Programmierkurs I (Inf., W-Info.), Übungsblatt 12

Claus/Weicker, Wintersemester 03/04 Abgabetermin: 29.01.2004, 23:59 Uhr

Bitte beachten Sie die Hinweise von Aufgabenblatt 8 sowie die Angaben in den jeweiligen Spezifikationsdateien $*$.ads.

Bitte denken Sie daran, dass für die Vergabe eines Scheins mindestens 40 Punkte aus den letzten 4 Aufgabenblättern benötigt werden! Das letzte Blatt wird Nummer 13 sein.

Aufgabe 1: Exceptions (leicht) 4 Punkte

In vielen Aufgaben der letzten Übungsblätter mussten fehlerhafte Eingaben über spezielle Status-Ausgabeparameter oder ausgezeichnete Rückgabewerte der aufrufenden Programmeinheit mitgeteilt werden. Damit muss nach dem Aufruf einer Prozedur oder Funktion zunächst geprüft werden, ob dieser erfolgreich war. Falls ja, kann anschließend mit dem regulären Programmablauf fortgefahren werden. Ein alternativer, wesentlich eleganterer Mechanismus für die Fehlerbehandlung stellt das Konzept der *Exceptions* zur Verfügung, das hier zumindest ansatzweise kurz erläutert wird.
Es gibt Standard-Exceptions, wie zum Beispiel `Constraint_Error` u.a. bei einer Division durch 0. In eigenen Paketen können jedoch auch neue Exceptions deklariert werden

```
MeineException: exception;
```

In den Prozeduren und Funktionen des Pakets kann nun im Fehlerfall, z.B. bei einer fehlerhaften Eingabe, eine Exception „werfen“:

```
raise MeineException;
```

Dadurch wird die Ausführung des aktuellen Programmblocks (i.d.R. der Prozedur oder Funktion) abgebrochen.
Im Programmblock, in dem die Exception geworfen wurde, oder in der aufrufenden Programmeinheit (dies kann bis zum Hauptprogramm weitergehen) kann die Exception nun aufgefangen werden und entsprechende Maßnahmen eingeleitet werden. Diese werden am Ende des Programmblocks angegeben

```
begin
  -- Programmtext einschließlich besagten
  -- Prozedur-/Funktionsaufrufs
exception
  when MeineException =>
     -- Anweisungen zur Fehlerbehandlung
end;
```

Nach der Fehlerbehandlung fährt das Programm hinter dem Programmblock fort, der die Exception aufgefangen hat, d.h. beim Programmblock einer Prozedur/Funktion mit der Rückgabe von Werten, bei einem Declare-Block mit der nächsten Anweisung. Bei einer Funktion ist häufig eine Return-Anweisung in der Fehlerbehandlung sinnvoll. Dort können auch andere Exceptions wieder nach außen gereicht werden.
Als erstes Beispiel wird hier die Aufgabe 1 vom Übungsblatt 6 wieder aufgegriffen. Ein Zahlenpaar $(a, b) \in \mathbb{N} \times \mathbb{N}$ lässt sich als eine einzelne Zahl verschlüsseln und darstellen, indem die Funktion

$$f(a, b) \quad = \quad 2^a 3^b$$

angewandt wird.
Schreiben Sie eine Funktion

```
function Verschluesseln (A, B: Natural) return Natural;
```

für die Verschlüsselung, wobei bei nicht mehr als Natural darstellbaren Zahlen die Exception `Ungueltige_Eingabe` geworfen werden soll.
Schreiben Sie eine Prozedur

```
procedure Entschluesseln (Zahl: in Natural; A, B: out Natural);
```

für die Entschlüsselung, wobei bei einem nicht entschlüsselbaren Wert ebenfalls die Exception `Ungueltige_Eingabe` geworfen werden soll.
Wie üblich ist die Spezifikation in einer Datei `verschluessel.ads` auf der Internetseite der Veranstaltung erhältlich. Geben Sie Ihre Implementation als `verschluessel.adb` ab.
Es gibt 2 Punkte für korrekte Funktionalität, 1 Punkt für die Beschreibung des Konzepts und 1 Punkt für die Einhaltung der Programmierrichtlinie.

Aufgabe 2: Stack (mittel) 7 Punkte

Schreiben Sie ein Paket, welches einen Datentyp `Integer_Stack` für einen Stack (Keller, LIFO-Liste) implementiert. Die Elemente im Stack sollen dabei vom Typ `Integer` sein und der Stack soll beliebig viele Elemente aufnehmen können. Die folgenden Funktionen und Prozeduren sind für den Stack zu implementieren:

```
function  Ist_Leer (S: Integer_Stack) return Boolean;
procedure Leere (S: in out Integer_Stack);
procedure Push (S: in out Integer_Stack; X: in Integer);
procedure Pop (X: out Integer; S: in out Integer_Stack);
function  Top (S: Integer_Stack) return Integer;
```

Ferner ist die Exception `Stack_Error` vorgesehen, die bei den Operationen `Top` und `Pop` bei einem leeren Stack geworfen wird, sowie bei der Operation `Push`, falls kein Speicher mehr zur Verfügung steht. (Dann wird vom Laufzeitsystem üblicherweise beim Aufruf eines `new` die Exception `Storage_Error` geworfen.)
Implementieren Sie den Datentyp in der Datei `intstack.adb` als einfach verkettete Liste. Dazu gehört die Deklaration des Datentyps für die einzelnen Elemente des Stacks sowie die Zugriffsfunktionen. Die Spezifikationsdatei `intstack.ads` ist auf der Internetseite verfügbar und darf nicht verändert werden.
Es gibt 3.5 Punkte für korrekte Funktionalität, 1.5 Punkte für die Beschreibung des Konzepts und 2 Punkte für die Einhaltung der Programmierrichtlinie.

