LEHRSTUHL FÜR INFORMATIK III
RWTH Aachen
Univ.-Prof. Dr.-Ing. Manfred Nagl,
Christian Fuß, Ulrike Ranger

Die SWT-Programmiersprache Ada 95
Übung zur Vorlesung
SS 2005
23.06.2005

Übungsblatt 05

Allgemeine Hinweise

Bitte geben Sie in jeder Quellcode-Datei im Kommentar auch Ihre Namen an.

Aufgabe 14 Typen in Ada (4 Punkte)

Für die Kodierung der Nachrichten des CAN-Controllers wurde ein Integer-Typ benötigt, der sich auf die Range 0..65535 erstreckt. Die folgende Definition ist fehlerhaft:

```
TYPE can_int IS integer RANGE 0..65535;
```

a) Was ist daran fehlerhaft?

Zeigen Sie, wie man daraus jeweils folgende gültige Deklarationen machen kann und erläutern Sie die Unterschiede dieser Möglichkeiten:

b) Integer-Typdeklaration

c) abgeleitete Typdeklaration

d) Untertypdeklaration

Aufgabe 15 Zeiger- und Zugriffstypen (5 Punkte)

Nachfolgend werden drei Zeigertypen definiert, erläutern Sie deren Unterschiede.

```
TYPE int_access IS ACCESS integer;
TYPE int_access_all IS ACCESS ALL integer;
TYPE int_access_const IS ACCESS CONSTANT integer;
```

Erläutern Sie die Bedeutung der Schlüsselwörter **ALIASED** und **ALIASED CONSTANT**, wie sie nachfolgend benutzt werden.

```
a: integer := 7;
b: ALIASED integer := 7;
c: ALIASED CONSTANT integer := 8;
p: int_access;
q: int_access_all;
r: int_access_constant;
```

Welche der folgenden Zuweisungen sind zulässig? Begründen Sie ihre Antwort.

```
a := b;                 c := a;                 p := a'ACCESS;
q := c'ACCESS;          r := NEW integer;       p := b'ACCESS;
b := c;                 a := p;                 q := p;
r := b'ACCESS;          r.all := a;             p := q'ACCESS;
```

Datenstrukturen, die Zeiger verwenden, werden oft mit dem aus der starken Verwendung von unbedingten **goto**-Sprüngen resultierenden Spaghetti-Code verglichen. Wie kommt es zu diesem Vergleich? Welche Argumente fallen Ihnen ein, warum dieser Vergleich hinkt? Was wird in Ada95 getan, um auch Zeiger verwendende Programme „sicher“ zu machen?

Aufgabe 16 Unterprogramme als Formalparameter (2 Punkte)

In Ada83 gibt es keine Typen für Unterprogramme. Als Konsequenz daraus können Unterprogramme nicht als Formalparameter in anderen Unterprogrammen verwendet werden. Welche Probleme können bei Verwendung von Unterprogrammen als Formalparameter auftreten (denken Sie z.B. an Sichtbarkeit/Gültigkeit)?

Ada95 bietet **access-to-subprogram** Typen, also Zeigertypen, die auf Unterprogramme verweisen. Was könnte die Sprachentwickler bewogen haben, ihre Meinung zu Unterprogrammen zu ändern?

Aufgabe 17 Verbesserung des bisherigen CAN-Treibers (3 Punkte)

Definieren Sie folgende Typen für den CAN-Treiber, die an der Schnittstelle zu sehen sein sollen:

- **can_id**: Ganzzahlen von 0 bis 65535
- **can_length**: Ganzzahlen von 0 bis 7
- **can_int**: Ganzzahlen von 0 bis 65535

Begründen Sie die Wahl ihrer Implementierung. Stellen Sie Ihre bisherige Implementierung so um, dass sie diese Typen nutzt.

Ändern Sie Ihre Implementierung der Message-Queue so, dass sie einen Verbund mit einer Diskriminanten nutzt, die die Größe der Queue angibt. Instanziieren Sie die Queue mit einer Größe von 16.

Aufgabe 18 Empfang von CAN-Nachrichten (8 Punkte)

Beim Empfang einer CAN-Nachricht durch die Hardware des CAN-Controllers wird ein Interrupt im Betriebssystem des Steuergeräts ausgelöst. Für diesen Interrupt wird im Betriebs-

system durch den CAN-Treiber eine sog. *Interrupt Service Routine* (ISR) registriert. Implementieren Sie eine parameterlose Methode isr_can_reception mit folgender Funktionalität (die Registrierung beim Betriebssystem wird später vorgenommen):

a) Auslesen der aktuell empfangenen Nachricht aus dem Feld **`can_controller.received`**.

b) Puffern der Nachricht in einem Feld, aus dem mit **`get_recent(0)`** die aktuellste und mit **`get_recent(15)`** die älteste gepufferte Nachricht ausgelesen werden kann. Implementieren Sie auch diese Methode. Wählen Sie eine **`array`**-Implementierung, die mit Indizes auf dem Array arbeitet, um immer die älteste Nachricht zu verdrängen.

c) Nach der Pufferung sollen alle Hooks aufgerufen werden. Ein *Hook* ist eine Methode, die von einer Anwendung, die den Treiber nutzt, mit der Methode **`register_reception_hook`** registriert wurde und auf den Empfang einer CAN-Nachricht reagieren soll. Die Hooks werden bei der Registrierung als **`access-to-subprogram`** Typen übergeben und sollen in einer einfach verketteten Liste gehalten werden. Implementieren Sie neben dem Aufruf aller Hooks auch die Hook-Liste und die Methode register_reception_hook.

***Abgabe**: **Donnerstag**, 30.06.2005*

Sie können Ihre Lösung zum obigen Termin in Papierform in der Vorlesung oder in elektronischer Form per eMail an ada@i3.informatik.rwth-aachen.de abgeben.

Bitte vermerken Sie in jedem Fall die Namen und Matrikelnummern aller beteiligten Personen (maximal 3), beim Versenden per eMail auch alle eMail-Adressen. Abgaben in elektronischer Form können ausschließlich in den Dateiformaten *Plain-Text oder PDF* erfolgen.

Aktuelle Informationen zur Vorlesung finden Sie auf den Webseiten des Lehrstuhls unter http://www-i3.informatik.rwth-aachen.de.