01612 Konzepte imperativer Programmierung

Datum: 15.02.2006
Dauer: ca. 35 min
Prüfer: Prof. Six
Note: 2,3

Zuerst bekam ich wie üblich die bekannte Typdefinition und die Aufgabe, "alle Elemente um eine Stelle nach vorn zu rücken". Er erläuterte dann noch, daß hierfür also das erste Element an das Listenende angehängt werden muß. Dann fragte er, was mir dazu als erstes einfallen würde und wollte hier die Sonderfälle genannt wissen (leere Liste, nur ein Element). Diese Fälle sollte ich auch in der Procedure berücksichtigen. Das Programmieren ging dann recht schnell.
Anschließend schockte er mich mit einer Frage zur Rekursion: wann denn eine rekursive Lösung angewendet werden muß und wann nicht. Antwort: wenn für die iterative Lösung eine Hilfsstruktur aufgebaut werden muß. Auf die Frage, was für eine Hilfsstruktur das wäre, antwortete ich (falsch) Array. Die richtige Antwort ist ein Stapel. Er zeichntete dann einen binären Suchbaum, in dem ein Element gesucht und, falls vorhanden, ein Zeiger darauf zurückgegeben werden soll und wollte wissen, ob die Lösung iterativ oder rekursiv erfolgen muß. Antwort: iterativ.

Dann ging er zur Theorie über. Was ist Verifikation und was braucht man dafür. Dann Angeben der Programmformel {P}S{Q}. Er fragte, wofür die Vorbedingung gelten würde und wollte auf die Eingabewerte hinaus, welche Logik (Prädikatenlogik) und warum (Aussagelogik nicht mächtig genug). Abschließend noch einige Fragen zum Hoare-Kalkül.

Auch wenn an sich eine lockere Atmosphäre geherrscht hat, so war es bei mir nach der unerwarteten Frage zur Rekursion mit der Ruhe völlig vorbei. Ich fand es teilweise sehr schwer, zu erkennen worauf er hinaus wollte. Die Benotung ist meiner Meinung nach aber sehr wohlwollend.

Viel Spaß

Prüfungsprotokoll

Vordiplomprüfungsleistung Praktische Informatik
Kurs 1612 „Konzepte imperativer Programmierung“

Datum: 23.11.2005
Dauer: ca. 30 Minuten
Prüfer: Prof. Six

Prof. Six stieg gleich ohne viele Worte in die Prüfung ein. Er legte mir einen DIN-A4-Zettel vor, auf dem ausgedruckt die Definition einer Liste lag (wie im Kurs):

```
type
  tRefElement = ^tElement;
  tElement    = record
                  info : integer;
                  next : tRefElement
                end;
```

Dann erklärte er noch einmal mit eigenen Worten diese Typdefinition. Als nächstes malte er eine lineare Liste wie folgt auf:

Anfang → ☐ → ☐ → ☐ → ☐ → ☐ → |

Eines der Elemente (er markierte es mit einem „x“) enthalte einen bestimmten Integer-Wert in der `info`-Komponente. Dieses Element soll gesucht und durch Änderung der Verkettung ans Ende gehängt werden. Zunächst wollte er wissen, welche Sonderfälle auftreten können. Das hat er allerdings so formuliert, dass ich nicht sofort wusste, was er meinte. Im Laufe des Gesprächs klärte sich das aber auf. Da es gleich mehrere Sonderfälle gab, sagte er, dass ich der Einfachheit halber in der nun folgenden Ausprogrammierung der entsprechenden Prozedur davon ausgehen könne, dass das Element in der Liste vorhanden sei, weder direkt am Anfang noch ganz am Ende stehe und dass die Liste mindestens drei Elemente enthalte.

Beim Programmieren gab es meinerseits natürlich situationsbedingt einige Unsicherheiten, die Prof. Six und sein Beisitzer aber ziemlich schnell kommentierten (z.B. wollte ich erst nur einen Hilfszeiger nehmen, aber Prof. Six sagte dann „Nennen Sie den mal `hilf1` statt `hilf`.“), so dass ich recht schnell eine lauffähige Lösung hatte. Ich glaube, dass es mehr darauf ankommt, dass der Prüfer sieht, dass man das Prinzip verstanden hat. Denn dass man nicht sofort ein lauffähiges Programm zu Papier bringt, zumal in einer Prüfungssituation, ist ja nicht ungewöhnlich.

Nach Beendigung des Programmierteils ging Prof. Six zu den Themen Syntax und Semantik über und stellte einige der Standardfragen, die sich auch in den anderen Protokollen finden („Was ist eine Syntax/Semantik?“, „Wie kann man eine Syntax/Semantik darstellen?“ usw.). Auf die Frage, wie man die Syntax darstellen kann, antwortete ich zunächst „mit einer Grammatik oder einem Flussdiagramm“. Als ich sein Stirnrunzeln sah, verbesserte ich das „Flussdiagramm“ zu einem „Syntaxdiagramm“, was er mit einem freundlichen „so gefällt mir das schon besser“ kommentierte. Da ich auf diese Themen aber eigentlich gut vorbereitet war – was er wohl auch merkte –, bohrte er an vielen Stellen nach, wohl um zu sehen, wie tief er gehen konnte. Das hatte ich so nicht erwartet. Er ließ sich die BNF-Darstellung der

kontextfreien Grammatik im Detail von mir erklären und fragte dann, ob man bei der Produktionsregel beliebige Aneinanderreihungen von verschiedenen Terminalen und Nichtterminalen zulassen könne. Die Frage überraschte mich ein wenig, da ich mich das noch nie selbst gefragt hatte. Nach kurzer Überlegung antwortete ich, dass man es wohl dahingehend einschränken müsse, dass man nicht einfach ein Nichtterminal durch dasselbe Nichtterminal ersetzen dürfe, weil man ja sonst eine Endlosschleife erhält. Ich weiß nicht, ob es noch mehr Einschränkungen gibt, jedenfalls sind mir auf die Schnelle keine weiteren eingefallen, und so ging Prof. Six zum nächsten Thema „Verifikation“.

