Prof. Dr. Stefan Brass
Dipl.-Inform. Annett Thüring
Institut für Informatik
MLU Halle-Wittenberg

13. Dezember 2010

Vorlesung "Objektorientierte Programmierung" — 2. Programmiertest (Aufgabe A) —

Hinweise/Regeln:

- Bearbeitungsdauer: 75 Minuten.
- Am Ende gibt es nur "bestanden" und "nicht bestanden", keine Punkte für partiell korrekte Lösungen.
- Sie dürfen bis zu 3 Blätter "Spickzettel"/"Quick Reference" verwenden, sowie ein Buch/Hefter (nicht zu groß, es muß noch auf den Tisch passen ohne zu stören).
- Eigene Notebooks, PDAs, etc. dürfen nicht verwendet werden. Mobiltelefone bitte ausschalten (oder mit der Aufsicht besprechen).
- Legen Sie bitte ein neues Verzeichnis `oop10/praxistest2` in Ihrem Homeverzeichnis an. Sie dürfen ein eventuell existierendes `Makefile` hineinkopieren oder Shellskripte zur Programmentwicklung. Ansonsten dürfen Sie keine weiteren Dateien kopieren oder öffnen, sondern nur den Quellcode des zu entwickelnden Programms.
- Kopieren Sie sich dann bitte die Datei `~brass/oop10/test2/p2a.cpp` in dieses Verzeichnis. Diese Datei enthält ein Hauptprogramm, das als Test dient. Ihre Aufgabe ist, an der markierten Stelle die in der Aufgabenstellung beschriebene Klasse einzufügen. Am Ende muss das Hauptprogramm genau der vorgegebenen Version entsprechen, zwischenzeitlich können Sie natürlich eigene Testaufrufe einbauen.
- Sie dürfen selbstverständlich die zur Programmentwicklung üblichen Programme verwenden (auch `man`), aber nicht auf das Internet zugreifen (z.B. kein Web-Browser, kein EMail-Programm).
- Die Homeverzeichnisse werden für Zugriffe von außen gesperrt. Falls Sie spezielle Zugriffsrechte gesetzt hatten, müssen Sie diese nach dem Test selbst wieder herstellen.
- Selbstverständlich dürfen Sie auch Microsoft Visual Studio oder eine andere IDE benutzen. Das abgegebene Programm muss aber unter Linux/g++ laufen.
- Tauschen Sie keinesfalls irgendwelche Dinge mit den Nachbarn aus. Notfalls rufen Sie eine Aufsichtsperson zur Kontrolle.
- Sie müssen Mindestanforderungen an den Programmierstil erfüllen, z.B. entsprechend der Programmstruktur einrücken, sinnvolle Variablennamen wählen und zum Verständnis notwendige Kommentare schreiben.
- Fragen Sie, wenn Ihnen die Aufgabe nicht klar ist! Melden Sie sich bitte auch, wenn es technische Schwierigkeiten mit Ihrem Rechner gibt.
- Wenn Sie an einer unverständlichen Fehlermeldung länger festhängen, können Sie probieren, zu fragen. Wir wollen aber nicht zu viele Tipps geben.

Aufgabe (Variante A)

Angenommen, Sie haben eine Datei, die alle ihre Kontobewegungen enthält, und Sie wollen die fünf größten Zahlungen daraus bestimmen. Die Datei ist sehr groß und Sie wollen sie auf keinen Fall komplett in den Hauptspeicher laden. Daher gehen Sie wie folgt vor: Sie gehen die Datei Posten für Posten durch, und verwalten mit Hilfe eines Objektes die fünf größten Zahlen, die bis zur aktuell gelesenen Zahl vorgekommen sind. In der Aufgabe sollen Sie lediglich eine Klasse `maxFive` entwickeln, welche die fünf größten Zahlen speichern kann. Die Dateizugriffe sind nicht Bestandteil dieser Aufgabe. Genauer soll die Klasse `maxFive` fünf natürliche Zahlen (alle ≥ 0) speichern und folgende Methoden haben:

- `store`, die einen Parameter `n` vom Typ `int` hat, und keinen Rückgabewert: Diese Funktion wird für jeden Eingabewert aufgerufen. Sie muss einen Wert nur speichern, wenn er zu den aktuell fünf größten gehört. Negative Eingabewerte sind unzulässig und sollen einfach ignoriert werden. Falls `store` für den gleichen Wert mehrfach aufgerufen wird, ist dieser Wert ggf. auch mehrfach zu speichern.
- `top`, mit einem Parameter `i` vom Typ `int`, der immer zwischen `0` und `4` liegt, und einem Rückgabewert vom Typ `int`. Diese Methode soll den `i`-ten Wert von oben liefern, also `top(0)` das Maximum der Menge, `top(1)` den zweitgrößten Wert, u.s.w. Falls bisher weniger als `i` Werte gespeichert wurden, oder der Parameter außerhalb des Bereichs 0–4 liegt, soll `-1` als Ergebnis geliefert werden.
- Außerdem müssen Sie selbstverständlich einen Konstruktor schreiben, der die Menge als leere Menge initialisiert.

Beispiel: Nach den Aufrufen `store(10)`, `store(8)`, `store(12)`, `store(5)`, `store(15)`, `store(14)`, `store(20)`, `store(15)` müsste `top(0)` den Wert `20` liefern, `top(1)` den Wert `15`, und `top(2)` auch den Wert `15` (dieser wurde ja zwei Mal gespeichert), ausserdem gelten `top(3)==14` und `top(4)==12`.

Beispiel-Algorithmus:

Sie können das Berechnungsverfahren frei wählen. Eine Möglichkeit wäre, sich ein Array mit 5 Elementen zu machen, und mit `-1` zu initialisieren. Das Array soll an der Position `i` den Wert `top(i)` enthalten. Ein zu speicherndes Element `n` vergleicht man dann nacheinander mit den Array-Elementen an Position `0`, `1`, u.s.w. Ist das zu speichernde Element `n` größer als das Element im Array an Position `i`, wird es an diese Position gespeichert. Den alten Wert im Array muss man sich vorher merken, und dann anstelle von `n` versuchen, in den Rest des Arrays einzutragen. Das ist dann immer an der jeweils nächsten Position erfolgreich, da die Elemente ja schon sortiert sind. Man könnte also auch gleich den Rest des Arrays einfach eine Position nach hinten/oben verschieben. Das letzte Element des Arrays fällt dabei heraus, es gehört jetzt ja nicht mehr zu den fünf größten.