Beispielangaben zu
OOP (Objekt-Orientierte Programmierung)

SS 2009

Beispiel 4

Relevante Themen:

- Generizität
- Ableiten von generischer Klasse

Design und Implementierung:

Mit dieser Aufgabe soll speziell das Ableiten von generischen Klassen (Templates) geübt werden. Dazu sind die Klassen des CPU-Simulator myCPU aus Beispiel 3 als generische Klassen umzuschreiben, sodass anstelle von CDat eine alternative Klasse (mit beliebigen Rechenregeln und Zahlenformat) benutzt werden kann. Die Rechenregeln bestimmen die Berechnung neuer Werte und das Zahlenformat bestimmt die Parse-Funktion, mit welcher eine Zahl der Speicherbelegungsdatei zu interpretieren ist.

Neben der Klasse CDat sollen nun zwei weitere Klassen als Rechentypen implementiert werden:

- CDatSet:
  Diese Klasse benutzt als Rechenregeln analog wie CDat den C++ Datentyp int. Allerdings wird ein anderes Parserformat für die Speicherbelegungsdatei benutzt: eine Zahl n wird durch n-mal den Buchstaben 'o' (klein O) geschrieben. Damit können allerdings nur positive Zahlen dargestellt werden (negative Zahlen können erst durch anschließende Negation im Programm erreicht werden).
  Beispiel für eine Speicherbelegungsdatei mit den Zahlen 6 und 8:
  ```
  oooooo
  oooooooo
  ```
- CDatN
  Diese Klasse kann im Konstruktor mit einem Parameter int width initialisiert werden (der Default-Konstruktor ohne diesen Parameter ist zu deaktivieren, siehe Folien). Der Parameter width gibt an, mit wie vielen Bitstellen gerechnet werden soll, wobei die minimale gültige Stellenzahl 2 ist und die maximale Stellenzahl 32 ist. Wenn in der Speicherbelegungsdatei Zahlen stehen, die eine größere Stellenzahl als gegeben erfordern würde, so werden die höherwertigen Bits weggeschnitten. Die Rechenregeln sind auf die entsprechende Stellenzahl anzupassen.
  Beispiel: width=5 (5bits)
  Damit wird aus einer Zahl 150 (entspricht der Binärzahl 10010110) des Speicherbelegungsfile die Zahl 22 (entspricht der Binärzahl 10110). Die Beschränkung auf eine bestimmte Bitzahl kann in C++ ganz einfach mit den

Bitmanipulationsoperatoren gemacht werden. Beispielsweise kann die Zahl 150 folgendermaßen auf 5 Bits beschränkt werden: var = ((1<<5)-1) & 150.

Die Synopsis des Programms ist folgende:

myCPU [-f⟨format⟩] -c ⟨programfile⟩ [-m ⟨memfile⟩]

Das optionale Argument –f⟨format⟩ wird benutzt, um das Rechenformat des CPU-Simulators umzustellen. Wird das Argument nicht angegeben, so soll die Rechenarithmetik von Beispiel 3 (Klasse CDat) bentutzt werden. Wird die Option angegeben, so hängt die zu wählende Rechenoperation vom Formatbezeichner. Ist der Formatbezeichner eine Zahl n von 2 bis 32, so soll die Klasse CDatN(n) benutzt werden. Ist der Formatbezeichner der Buchstabe ‚s', so soll die Klasse CDatSet als Rechenformat benutzt werden.

Die in der Spezifikation nicht genauer ausgeführten Teile können selbst sinnvoll ausgestaltet werden, wobei jedoch die folgenden Anforderungen einzuhalten sind.

Anforderungen:

Kommentieren Sie den Code, um das Verstehen des Codes zu vereinfachen. Zusätzlich soll zu Beginn jeder Klasse eine Klassenbeschreibung stehen. Am Beginn jeder Datei soll außerdem Name, MNr, KZ der Gruppenmitglieder sowie die Beispielnummer stehen. Ein nicht kommentierter Code wird negativ bewertet!

Ausnahmen (Exceptions) müssen immer abgefangen werden, sofern möglich. Überlegen Sie sich dabei aber auch, welche Ausnahmen zu einer Programmbeendigung führen müssen und in welchen Fällen es ausreicht, eine Warnung bzw. Fehlermeldung auszugeben. Bei jeder Ausnahmebehandlung ist auf jeden Fall eine Fehlermeldung auf die Fehlerausgabe (cerr) auszugeben. Als Orientierung seien einige mögliche Quellen für Ausnahmen aufgezählt: Speichermangel, inkorrekter Input, Schreiben auf Datei fehlgeschlagen, etc.

Schreiben Sie klare Zusicherungen an den passenden Stellen im Programm. Dort, wo man Zusicherungen kompakt als Code formulieren kann, wird empfohlen, die Zusicherungen zusätzlich als „assert()" hinzuschreiben, damit der Compiler Code generieren kann zur automatischen Prüfung der Zusicherungen zur Laufzeit. Mit der Compileroption „-DNDEBUG" kann man den fertig ausgetesteten Code generieren sodass „assert()" keinen Overhead mehr verursacht.

Benutzen Sie zur Programmentwicklung auch ein Makefile, welches zumindest die Targets „clean", „all" und „run" enthält. Mittels „make run" soll das Programm direkt aufgerufen werden können (mit sinnvollen Argumenten versehen).

Es sind die Programmierrichtlinien der LVA einzuhalten.

Schreiben Sie zu dem Programm ein Protokoll, das folgende Informationen beinhaltet:

- Identifikation (Name, MNr, Kz)
- Aufgabenstellung (kann direkt von dieser Angabe übernommen werden, Beispiele mit Grafiken sind nicht zu übernehmen)
- Klassendiagramme (inkl. Beschreibung)
- Schematische Beschreibung der Allokation und Freigabe von Ressourcen
- Schematische Beschreibung der Ausnahmebehandlung (mögliche Ursachen von Exceptions, sowie die Strategie bei deren Behandlung (u.a., ob lokal oder eher global behandelt))
- Dokumentation des Arbeitsaufwandes
- Dokumentation von ev. aufgetretenen Problemen
- Kommentierter Programmcode

Der Umfang des Protokolls soll (ohne Mitzählen der Codeseiten) zwischen 5 bis 10 Seiten sein.

Abgabe:

Legen Sie ein Verzeichnis „Beispiel4“ an, in dem Sie alle Ihre Programmdateien hineingeben. Geben Sie auch das Protokoll in dieses Verzeichnis hinein.
Packen Sie das Verzeichnis folgendermaßen ein:

```
tar –zcvf Beispiel4.tgz Beispiel4
```

und geben Sie dieses Archivfile elektronisch ab. Ohne diese elektronische Abgabe sowie das Mitbringen eines schriftlichen Ausdruckes des Protokolls kann die Abgabe nicht positiv gewertet werden.

Der relevante Stoff für das Abgabegespräch sind Kapitel 1, 2 und 3 des Skriptums. Es ist dabei auch notwendig, dass Sie den Text, wo zutreffend, für die Lösung des Beispiels anwenden.