Hier stellte er auch wieder einige der bekannten Fragen. Die while-Regel musste ich aufschreiben und erklären. Zum Ende fragte er mich („jetzt kommt eine schwere Frage“), warum Hoare eigentlich dieses Schema gewählt hat. O-Ton etwa: „Wenn ich bei einer while-Schleife doch das Schema „{P} while B do S {P ∧ –B}“ unter dem Strich schon habe, warum brauche ich dann noch das, was über dem Strich oben steht? Da steht doch praktisch dasselbe!“ Da war ich ziemlich verdutzt und sagte erst, dass es doch damit zusammenhängt, dass die Begriffe „while“ und „do“ PASCAL-spezifisch sind und man deshalb erst den Rumpf betrachten müsse, um dann noch die PASCAL-Wörter „drumherum“ setzen zu können (wie bei der Zusammensetzungsregel). Das war aber wohl nicht das, was er hören wollte. Er erklärte es mir dann nach ein paar weiteren erfolglosen Rateversuchen. Leider kann ich die genaue Erklärung nicht mehr wiedergeben, so dass ihr euch da schon selbst Gedanken machen müsst ;) Es hatte was mit der Kontrollstruktur und einer Analogie zur Bedingungsregel zu tun. Die Frage ging aber wohl nicht in die Bewertung ein, da er ja schon vorher bemerkte, dass die Prüfungszeit abgelaufen war.

Fazit: Prof. Six ist ein sehr angenehmer Prüfer. Nach der eigentlichen Prüfung gab es noch ein bisschen Smalltalk über den Studienverlauf und den persönlichen Hintergrund. Letztendlich liegt es wohl an der eigenen Person, wie viel Stress man in einer Prüfungssituation empfindet, aber Prof. Six war während der ganzen Prüfung sehr entspannt. Ich hätte mir lediglich ein bisschen mehr Bestätigung während meiner Erläuterungen gewünscht, denn er hörte sich meist einfach alles an, was ich sagte, ohne es mit einem irgendwie gearteten „richtig“ oder „falsch“ zu kommentieren. Wenn er dann Dinge hinterfragt, kommt man manchmal selbst ins Grübeln, ob er jetzt nachfragt, weil es falsch war, oder weil er einfach nur sehen möchte, wie sicher man den Stoff beherrscht. Überrascht war ich davon, dass er nicht einen groben Abriss des Kurses abgefragt hat, sondern sich auf wenige Themen (Zeiger, Syntax/Semantik, Verifikation) beschränkt, diese aber sehr detailliert behandelt. Die 30 Minuten vergingen trotzdem wie im Fluge. Hab eine ziemlich gute Note bekommen und drücke allen Prüflingen die Daumen – wer den Kurstext verstanden hat und ordentlich mit Zeigern umgehen kann, sollte hier eigentlich keine Probleme haben.

Fachprüfung: Praktische Informatik
Prüfungsinhalt: Konzepte Imperativer Programmierung 1612
Prüfer: Prof. Dr. H.-W. Six
Datum: 01.02.05
Dauer: 30 Minuten für die eigentliche Prüfung
Note: 1,7

1. Teil:
Nach einer kurzen Vorstellung zeigte Prof. Six eine Typdefinition für eine lineare Liste (die gleiche wie in den anderen Prüfungsprotokollen). Dann zeichnete er eine solche Liste auf und erläuterte die Aufgabenstellung: Durch Verkettung der Liste soll das erste Element an das Ende der Liste gehängt werden. In der Skizze zeichnete ich die Veränderungen der Zeiger ein.

Er wollte danach von mir hören, was für Sonderfälle auftreten können (leere Liste und Liste mit einem Element), was er aber etwas indirekt fragte.

Danach ging es an die eigentliche Programmierung. Den Prozedurnamen gab er vor, der Rest war meine Sache. Dabei erläuterte ich jeden Schritt, den ich durchführte.

2. Teil:
Nach dem guten Start mit der Programmierung schwächelte ich etwas im theoretischen Teil. Er wollte von mir die Definition der Syntax einer Programmiersprache wissen, das sind alle (Schlüssel-)Worte, die vorkommen und in welcher Reihenfolge sie vorkommen, allerdings auch Kontext-Sachen, wie zum Beispiel ob eine benutzte Variable auch deklariert ist. Das habe ich jedoch nicht gleich erwähnt, weil die kontextfreien Grammatiken kontextfrei sind (diese aber nur eine Definitionsform der Syntax sind). Die Erwähnung der kontextfreien Grammatik war vielleicht nicht so schlau, denn dann musste ich auch erklären, was es damit auf sich hat. (Grammatik ist ein Tupel aus Terminalen, Nichtterminalen, Ersetzungsregeln und dem Start-Nichtterminal). Er wollte auch noch erwähnt haben, dass Syntaxdiagramm und BNF gleich mächtig sind.

Schließlich habe ich noch die Semantik einer Programmiersprache, bzw. der vorkommenden Anweisungen anhand einer Programmformel erklären sollen. Dabei hätte er gerne hören wollen, dass bei {P} S {Q} das Prädikat P mittels des Programmes S in das Prädikat Q transformiert wird. Der Unterschied zwischen Aussagen- und Prädikatenlogik ist auch wichtig, welcher mir leider nicht so bewusst war.

Prof. Six ist ein freundlicher Prüfer, der einem auch etwas die Nervosität nehmen kann. Wenn man an einer Stelle nicht weiterkommt, versucht er einen mit kleinen Hilfestellungen auf den richtigen Weg zu bringen, dadurch gibt es aber leider auch wenige Themenwechsel, wenn man bei einem Kapitel ist, das einem vielleicht nicht so liegt. Die Benotung ist durchaus als wohlwollend anzusehen.

Fachprüfung: Praktische Informatik
Prüfungsinhalt: Konzepte imperativer Programmierung 1612
Prüfer: Prof. Dr. H.-W. Six
Datum: 26.02.03
Dauer: ca. 40 min

Praktischer Teil:
Programmieren und erläutern einer Prozedur, die in einer linearen Liste ein Element sucht und dieses an den Listenanfang stellt.
Fragen nach Sonderfällen und Parameterübergabe.

