Parameter übergebene (einfach verkettete) `person`-Liste "spiegelt", indem sie aus jeder Paarbeziehung a->b zwischen Elementen a und b der Liste die Beziehung b->a macht.
Beispiel: die Liste 1 -> 25 -> 8 -> 3 wird zu 3 -> 8 -> 25 -> 1.

d) Eine weitere Variable `liste2` ist als Anker einer zwei-elementigen `person`-Liste vorgegeben. Schreiben Sie die C-Befehle (Funktionsaufrufe) hin, um
- das in der zweiten Person (von `liste2`) enthaltene Eintrittsdatum mit Hilfe Ihrer Funktion aus Teilaufgabe b auszudrucken
- die Liste mit Hilfe Ihrer Funktion aus Teilaufgabe c zu spiegeln.

Aufgabe 2 (Klassen in C++; etwa 33%)

a) Schreiben Sie die Spezifikation und die Implementierung in C++ für eine Klasse `Point`, die Punkte in der Ebene behandelt.
Die Datenelemente (Koordinaten `px` und `py`) sind nicht direkt von außen greifbar.
Folgende Methoden sollen öffentlich verfügbar sein:

- ein Konstruktor zur vollständigen Initialisierung eines Punktes
- eine Prozedur `move` mit zwei Parametern `x` und `y` zum Verschieben des Punktes
- eine Funktion `distance`, die den (euklidischen) Abstand zwischen zwei Punkten liefert (zur Erinnerung: Abstand zwischen (a, b) und (c,d) ist $\sqrt{(a-c)^2+(b-d)^2}$)
- zwei Zugriffsfunktionen `x` und `y`, die jeweils die entsprechende Koordinate liefern

b) Schreiben Sie die Spezifikation und die Implementierung für die Klasse `Polygon`, die ebene Vielecke (mit maximal 10 Ecken) beschreibt. Realisieren Sie die Liste für die Eckpunkte als Reihung (`vertices`). An Methoden soll öffentlich verfügbar sein:

- ein Konstruktor zur Initialisierung des Vielecks mit einer `Point`-Reihung
- eine Prozedur `move` mit zwei Parametern `x` und `y` zum Verschieben des Vielecks (also aller enthaltenen Punkte)
- eine Funktion `perimeter`, die den Umfang des Vielecks liefert

Nutzen Sie bei der Programmierung die bereits in der vorigen Teilaufgabe erstellte Klasse `Point`!

c) Schreiben Sie die Spezifikation und die Implementierung für die Klasse `Box`, die achsenparallele Rechtecke beschreibt. Boxen sind Vielecke, deshalb wird `Box` von `Polygon` abgeleitet. Zusätzlich zu berücksichtigen ist:

- der Konstruktor bekommt nur zwei Punkte (den linken unteren und den rechten oberen, woraus sich die beiden anderen ergeben)
- der Umfang soll effizienter berechnet werden (Formel: `2*(side1+side2)`)
- eine zusätzliche Funktion `diameter`, die die Diagonale der Box liefert

Aufgabe 3 (Klassen und Polymorphie; etwa 37%)

a) In einem Computer-Spiel kommen verschiedene Kreaturen vor, für die je eine Klasse deklariert werden soll: `Dragon`, `Griffin` und `Rabbit`. Dazu gibt es noch die (Ober-)Klassen `Monster`, `Animal` und `Creature`.
Skizzieren Sie die Klassenhierarchie (mit Eigenschaften und Methoden) zunächst graphisch (noch kein C++) nach folgenden Vorgaben:

- Jedes `Monster` ist ein `Creature` (=Wesen).
- `Dragon` und `Griffin` sind `Monster`.
- Alle Wesen haben (nicht-öffentliche) Namen und Energie-Zähler.
- Alle Wesen können angegriffen werden (Methode `getAttack` regelt die Wirkung des Angriffs)
- `Monster` haben einen Wut-Zähler und können zurückschlagen (Methode `react`)
- Jedes einzelne `Monster` nimmt Angriffe verschieden auf und schlägt verschieden zurück (Drachen sind nicht nur gegen Feuer immun, es erhöht sogar ihre innere Energie ...).

- Jedes `Animal` ist ein `Creature` und hat einen (nicht-öffentlichen) Freundlichkeits-Zähler.
- `Rabbit` ist ein `Animal` und hat eine Methode `getAttack`.