Aufgabe 3: Postfix (schwer) 9 Punkte

Mathematische Terme werden gewöhnlich in der Infix-Notation aufgeschrieben, d.h. der Operator wird zwischen die Operanden geschrieben, wie z.B. `2 + 4`. Bei der Postfix-Notation wird der Operator hinter die Operanden geschrieben, das Beispiel von eben wird zu `2 4 +`. Ein Vorteil dieser Notation ist, dass sie ohne Klammern auskommt, die bei der Infix-Notation wegen der Punkt-vor-Strich-Regel bisweilen notwendig sind. So ist der Infix-Term `2 * ( 3 + 4)` bedeutungsgleich mit dem Postfix-Term `2 3 4 + *`.
Schreiben Sie eine Funktion

```
function Auswertung (Ausdruck: String) return Integer;
```

die in einer beliebig langen Zeichenkette einen Ausdruck in Postfix-Notation erhält, der ausgewertet werden soll. Das Ergebnis wird als Funktionswert zurückgegeben.
Die Zeichenkette kann natürliche Zahlen, `+`, `*`, `-` und Leerzeichen enthalten. Beachten Sie dabei, dass Zahlen aus mehr als einer Ziffer bestehen können. Eine gültige Eingabe ist also auch ␣␣3␣1␣+100␣␣-␣ (wobei ␣ für ein Leerzeichen steht).
Benutzen Sie den Stack aus Aufgabe 2, um den Ausdruck auszuwerten. Grundidee ist hierbei, die Zeichenkette von links nach rechts zu lesen, jede gelesene Zahl auf den Stack zu legen und eine gelesene Operation auf die obersten beiden Werte im Stack anzuwenden, dabei die beiden Werte zu entfernen und das Resultat der Operation auf den Stack zu legen.
Falls es sich bei der Zeichenkette um keinen gültigen Ausdruck in Postfix-Notation handelt, soll die Exception `Ungueltig` geworfen werden. Treten beim Einlesen oder beim Auswerten Zahlen ausserhalb des Wertebereichs von `Integer` auf, soll die Exception `Overflow` geworfen werden.

Geben Sie die Implementation in der Datei `postfix.adb` ab. Die dazugehörige Datei `postfix.ads` ist auf der Programmierkursseite im Internet erhältlich. Ihre Abgabe soll allein aus `postfix.adb` bestehen, da sie mit unserer Version von `stack.adb` getestet wird.
Es gibt 4.5 Punkte für korrekte Funktionalität, 2 Punkte für die Beschreibung des Konzepts und 2.5 Punkte für die Einhaltung der Programmierrichtlinie.

Hinweise

- Zur Abgabe wird das eClaus-System verwendet:
  `http://eclaus.informatik.uni-stuttgart.de`
- Die Abgabe zu jeder Teilaufgabe besteht aus einem kompilierbaren Ada-Quelldatei. In jeder Aufgabe wird ein Dateiname vorgegeben. Verwenden Sie diesen bitte für das Hauptprogramm. Auch sind in der Aufgabe Angaben zu Ein- und Ausgabetexten sowie zur Formatierung der Ausgabe enthalten. Bitte folgen Sie diesen Angaben so genau wie möglich.
- Beachten Sie beim Programmieren bitte die folgenden Hinweise („kleine Programmierrichtlinie“).
  - Halten Sie einzelne Bestandteile überschaubar, z.B. indem Sie nur eine Anweisung pro Zeile schreiben, sowie pro Zeile höchstens 80 Zeichen, höchstens 40 Zeilen pro Prozedur/Funktion, nicht mehr als 800 Zeilen pro Datei und höchstens 5 Parameter bei Prozeduren/Funktionen benutzen.
  - Bezeichner sollen selbsterklärend und maximal 15 Zeichen lang sein. Bezeichner enthalten nur Buchstaben, den Bindestrich oder den Unterstrich.
  - Durch Einrückung um 2 Zeichen soll die logische Gliederung eines Programms verdeutlicht werden. Beispielsweise wird der Anweisungsteil einer IF-Verzweigung eingerückt, während „`end if;`“ nicht mehr eingerückt wird. Auch auf der nächsten Zeile fortgesetzte Zeilen werden um 2 Zeichen eingerückt.
  - Zu Beginn des Programms muss in Kommentaren das Konzept und die Lösungsidee des Programms ausführlich erläutert werden.
  - Auch im Programmtext sind Kommentare einzufügen, um den Code zu erläutern.
- Beachten Sie, dass jede Abgabe individuell vom jeweiligen Studierenden erstellt werden muss. Werden von den Tutoren Plagiate erkannt, d.h. exakte oder leicht modifizierte Kopien, werden für die Aufgabe keine Punkte vergeben. Falls Sie die Lösung der Aufgaben vor der Abgabe in Gruppen besprechen, achten Sie darauf, dort nur das generelle Konzept abzuklären und die Programmierung jedem selbst zu überlassen.
- Bei Fragen wenden Sie sich bitte an Ihren Tutor oder an die Übungsleitung:
  `Karsten.Weicker@fmi.uni-stuttgart.de` oder Tel. 7816-337
- Weitere Veranstaltungshinweise einschließlich der Übungsblätter finden Sie unter:
  `http://www.informatik.uni-stuttgart.de/ifi/fk/lehre/ws03-04/ada95/`