Programmierparadigmen

Sommerersemester 2017
Martin Wittiger, Felix Krause, Timm Felden

3. Übung

Abgabe bis 17. Mai um 4:44
Beachten Sie die Abgabehinweise auf der Vorlesungswebseite!

Aufgabe 3.1 ..... 2 Punkte

Dateiname: `bitvector.pdf` ★☆☆

Ein Haupspeicher sei 100 Worte groß. Er wird mit Bitvector-Speicherverwaltung verwaltet. Ein Programm hat folgendes Speichernutzungsverhalten:

- Allokiere `A` (Größe 1).
- Allokiere `B` (Größe 2).
- Allokiere `C` (Größe 13).
- Allokiere `D` (Größe 15).
- Allokiere `E` (Größe 10).
- Allokiere `F` (Größe 15).
- Deallokiere `A`
- Deallokiere `B`
- Allokiere `Q` (Größe 42).
- Deallokiere `C`
- Deallokiere `D`
- Allokiere `P` (Größe 9).

Geben Sie für alle Allokationen die Speicherposition an, die genutzt wird, ...

1. ... wenn die Vergabestrategie First-Fit ist.
2. ... wenn die Vergabestrategie Next-Fit ist.

Hinweis: Das erste Objekt wird in beiden Fällen ab Adresse 0 im Speicher liegen.

Aufgabe 3.2 ..... 2 Punkte

Dateiname: `zeiger.pdf` ★☆☆

Beim Aufrufen einer Funktion mittels Dereferenzierung eines Funktionszeigers können (je nach Sprache) Probleme auftreten. Charakterisieren Sie kurz drei solche Probleme und schreiben Sie Pseudo-Code, der ihr Auftreten zeigt.

Unbenotete Bonusaufgabe: Implementieren Sie jedes der drei Programme in C oder Ada. Probieren Sie aus, was passiert, wenn Sie die Programme starten.

Aufgabe 3.3 ..... 3 Punkte

Dateiname: `firma.h` ★★☆

Auf der Vorlesungsseite finden Sie das Skelett eines C-Programms. Dabei geht es um eine Firma und ihren Gewinn. In unserem Modell macht eine Firma jedes Jahr Gewinn. Dieser ist eine ganze Zahl. Ein Hellseher weiß schon im Voraus, wieviel Gewinn die Firma jedes Jahr erwirtschaften wird. Er möchte zu einem Zeitpunkt Anteile kaufen und diese dann zu einem späteren Zeitpunkt wieder verkaufen. Dabei soll die Summe der dazwischen liegenden Gewinne maximal sein. Sein Investment kann auch den gesamten Zeitraum überdauern oder keine einzige Ausschüttung enthalten.

Bei seinem Investment benötigt er Ihre Hilfe. Sie sollen die Funktion `gewinn` vervollständigen. Sie geben nur die Datei `firma.h` ab. Die Funktion bekommt ein »Array«, das die Gewinne enthält, übergeben und berechnet was der Hellseher mit seiner Strategie erhalten wird.

In den folgenden Beipielen steht in der ersten Zeile der Aufruf des fertigen Programms. In der zweiten Spalte das Ergebnis. In der dritten Spalte, zur Veranschaulichung, eine Folge von Auszahlungen, die das Ergebnis realisieren.

```
firma 11 -3 -4 -1 8  |  11  |  11
firma -7 1 2 3       |   6  |  1 2 3
firma 2 4 -5 1 7     |   9  |  2 4 -5 1 7
firma -1 -2 -5       |   0  |
firma 4 0 0 -2 3     |   5  |  4 0 0 -2 3
firma 2 3 4 -5 1     |   9  |  2 3 4
```

In der Datei `firma.c` sind die Makros `max` und `min` definiert. Sie können diese gerne verwenden. Ich möchte aber ausdrücklich darauf hinweisen, dass sie in gewissem Sinne nicht typ-sicher sind. Sie können davon ausgehen, dass die Summe der Beträge aller Gewinne kleiner als 2^{60} ist.

Aufgabe 3.4 ..... 6 Punkte

Dateiname: `suv.adb`

Sie erinnern sich an Peter und sein dieselbetriebenes SUV? In dieser Aufgabe geht es um ein ganz ähnliches Problem.

Peter startet diesmal stets mit einem voll getankten Fahrzeug. Die Etappen sind wieder gegeben. Sie sollen diesmal die minimale Kapazität berechnen, mit der sich die Etappen mit einer gegebenen maximalen Anzahl an Tankstopps bewältigen lassen.

Angenommen, die Etappen haben Längen von 11, 4, 6, 2, 8 und 9. Ist dann nur ein Tankstopp erlaubt, braucht der SUV eine Reichweite von 21. Er kann dann nach der dritten Etappe einen Tankstopp einlegen.