Soweit benötigte Einzelheiten durch die Vorgaben nicht festgelegt sind, dürfen Sie selbst (sinnvolle!) Annahmen treffen.

b) Im Folgenden ist grob der Code des Spiels vorgegeben. Das Spielfeld ist ein Raster von Zellen, in denen sich Wesen befinden können. Die Spielerin befindet sich auch in einer Zelle und kann verschiedene Aktionen ausführen, von denen uns hier nur eine einzige interessiert: die Rundumschlag-Attacke auf alle Wesen in ihrer Nachbarschaft.
Sehen Sie nun den Code an und versuchen Sie ihn zu verstehen.

```
//
// Klassendefinitionen hier weggelassen
//
int main()
{
  Creature *area[70][20]; // Spielfeld
  int playerX, playerY; // Koordinaten der Spielerin
  bool stop = false; // Hilfsvariable
  char cmd[10]; // Hilfsvariable
  int i, j, k; // Hilfsvariable
  for (i=0; i<70; i++)
    for (j=0; j<20; j++)
      area[i][j] = NULL; // Feld leer machen

  // Ausbringen der Kreaturen ins Spielfeld:
  area[17][3] = new Griffin();
  area[15][2] = new Dragon();
  area[17][2] = new Rabbit();
  // usw.
  playerX = 16; playerY = 3; // hier stehen wir!

  // Hauptschleife:
  while (!stop)
  {
    // lies Kommando:
    gets(cmd);
    switch (cmd[0]) {
    case 'r':
      // Rundumschlag-Attacke schaedigt die ganze Umgebung;
      // 1) Sammeln aller Wesen in der Nachbarschaft:
      //
      // ... siehe Teilaufgabe c)
      //
      // 2) Verteilen des Angriffs auf die Nachbarschaft:
      //
      // ... siehe Teilaufgabe d)
      //
      // 3) Reaktion der Monster (NUR die schlagen zurueck!):
      //
      // ... siehe Teilaufgabe e)
      //
      break;
    case 's':
      stop = true;
      break;
```

```
    default:
      // nichts tun
      break;
    } // end switch
    // sonstige Dinge erledigen ...
  }
  return 0;
}
```

c) Geben Sie eine Befehlsfolge (Programmausschnitt) an, die ausgehend von der Spielerposition die unmittelbare Nachbarschaft (8 Felder im Umkreis) nach Wesen absucht, und diese (bzw. Zeiger auf diese) in der (eindimensionalen!) Reihung `neighbourhood` speichert (`n_cnt` speichert die Anzahl der gefundenen Wesen). Geben Sie auch geeignete Deklarationen für `neighbourhood` und `n_cnt` an. Hinweis: "Randeffekte" (Zelle der Spielerin hat weniger als 8 Nachbarzellen) dürfen Sie hier vernachlässigen.

d) Geben Sie einen Programmausschnitt an, der die in `neighbourhood` gespeicherten Wesen der Reihe nach attackiert. (Sind Ihre Deklarationen von Teilaufgabe a mit Ihrem Programmstück verträglich?)

e) Wir möchten die in `neighbourhood` gespeicherten Monster reagieren lassen (und zwar so einfach wie in Teilaufgabe d). Das Problem dabei ist: wir wissen nicht, welche von den Wesen Monster sind, und welche nicht. -- Deshalb müssen wir auch in der Klasse `Creature` eine Methode `react` spezifizieren, die dann in den Unterklassen bei Bedarf geeignet überschrieben wird.
Skizzieren Sie den veränderten Klassenbaum, wobei Sie Sich bei den Einzelheiten auf die Methode `react` beschränken können. (Wo wird sie spezifiziert? Wo wird sie implementiert?)

f) Geben Sie nun (in C++) die kompletten Spezifikationen aller Klassen an (Implementierungen sind hier nicht verlangt).

g) Falls wir eine weitere Art von Tieren ins Spiel bringen wollten, müssten wir die neue Klasse spezifizieren und die nötigen Methoden implementieren. Was sonst müssten wir noch tun? (In anderen Klassen? Im Hauptprogramm?)

Aufgabe 4 (Verschiedenes; etwa 10%)

a) "OOP ist schön und gut, aber mit herkömmlicher C-Programmierung komme ich schneller zum Ziel, und ausserdem laufen die Programme schneller!" -- Diskutieren Sie kurz die Richtigkeit und Wichtigkeit der beiden Teilaussagen (schnellere Entwicklung, schnellere Programme) im Allgemeinen, also abgesehen von irgendwelchen Spezialbedingungen.
Unter welchen Umständen könnte jemand obige Aussage gemacht haben?
(etwa 7-8 Punkte)

b) "Java-Programme laufen auf jedem Rechner." -- Stimmt das? Begründung! (Wenn ja, warum? Wenn nein, warum nicht?) (etwa 2-3 Punkte)