Prüfung Grundlagen Informatik 2

Aufgabensteller: Prof. Dr. B.Glavina, Prof. Dr. J.Schweiger
Prüfungsdauer: 90 Minuten
Hilfsmittel: vier Seiten Skript-Zusammenfassung

Aufgabe 1 (Klassen; etwa 30%):

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
- 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 einem Punktefeld
- eine Prozedur `move` mit zwei Parametern `x` und `y` zum Verschieben des Vielecks
- 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 2 (Ausnahmen; etwa 20%):

Gegeben sei folgender Programmauszug in C++ mit konventioneller Fehlerbehandlung:

```
#define M 100

class Speicher {
  private: char s[M+1];
  public:  int eintragen(int i, char c);
           int listeEintragen(char s[], int a, int e);
};

int Speicher::eintragen(int i, char c) {
  if (i<0) /* Fehler */ return -1;
  else if (i>M) /* Fehler */ return i-M;
  else { s[i] = c; return 0; }
}

int Speicher::listeEintragen(char s[], int a, int e) {
  int j, res;
  if (e>=(int)strlen(s)) /* Fehler */ return -2;
  for (j=e; j>=a; j=j-1) {
    res = eintragen(j, s[j]);
    if (res!=0) /* Fehler */ return res;
  }
  return 0;
}

void main() {
  char s[256]; Speicher sp; int a, e, res;
  scanf("%d %d", &a, &e); fflush(stdin); gets(s);
  res = sp.listeEintragen(s, a, e);
  if (res==-1)
    printf("Index negativ\n");
  else if (res>0)
    printf("Index um %d zu gross\n", res);
  else if (res==-2)
    printf("Zeichenkette zu kurz\n");
  printf("weitere Eintragungen\n");
}
```

Wandeln Sie das Programmstück so um, dass die konventionelle Fehlerbehandlung durch Ausnahmen-Behandlung (exception handling) in C++ ersetzt ist. Führen Sie dazu für jede Fehlerart eine globale Fehlerklasse ein. Fügen Sie zur Rückgabe der Höhe der Indexüberschreitung in die zugehörige Fehlerklasse ein geeignetes Attribut ein. Nach der Umwandlung des Programmstücks liefern die Methoden der Klassen kein Ergebnis zurück.

Aufgabe 3 (Listen; etwa 30%):

Gegeben ist die C-Deklaration des Verbundtyps `elem` wie folgt:

```
struct elem {
  char *name;
  unsigned int matnr;
  struct elem *next;
};
```

Schreiben Sie eine C-Funktion `merge_lists`, die drei auf obigem Verbundtyp basierende Listen als Parameter (a, b und c) übergeben bekommt und daraus eine einzige Liste konstruiert, welche als Funktionsergebnis geliefert wird.

Als Zusatzbedingung muss die Ergebnisliste nach der Matrikelnummer (`matnr`) aufsteigend sortiert sein; dazu dürfen (sollen!) Sie ausnützen, dass die zu verarbeitenden Listen bereits -- jede für sich -- diese Bedingung erfüllen.
Die Eingabelisten dürfen bei der Verarbeitung zerstört werden.

Aufgabe 4 (Zahldarstellung; etwa 20%):

Wir möchten in drei Schritten das Produkt aus 1000 und 0,001 berechnen, und zwar so wie es im Rechner geschieht.

a) Geben Sie folgende Darstellungen der Zahl 1000 an:
- reine Dual-Darstellung (Festkomma)
- normalisierte halblogarithmische Dual-Darstellung (Fließkomma)
- IEEE-short-real-Darstellung (32 Bit)

b) Geben Sie folgende Darstellungen der Zahl 0,001 an:
- reine Dual-Darstellung (Festkomma; es reicht die Bestimmung der Stelle mit der ersten 1 nach dem Komma)
- normalisierte halblogarithmische Dual-Darstellung (Fließkomma)
- IEEE-short-real-Darstellung (32 Bit)

c) Multiplizieren Sie nun die normalisierten halblogarithmischen Dual-Darstellungen der beiden Zahlen miteinander. Wandeln Sie das Ergebnis wieder in eine reine Dual-Darstellung.
Wie groß ist der absolute Fehler (Differenz zum erwarteten Ergebnis)? Woher kommt dieser Fehler?

(Ende des Aufgabentextes)