Theoretischer Teil:
Gefragt wurde nach den Begriffen Semantik und Verifikation. Hier erwischte er mich auf dem linken Fuß, denn meine Schwachstelle war eindeutig die Verifikation. Meine Hoffnung, dass über dieses Thema (mit entsprechenden Minuspunkten) hinweggegangen wird, erfüllte sich leider nicht. Prof. Six verlangt, dass die Verifikation wirklich verstanden wird und stellt entsprechende Fragen.
Das Verstehen der Verifikation und dies auch entsprechend wieder geben zu können sollte man unbedingt leisten können.
Da die Prüfung hier , hängen geblieben' ist, wurden weitere Themen (Syntax, Grammatik, Testen,...) nicht abgefragt.

Allgemein:
Prof. Six ist ein sehr freundlicher und sympathischer Prüfer. Er lässt den Prüfling ausreichend Zeit zur Beantwortung von Fragen. Durch Hinterfragen der Antworten prüft er, ob der Prüfling den Stoff verstanden, oder diesen nur auswendig gelernt hat.
Seine Benotung war entsprechend dem Prüfungsverlauf gerecht (Teilnoten: Praktischer und theoretischer Teil jeweils ca. 50%)..
Insgesamt ist Prof. Six ein Prüfer, den ich absolut empfehlen kann.

Fachprüfung : Praktische Informatik
Prüfungsinhalt : Konzepte imperativer Programmierung 01612
Prüfer : Prof.Dr.H.-W.Six
Datum : 08.01.02
Dauer : ca.40 min
Note : 1.3

Der praktische Teil der Prüfung :
es wurde eine Prozedur gesucht, die das erste Element einer Liste mit dem zweiten vertauscht. Bei der Ausprogrammierung der Prozedur wurden sowohl die Parameterübergabeart (hier call-by-reference) als auch die Sonderfälle (leere Liste, die Liste enthält nur ein Element) diskutiert. Ich habe versucht Definitionen aller erwähnten Begriffe während der Diskussion ohne Nachfragen selbstständig zu erläutern, was, nach meiner Meinung, positiv auf die gesamte Benotung wirkte.

Der theoretische Teil der Prüfung :
?? Definition "korrektes Programm"
?? Programmformel {P}S{Q} erläutern
?? Definition "Spezifikation eines Programms"
?? partielle, totale Korrektheit
?? Definitionen und Gegenüberstellung für Verifikation und Testen

Prof.Dr.Six ist ein sehr freundlicher und angenehmer Prüfer. Er sorgt für eine entspannte Prüfungsatmosphäre in dem er nach einer korrekt formulierten Frage dem Prüfling genug Zeit für die Antwortvorbereitung läßt. Bei der Vorbereitung zur Prüfung sollte man nicht nur das Kapitel "Lineare Liste", sondern auch Suchbäume und arrays gründlich wiederholen und verstanden haben.

Viel Erfolg!

Aleksej Petri

Gedächtnisprotokoll Vordiplomprüfung in Praktischer Informatik.

25.9.2000 Prüfer:
Prof. Six
Thema: Kurs 1612 - Konzepte imperativer Programmierung.

```
type
tRefElement = ^tElement
        tElement = record
                info: integer,
                next: tRefElement
        end;
```

procedure PosEntfemen (var inPos: tRefElement; var ioAnfang: tRefElement);
(entfernt aus einer Liste das Element, auf das InPos zeigt)

```
var
zeiger: tRefElement;

begin
      if inPos = ioAnfang then
            ioAnfang: = ioAnfang. next
      else
            begin
                zeiger: = ioAnfang
                while zeiger^.next <> nPos do
                 zeiger: = zeige^. next;
                zeiger^next: = inPos.^next,
                dispose (inPos)
            end;
end
```

Was versteht man unter partieller und totaler Korrektheit, gültiger Programm-formel, formaler Semantik und Spezifikation eines Programms.
Welche Testmethoden gibt es?
Was sind black-box und white-box- Verfahren?
Wie arbeiten kontrollflußbezogene Testverfahren?
Woraus besteht der Graph?
Welche Überdeckungsmöglichkeiten gibt es?
Entspricht eine Knotenüberdeckung in jedem Fall einer Zweigüberdeckung?
(Nein, z.B. leerer else-Zweig)
Welche Testdaten werden benutzt, was sind Äquivalenzklassen, spezielle Wer-te, Grenzwerte?

Gedächtnisprotokoll vom 24.03.2000; 11.00 bis 11.35 Uhr

Vordiplom-Prüfung „Praktische Informatik“

Prüfer: Prof. Dr. H.-W. Six
Mitarbeiter: A. Homrighausen

Geprüfter Kurs: 1612 – Konzepte imperativer Programmierung (10/96)

Zuerst erfolgte eine lockere Begrüßung, die etwas kürzer ausfiel, da ich ca. 4 Wochen zuvor eine Prüfung als Beobachter erleben konnte. Prof. Six erkannte mich wieder und überprüfte zuvor, dass nicht die gleiche Programmieraufgabe von ihm gestellt wurde.

Danach wurde mir die Typdeklaration der Programmieraufgabe vorgelegt und die Aufgabe unter Anfertigung einer Skizze erklärt:

```
type
tRefElement    =    ^tElement;
tElement       =    record
                      info : integer;
                      next : tRefElement
                    end;
```

„ Wir haben eine lineare Liste, aus der das letzte Element an den Anfang der Liste gesetzt werden soll. “

Die Lösung kann wie folgt aussehen:

```
procedure umhaengen ( var ioAnf : tRefElement );

  var
  hilfl,
  hilf2 : tRefElement;

begin
  hilf1 := ioAnf;
  while (hilf^.next^.next <> nil) do
    hilf1 := hilf1^.next;
  hilf2 := hilf1^.next;
  hilf1^.next := nil;
  hilf2^.next := ioAnf;
  ioAnf := hilf2
end; { umhaengen }
```

Prof. Six legte Wert darauf, dass die Prozedur nicht nur aufgeschrieben, sondern auch erklärt wurde. Dazu wollte er auch die graphische Änderung in der linearen Liste von mir aufgezeichnet bekommen, um sich ein Bild darüber zu machen, ob der Algorithmus korrekt sein wird. Positiv fiel auf, dass ich die Sonderfalle (Liste ist leer oder hat nur ein Element) gleich angesprochen und berücksichtigt habe. Er konkretisierte danach die Programmierung durch Zwischenfragen und half bei Problemen mit kurzen Hinweisen weiter. (Dies ist wohl dadurch zu verstehen, dass die Aufgabe insgesamt nicht länger als ca. 15 Minuten in Anspruch nehmen soll,)

Als weitere Themen wurden im Anschluß die in den bisherigen Prüfungsprotokollen häufig genannten Gebiete geprüft:

- *Was sind Syntax und Semantik.?*
  Die Syntax wird durch das Vokabular und die erlaubten Wörter und Begriffe, sowie den Aufbau der Programme definiert, die Semantik ist die Bedeutung bzw. Wirkung der Anweisungen bzw. des Programms.