Sind hingegen zwei Tankstopps erlaubt, reicht eine Reichweite von 16. Wenn drei Tankstopps erlaubt sind reicht sogar eine Reichweite von 11.

Implementieren Sie die Funktion `Solve` in der zur Verfügung gestellten Datei. Sie können auch Hilfsfunktionen anlegen. Die Musterlösung tut dies auch. Achten Sie darauf, die Ausgabe und Eingabe nicht zu verändern. In den Eingabedateien ist nicht die Zahl der Stopps, sondern die Zahl der Intervalle zwischen den Stopps angegeben. Lassen Sie sich dadurch nicht verwirren.

Stellen Sie durch eigene Tests sicher, dass Ihr Programm funktioniert, bevor Sie es abgeben.

Sie können folgende Annahmen zum Mengengerüst machen:

- Die Gesamtstrecke ist kleiner als 2^{60}
- Die Zahl der Stopps ist kleiner als 2^{20}
- Die Zahl der Etappen ist kleiner als 2^{25}

Aufgabe 3.5 ..... 1 Punkt

Dateiname: `paramref.pdf`

Gegeben sei das folgende Programm in einer Ada-ähnlichen Sprache, die es im Gegensatz zu Ada ermöglicht, einen Parameterübergabemechanismus durch einen Compiler-Schalter auszuwählen, so dass dann alle Parameter mit diesem Mechanismus übergeben werden. Auf alle formalen Parameter seien auch schreibende Zugriffe zulässig.

```
 1  procedure Params is
 2     X : Integer := 2;
 3     Y : Integer := 5;
 4     Z : Integer := 1;
 5
 6     A : array (1 .. 5) of Integer := (2, 3, 1, 4, 5);
 7
 8     procedure G (X : in out Integer; Y : in out Integer) is begin
 9        A (Y) := A (A (Y));
10        Z := X;
11        Y := Y + X;
12     end G;
```

```
13
14      procedure F (X : in out Integer; Y : in out Integer) is
15         Z : Integer := 2;
16      begin
17         X := A (Z);
18         Y := A (X);
19         G (X, Z);
20         A (X) := A (Z);
21      end F;
22
23   begin
24      F (Z, A (Y));
25
26      Put (X); Put (Y); Put (Z); New_Line;
27      Put (A (1)); Put (A (2)); Put (A (3)); Put (A (4)); Put (A (5));
28   end Params;
```

Geben Sie für den *By-Reference-Übergabemechanismus* an, welche Ausgaben das Programm produziert, und erklären Sie, wie diese Ausgaben zustande kommen. Sollte das Programm auf das Array mit falschem Index zugreifen, geben Sie den Zustand der globalen Variablen zu diesem Zeitpunkt an. Gehen Sie dabei davon aus, dass jegliche Optimierungen des Compilers deaktiviert sind.

Aufgabe 3.6 ..... 4 Punkte

Dateiname: `smartpointers.pdf` ★★★

Im Rahmen dieser Aufgabe soll die Heap-Speicherverwaltung mittels des Reference-Counting-Mechanismus betrachtet werden. Auf der Veranstaltungs-Homepage finden Sie dazu einige vorbereitete Ada-Quelltextdateien.

Die Implementierung des Reference-Counting-Mechanismus, enthält vier Fehler in der Datei `smart_pointers.adb`. In einer fehlerfreien Version wurden kleine Bereiche verändert um Fehler einzubauen.

Die Datei `Test_Five` enthält einen Test für die Implementierung. Das zeigt, wie sie die Implementierung testen können.

Sie sollen nun die vier Fehler finden und dokumentieren. Die Dokumentation jedes Fehlers besteht aus drei Teilen:

1. Der von Ihnen vorgeschlagenen sehr kleinen Änderung am Code.
2. Einem Testfall, ähnlich wie `Test_Five`, der vor und nach Ihrer Änderung unterschiedliche Ausgaben (und damit Ergebnisse) liefert.
3. Einer Begründung, warum das ursprüngliche Ergebnis falsch und das neue richtig ist.

Nachdem die vier Codeanpassungen gemacht wurden, muss die Implementierung richtig funktionieren. Stellen Sie dies durch eigene Tests sicher.

Wichtiger Hinweis: Achten Sie bei dieser Aufgabe auf übersichtliche, ansprechende und kompakte Darstellung. Ihre Testfälle sind kurz und bündig. Wenn Ihre Ausführungen nicht leicht nachvollziehbar sind, werden Ihnen Punkte abgezogen.

Die Testfälle sollten so vollständig im pdf abgebildet sein, dass man sie leicht per Copy-and-Paste in eine Textdatei übertragen und kompilieren kann. Es darf aber nicht so sein, dass ihre Ausführungen nur zu verstehen sind, wenn man die Programme tatsächlich ausführt. Tatsächlich muss Ihre Beschreibung hinreichend sein.