- *Wie erkläre ich die Syntax von PASCAL?*
  Durch ein Syntax-Diagramm oder/und eine Grammatik.

- *Was ist eine Grammatik.?*
  Eine Konstruktionsvorschrift zum Erzeugen einer Sprache.

- *Kann ein Syntax-Diagramm „mehr“ als eine Grammatik?*
  Nein.

- *Wie kann ich die Semantik einer Anweisung erklären?*
  Hier wurde der einfache Satz „Wenn vor der Ausführung der Anweisung S die Vorbedingung (P} gilt, gilt nach der Ausführung die Nachbedingung(Q)” verlangt.

Fazit: Herr Prof. Six ist als Prüfer ohne weiteres zu empfehlen, da er sich Mühe gibt, einem Prüfling die Unsicherheit in einer Prüfung so weit wie möglich zu nehmen. Außerdem sucht er nach dem Wissen des Prüflings und nicht nach seinen Lücken. Ich habe das Gefühl gehabt, dass er meine vorherige Teilnahme als Besucher an einer Prüfung als sinnvoll erachtete. In der Nachbesprechung brachte er seine ehrliche Meinung, sowie einige gute Hinweise, die mir im weiteren Studium weiterhelfen werden. Ich bin ihm dafür auch dankbar, denn ich habe beruflich mit Programmierung nichts zu tun und kann daher keine Routine aufweisen. Seine Benotung war wohlwollend.

Viel Glück, wenn Du dran bist!

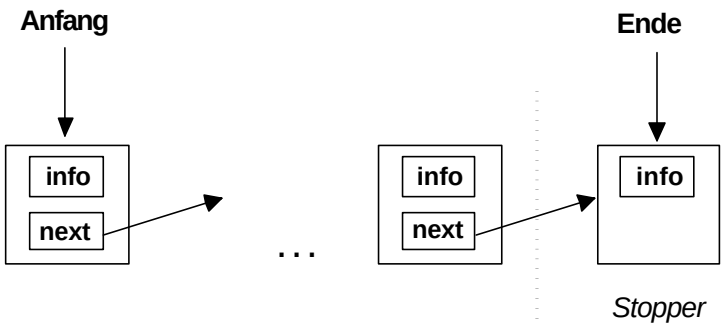
Prüfungsinhalt:

- **Programmieraufgabe:**

Gegeben sei die Typdefinition

```
type
tRefListe = ^tElement;
tElement = record
             info : integer;
                   next : tRefListe
                 end;
```

Die betrachtete Liste habe zusätzlich am Ende einen sogenannten "Stopper", d.h. ein Element dessen info-Komponente selber nicht zu den verwalteten Listenelementen gerechnet wird, der keine next-Komponente besitzt (also kein **nil** -Zeiger für das Listenende notwendig!) und auf den ein Ende-Zeiger gesetzt ist.



Durch diese Anordnung wird nun die Suchoperation zusätzlich vereinfacht. Dazu wird jeweils das gesuchte Element vor dem Suchdurchlauf in die info-Komponente des Stoppers geschrieben.
Schreiben Sie eine Prozedur oder eine Funktion welche ein integer-Element in dieser Liste sucht.

Meine Lösung sah folgendermassen aus (falls ich mich recht erinnere):

```
procedure SucheElem (
                      inAnfang,
                      inEnde : tRefListe;
                  var outRes : tRefListe;
                inZahl : integer);
{outRes gibt den Zeiger auf das erste Element mit der info-Komponente inZahl
zurueck. Tritt inZahl nicht auf, wird outRes auf den Stopper gesetzt}

        var
        Hilf : tRefListe;

begin
        inEnde^.info := inZahl;
        Hilf := inAnfang;
  while Hilf^.info <> inZahl do
          Hilf := Hilf^.next;
  outRes := Hilf
end; {SucheElem}
```

- **Diskussion der Programmieraufgabe:**

  ? Genaue Funktion des Stoppers
  ? Wo Vereinfachung ? (keine Abbruch-Bedingung für die next-Komponente; "suche nach nil nicht erforderlich")
  ? Wieso das **var** vor dem outRes-Zeiger in der Prozedur- Schnittstelle ? (kurz was über Parameter-Übergabearten gesagt; als ich ausholen wollte um Para- mit Standard-Pascal zu vergleichen, blockte Prof. Six ab)

- **Weitere Themen/Fragen waren:**
  ? Was ist Verifikation ? (Überblick gegeben, keine Details. Stichworte: Programmformel, Semantikdefinition & Hoare-Regeln [wollte aber keine Hoare-Regel wissen!])
  ? Was ist Testen ? (ebenfalls Überblick über analytische Qualitätssicherung gegeben. Stichworte: dynamische Testverfahren; strukturorientierte und funktionale Test; Kontrollflussgraphen [erwähnt]; Unterschied: Test gegen Spezifikation und Test eines Programms gegen sich selbst)
  ? Der letzte Punkt führte dann auf die letzte Frage:
  Stellen Sie sich vor, Sie arbeiten für ein Softwarehaus und haben soeben eine grössere Applikation (mit)entwickelt. Wegen Termindruck müssten Sie sich nun entscheiden, entweder einen funktionalen oder einen strukturorientierten Test zu machen. Beides geht zeitlich nicht mehr. Wofür würden Sie sich entscheiden ?
  (Da ich über diese Frage bzw. die Art der Frage etwas überrascht war, wollte ich mich nicht festlegen und sagte ich würde unter allen Umständen versuchen beides - d.h. Tests gegen den Code wie auch gegen die Spezifikation - durchzuführen, wenn nötig halt beide Verfahren im Minimum. Der Kunde wolle zwar eine Lösung, die sein Problem löst; jedoch wolle ich ihm auch nicht eine Applikation bieten, die des öftern abstürzt.
  Damit war Prof. Six nicht einverstanden und meinte, das Wichtigste sei, dass das Programm die Spezifikation erfülle. In diesem Fall akzeptiere der Kunde auch mal einen Absturz - welchen man ja immer noch beheben kann.)

Prüfungsatmosphäre

Prof. Six ist meiner Meinung nach vorbehaltlos als Prüfer zu empfehlen. Seine Art ist sehr ruhig und freundlich. Die Notengebung ist fair. Er gab mir die Möglichkeit relativ frei zu sprechen und an Stellen, wo er mehr wissen wollte oder mich nicht ins Detail gehen lassen wollte, tat er das so, dass ich dadurch nicht "aus dem Konzept" kam.

Bei der letzten Frage (die "Softwarehaus-Frage") hat er mir versichert, dass es ihm weniger um eine "korrekte" Antwort geht (das stehe ohnehin nicht im Kurs), als darum zu schauen, wie ich argumentiere. Hier hätte ich selber mehr berücksichtigen sollen, dass er die Kurse über Software Engineering anbietet. Seine Argumentation ist aber zweifellos auch sonst richtig und ich glaube er wollte mir vor allem aufzeigen, dass Programmieren "im Kleinen" wie in 1612 nur ein (kleiner) Aspekt ist, und dass eben Software-Design und -Modellierung heute durch Objektorientierung, OMT, UML etc. eine grössere Bedeutung haben. Dies ist bei der Mächtigkeit heutiger Applikationen wohl unbestreitbar so.
Also: Bei dieser Frage unbedingt über den Kurs 1612 hinausdenken...

Viel Erfolg!

Prüfungsprotokoll

Vordiplomsprüfung Praktische Informatik am 07.10.99

Prüfer: Prof. H.-W. Six

Kurs 1612 WS 98/99

Allgemein:

Die Prüfung teilt sich in einen Programmier-Teil sowie einen Teil mit Fragen zum Stoff.

Das Prüfungsklima ist ausserordentlich angenehm, der Prüfer sehr freundlich und bereit, dem Prüfling auf die Sprünge zu helfen. Die Fragen im Theorie-Teil haben deutlichen Bezug zu Praxis – es ist wichtig, sich seine eigenen Gedanken zu einem Thema zu machen.

Urteil: sehr empfehlenswert.

Programmier-Teil:

Es wird sicherer Umgang mit Zeigern erwartet.

Meine Aufgabe lautete festzustellen, ob in einer Liste ein bestimmtes Element (integer-Zahl enthalten) ist und dieses Element dann aus der Liste zu entfernen. Der Trick an der Liste war, dass dem letzten Listenelement das gesuchte Element beim Aufruf der Funktion zugewiesen wird. Es wird also beim Dursuchen der Liste auf jeden Fall beim gesuchten Element angehalten. Damit wird eine Sonderbehandlung des Listenendes umgangen, sollte die Liste das gesuchte Element nicht enthalten. Das Element wird nur entfernt, wenn der Suchzeiger nicht auf dem letzten Element, sondern auf einem anderen Element, stehen bleibt.

Vorgaben:

Die Liste sollte mindestens ein Element enthalten.

Es wird ein Zeiger auf den Anfang der Liste (RefAnfang) sowie ein Zeiger auf das letzte Element der Liste übergeben (RefEnde). Ausserdem natürlich der gesuchte integer-Wert.

Es werden zuerst die Sonderfälle behandelt:
Die Liste enthält nur ein Element (das Schlusselement).

Idee:

Zuweisung des integer-Wertes an RefEnde.

Behandlung des Sonderfalles „Liste hat nur ein Element“: RefAnfang^.next = nil?

Wenn ja -> fertig

Wenn nein:

Setzen des Such-Zeigers (RefSuch) und Durchlaufen der Liste bis RefSuch^.next^.info = Wert

Ist das gefundene Element das letzte der Liste? (RefSuch^.next^.next = nil)

Wenn ja: Liste enthält das Element nicht -> fertig

Wenn nein: Liste enthält das Element -> Element wird entfernt

Entfernen: RefSuch^.next := RefSuch^.next^.next

Es wird Wert auf sauberen Programmierstil gelegt. Eine gute Idee ist es auch, beim Niederschreiben gleich ein bisschen zu kommentieren, was man da tut, der Prüfer kann einschreiten, sollte man in der völlig falschen Richtung unterwegs sein.

<u>Theoretischer Teil:</u>

- Definition und Unterschied zwischen Verifikation und Testen
- Verifikation: Definition von Semantik, wie haben wir im Kurs verifiziert?
- Testen: Unterschied zwischen Black-Box und White-Box-Verfahren
- Sie verkaufen ein Programm an einen Kunden und sind als Programmierer mit der Endabnahme befasst – verifizieren oder testen sie und warum?

Konzepte inperativer Programmierung - Kurs 1612

Gedächtnisprotokoll einer mündlichen Prüfung

Prüfer: Prof Dr Six **Tag: 06.11.98**
Dauer: ca. 35 Minuten **Note: 1.7**

Zuerst wurde die wohl in jeder Prüfung übliche Programmieraufgabe mit einer Liste gestellt: Auf einem Zettel lag die bekannte Typenvereinbarung einer Liste vor (Kurstext, S. 167) und die Aufgabe bestand darin, das letzte und das erste Element der Liste zu vertauschen. Zunächst sollten Veränderungen an einer von Prüfer skizzierten Liste aus vier Elementen dargestellt werden.

Der übergebene Anker bzw. Zeiger auf den Anfang der Liste muß als Referenzparameter übergeben werden: *...var ioAnf : tRefElement...*, da er verändert wird. Bei der Programmierung ist auf Sonderfälle zu achten: Leere Liste und ein Element in der Liste. Diese Sonderfälle sind bei der Programmierung der Vertauschung auszuschließen. Wichtig war dabei der Hinweis, daß die folgende Programmzeile i. a. zu einem Laufzeitfehler führt:
if (ioAnf <> nil) and (ioAnf^ .next <> nil) then
Wenn *ioAnf = nil* ist, so wird außer bei Turbo-Pascal trotzdem *ioAnf^ .next <> nil* abgefragt, was zu einem Laufzeitfehler führt, da dann *ioAnf^ .next* nicht existiert. Auf Abhilfe, z. B. durch zwei geschachtelte If-Abfragen brauchte in der Prüfung nicht weiter eingegangen werden.

Die Lösung der Vertauschung erfolgte durch Durchlaufen der Liste bis zum vorletzten Element mit einer prozedurlokalen Zeigervariablen *help*. Abbruchbedingung für die while-Schleife: *help^.next^.next* = nil, also die while-Schleife wird durchlaufen wenn das gerade nicht erfüllt ist. Danach sind mit dem Zeiger *help* die Vertauschungen vorzunehmen.

Anschließend war der Begriff *call-by-reference* zu erläutern, dabei wird die Speicheradresse, nicht der Speicherwert, übergeben. Der Prüfer wollte auch die Begriffe *aktueller und formaler Parameter* hören. Diese Fragen nach Begriffen sind meistens ziemlich verschlüsselt gestellt, damit der Begriff in der Frage vermieden werden kann.

Im restlichen Teil der Prüfung ging es ausschließlich um Programmverifikation. Totale Korrektheit ist partielle Korrektheit und Terminierung. Diese Begriffe wurde ausführlich behandelt, dabei auch die Begriffe Programmformel und Programmspezifikation, {P}S{Q} und {P'}S'{Q'} mit P'=>P und Q=>Q', mit grafischer Darstellung von P'=>P entsprechend Kurstext S. 291-292. S bzw. S' steht für ein ganzes Programm oder auch nur für eine einzige Programmzeile, wobei zur Verfeinerung die Hoare-Regeln benutzt werden.

Auf den Terminierungsnachweis wurde ebenfalls ausführlich eingegangen. Wie wird die Terminierungsfunktion gebildet (Detail: τ: Zr -> Z, Kurstext S. 337)? Sie muß streng monoton fallend sein. Sie muß beschränkt sein (>=t*), vor jedem Schleifendurchlauf. Der Umkehrschluß gilt nicht, beim Terminieren der Schleife muß nicht zwangsläufig τ<t* werden. Dies wurde noch relativ eingehend erörtert.

Allgemein ist festzustellen,

- daß der Prüfer auf genaue Formulierungen Wert legt,
- daß man von sich aus wichtige Sachen ansprechen sollte wie z. B. Sonderfälle,
- daß bei dieser Prüfung nur wenige Kursinhalte geprüft wurden, dafür diese aber sehr tiefgehend. Über Module, Rekursion, Testen, Grammatik, Aussagenlogik kamen keine Fragen.

Viel Glück!

Prüfungsprotokoll

Vordiplomprüfung Praktische Informatik

Nov.98

Prüfer : Prof .Six

Kurs 1612 (WS 97/98)

Dauer : 30 Minuten

1 Gegeben ist folgende Typdefinition:

```
tRefElement = ^tElement;
tElement = record
              info : integer;
              next : tRefElement
           end;
```

Schreiben Sie eine Prozedur,die das letzte Element einer linearen Liste an den Anfang hängt

2.Erklären Sie die Begriffe "call by value","call by reference".

3.Was bedeutet die Korrektheit von Programmen ?

4.Was bedeutet totale und partielle Korrektheit7

5.Wenn ein Programm keine Schleifen hat und partiell korrekt ist,ist es dann auch total korrekt ?

6.Wie zeigt man die Terminierung von Programmen ?

7.Kann eine Schleife auch abbrechen bevor die zugehörige Terminierungsfunktion die untere Schranke erreicht.Bitte ein Beispiel angeben.

Protokoll zur Vordiplomsprüfung 1612 - Konzepte imperativer Programmierung
Prüfer: Prof. Dr. Six
Termin: August 98
Dauer: ca. 20 min

Programmieraufgabe

Gegeben ist eine lineare Liste.

□➔□➔□➔□➔nil

Das letzte Element dieser Liste soll so umgehängt werden, daß es danach das erste Element ist.

Lösung:

Vor dem Skizzieren einer Lösung sollten folgende Fragen geklärt werden:
- Rekursive oder iterative Lösung?
- Parameter-Übergabe referentiell oder Wert-Übergabe
- Sonderfälle „Liste leer“ und „Liste enthält nur ein Element“ prüfen und erklären
  hier: Keine Sonderfälle, da die Liste laut Voraussetzung 2 Elemente enthält.

```
procedure umhaengen (var ioAnfang: tRefElement);
        var
        Zeiger: tRefElement;
begin
        while Zeiger^.next^.next <> nil do
                Zeiger:= Zeiger^.next;
        Zeiger^.next^.next:= ioAnfang;
        ioAnfang := Zeiger^.next;
        Zeiger^.next:= nil
end; {umhaengen}
```

Prozeduren und Moduln
- Unterschied zwischen Prozeduren und Moduln
- Begriff für ein Modul in Pascal: unit
- Bestandteile einer unit: interface und implementation
- Was gehört ins Interface, was in den Implementation-Teil?
- Beispiel für unit: Datenstruktur nicht relevant

Testen
- Wie haben Sie in Ihrer beruflichen Programmiertätigkeit Ihre Programme getestet?
  Eingabedaten gebildet und Ergebnis überprüft.
- Zu welcher Art von Test gehört das? Funktionaler Test
- Gehört der funktionale Test zu den Black-Box- oder White-Box-Verfahren?
- Beschreibung der Durchführung des funktionalen Tests
- Welche White-Box-Tests gibt es hauptsächlich? Strukturorientierte Tests / Kontrollflußbezogene Tests
- Beschreibung der Erstellung eines kompakten Kontrollflußgraphen
- C_0-Test und C_1-Test beschreiben
- Inwieweit geht der C_1-Tst über den C_0-Test hinaus? Kann es Pfade geben, die beim C_0-Test noch nicht überprüft worden sind?
  Beispiel von mir while-Schleife, sinnvoller wäre if-then gewesen.

Semantik
- Wie wird Semantik beschrieben? Mit der Programmformel
- Welche Anweisungen können mit der Programmformel beschrieben werden?
  Leere Anweisung und Zuweisung, die anderen mit dem Inferenzschema
- Allgemeines Inferenzschema aufschreiben
- Warum hat Hoare nicht das Zeichen für logische Implikation ($\Rightarrow$)verwandt?
  H_1...H_i und H sind keine Prädikate, sondern Programmformeln. Zwischen Programmformeln ist die logische Implikation nicht erlaubt.

Prüfung: Vordiplom Praktische Informatik
Inhalt: Konzepte imperativer Programmierung (01612), Version 09/96
Prüfer: Prof. Dr. Hans-Werner Six
Beisitzer: Henrik Behrens
Datum: 29.06.1998
Dauer: ca. 35 Minuten

Die Prüfung war zweigeteilt in einen praktischen und einen theoretischen Teil.

Im praktischen Teil war eine Prozedur gesucht, die aus einer linearen Liste (bestehend aus integer-Elementen) ein Element mit einem bestimmten integer-Wert löscht. Zunächst erfolgte eine gemeinsame Diskussion der Sonderfälle, dann die Ausprogrammierung der Prozedur. Anschließend wurde noch das Modulkonzept diskutiert.

Auch im theoretischen Teil gab es für mich keine Überraschungen - es waren die Standardthemen dran:

- Definition Syntax und Darstellungsmöglichkeiten (Syntaxdiagramm, Grammatik)
- Definition Semantik
- Wie läßt sich die Semantik darstellen? (Programmformel {P}S{Q} erläutert)
- Bei welchen Anweisungen genügt die Angabe der Programmformel? (Nullaxiom und Zuweisungsaxiom, bei den anderen muß ein Inferenzschema angegeben werden)
- Wie lautet die WHILE-Regel?
- Interpretation der WHILE-Regel (z.B. wann ist INV gültig? vor jedem Schleifendurchlauf und nach jedem Schleifendurchlauf, usw. wie im Kurstext diskutiert)

Insgesamt ist Herr Prof. Six als Prüfer uneingeschränkt zu empfehlen, auch die Benotung war überaus zufriedenstellend. Die gesamte Prüfung lief in Gesprächsform ab, deshalb ist es auch schwer, ein exaktes Protokoll festzuhalten.

Zur Vorbereitung habe ich nur den Kurstext selbst sowie alte Prüfungsprotokolle des AStA verwendet. Zum Kurs 1612 gibt es aber leider relativ wenig Protokolle (die meisten Studenten machen die Prüfung Prakt. Inf. offenbar über den Kurs 1663 Datenstrukturen).

Prüfungsprotokoll zur Vordiplomsprüfung „Praktische Informatik“

Prüfer: Prof. Six
Zeit: März 98
Dauer: 30 min
Kursversion: WS 93/94

Allgemeine Hinweise:
Prof. Six ist ein ruhiger, freundlicher Prüfer, dessen Prüfungsstil ich persönlich als sehr angenehm empfand. Er stellt meist eine einführende Frage zu einem Thema und läßt einem dann Zeit, sein Wissen zu diesem Thema zu präsentieren, bevor er konkret nachhakt. Er legt großen Wert auf systematisches Vorgehen bei der Programmierung und ist angetan wenn man die axiomatische Methode der Verifikation möglichst formal beschreiben kann.
Auf meine Bemerkung, daß ich die mir gestellte Programmieraufgabe als relativ schwer im Vergleich zu früheren Prüfungen empfunden hätte, gab er mir recht und wies daraufhin, daß allgemein das Niveau der Programmieraufgaben von nun an höher liegen würde, insbesondere bei den Prüflingen, die keinen Schein über den Kurs 16 12 vorweisen könnten (war bei mir der Fall) und somit ihre Programmierkenntnisse noch beweisen müßten.
Weiterhin betonte er, daß es ihm wichtig sei, daß man nicht nur programmieren könne, sondern auch auch wisse und begründen könne, warum man eine bestimmte Lösung einer anderen vorzöge.

Prüfungsablauf:

1. **Programmierung:**
   Ich sollte eine Prozedur angeben, die die Werte einer linearen Liste verschiebt: Aus der Beispielliste
   A=>B=>C=>D
   soll somit
   D=>A=>B=>C resultieren.

   Ich wollte diese Verschiebung dadurch erreichen, dass ich durch Zeigerumhängen das letzet Element zum ersten machte, doch Prof. Six unterbrach mich und betonte, es sei eine „physikalische“ Verschiebung gefordert, d.h. tatsachliches Kopieren und Verschieben der Speicherzelleninhalte.
   Das fiel mir erst etwas schwer, mit etwas Hilfestellung von Prof. Six gelang es dann aber doch: Man durchwandert die Liste mit einem Zeiger und vollfuhrt dabei stets den gleichen Vorgang:
   1. Sichern des Wertes des gerade betrachteten Elementes (Aktüll:=Zeiger^.info)
   2. Kopieren des zuvor gesicherten Wertes in das gerade betrachtete Element (Zeiger^.info:=Zuvor)
   3. Aktüll=>Zuvor (Zuvor:=Aktüll)
   4. Zeiger nach rechts verschieben

   Nach Durchlaufen der Liste muss dann noch das erste Element gesondert behandelt werden.

   Wichtig ist bei der Programmieraufgabe, dass man systematisch vorgeht und folgende Punkte erklärt:
   1. Allgemeines Prinzip der Lösung (ohne Sonderfalle)
   2. Fallunterscheidungen (Sonderfälle, wie z.B. leere Liste, etc.)

3. Parameter: welche und von welchem Typ (in, io oder out)
4, Rekursive oder iterative Losung?

2. Syntax:
   - **Definition** Syntax
   - **Notationsmöglichkeiten** (kontextfreie Grammatik in BNF-Form, Syntaxdiagramme, Kellerautomaten)
   - Warum sind Programmiersprachen wie PASCAL nicht **100%ig** kontextfrei beschreibbar? (Wegen kontextsensitiver Randbedingungen, z.B. darf ein Bezeichner nicht zweimal im gleichen Block deklariert werden.)

3. **Semantik + Verifikation:**
   - **Definiton** Semantik
   - Unterschied operationale / denotationale /axiomatische Semantik (kommt in der aktuellen Kursversion nicht mehr vor!)
   - Def.: Axiomatische Semantik
   - Wie gibt man die axiomat. Semantikdef. eines Programms an? (Durch eine gültige Programmformel.)
   - **Def.**: Programmformel
   - Was ist die notwendige Voraussetzung **für** eine axiomat. Semantikdefinition eines Programms? (Eine axiomat. Semantikdef der Programmiersprache, in der das Programm vorliegt.)
   - Man hat die Spezifikation eines Programms und will die Semantik eines vorliegenden Programms auf **„Kompatibilität“** prüfen. Wie geht man konkret vor? (Man **„synthetisiert“** die Semantik des Programms aus der Semantik der einzelnen Anweisungen und muss abschliessend zeigen, dass gilt: Vorbedingung (Spez) impliziert Vorbedingung **(Prog)** und Nachbedingung **(Prog)** impliziert Nachbedingung **(Spez).)**

Gedächtnisprotokoll zur mündlichen Vordiplomsprüfung in Praktischer Informatik

Datum: 16.05.1997
Prüfer: Prof. Six
Thema: Kurs 1612 - Konzepte imperativer Programmierung
Note: 2,7

Inhalt der Prüfung:

Praktischer Teil:
Es waren zwei Anfangszeiger auf Listen gegeben. Es sollte Liste 2 zwischen dem 1. und 2. Element von Liste 1 eingefügt werden. Dabei wurde vorausgesetzt, daß Liste 2 nicht leer sein sollte.

Theoretischer Teil:
– Wofür sind Parameter bei Prozeduren gut?
(→ parametrisiertes Problem)
– Gibt es unterschiedliche Parameterarten?
(→ Werte-/Variablenparameter)
– Was ist der Unterschied zwischen einer Prozedur und einem Modul?
(→ Modul: Geheimnisprinzip, Sicherheit, Änderungsfreundlichkeit, Wiederverwendbarkeit)
– Wie ist ein Modul aufgebaut?
(→ Interface-/Implementationsteil)
– Was ist Syntax/Semantik?
(→ Syntax = äußere Form eines Programms; Semantik = Bedeutung der Schlüsselwörter)
– Wie ist Semantik definiert?
(→ Programmformeln bzw. Inferenzschemata)
– Wie ist ein Inferenzschema aufgebaut?
(→ Prämissen, Konklusion)

Fazit:
Prof. Six ist ein sehr angenehmer Prüfer. Er legt viel Wert auf eine präzise Beantwortung seiner Fragen und gibt viele Hilfestellungen, falls man Schwierigkeiten bei der Beantwortung der Fragen hat. Ansonsten schließe ich mich dem positiven Bild an, welches beim Lesen der früheren Gedächtnisprotokolle entsteht. Viel Glück bei der Prüfung!

Alexander Salim

Termin: 03/97
Pruefer: Prof. Dr. Six
Kurs: 1612 (Version WS 96/97)
Dauer: 40 Minuten (inkl. Vor- und Nachbesprechung)
Note: 1,3

1. Gegeben sei folgende Typdefinition:

```
type
tRefElement = ^tElement;
tElement = record
           info:integer;
           next:tRefElement
         end;
```

   Aufgabe:
   Schreiben Sie eine Prozedur, die zwei Listen so zusammenfuegt, dass die 2. Liste zwischen das 1. und 2. Element der 1. Liste eingefuegt wird.
2. In welchen Faellen zieht man eine rekursive Loesung einer iterativen vor?
3. Erklaeren Sie die Begriffe Syntax und Semantik einer Programmiersprache!
4. Wie kann man die Syntax einer Programmiersprache beschreiben?
5. Wie kann man die Semantik einer Programmiersprache beschreiben?

Kommentar:
Wie man bereits an den Aufgabenstellungen der Pruefungsprotokoile erkennt, laufen alle Pruefungen bei Prof. Six zu diesem Kurs in etwa gleich ab. Man muss also den gesamten Kurstext verstanden haben, kann sich aber mit Hilfe alter Protokolle sehr gut vorbereiten.
Da Prof. Six ausserdem ein sehr netter Pruefer ist, sollte diese Pruefung keine Huerde darstellen.

Gedächtnisprotokoll

Vordiplomprüfung Praktische Informatik

Termin: 26.09.95
Dauer: 55 Minuten
Prüfer: Prof. Dr. Six
Kurse: 1611, 1612

1. Schreiben Sie eine Prozedur, die aus einer Liste das vorletzte Element entfernt. Zur Vereinfachung setzen Sie voraus, daß die Liste wenigstens zwei Elemente enthält.
Zu dieser einleitenden Programmieraufgabe wurden vom Prof. Six nachfolgende Vorgaben gemacht:

```
Type  d e          = ^tRefElement;
      tRefElement = record
                        into: integer;
                        next: Node
                    end;
```

Die folgende Prozedur habe ich nahezu sofort herunterschreiben können, habe jedoch am Anfang vergessen, im Fall einer lediglich zweielementigen Liste (das vorletzte Element ist dann das erste Element!) auch das Versetzen des Anfangszeigers zu programmieren. Auf diesen Lapsus wurde ich von Prof. Six behutsam hingewiesen.

```
procedure Entferne-Element (var pAnfang: Node);
var   pHilf_1,pHilf_2: Node;
beain
      if   Anfang^.next^.next nil then {Liste nur zwei Elemente}
           beain
                pHilf_1:= Anfang;
                Anfang: = Anfang^.next;
                dispose (pHilf_1)
           end {if}
      else
           beain
                pHilf_1:= Anfang;
                pHilf_2: = Anfang^.next
                while pHilf_2^.next^.next <> nil da pHilf_2^.next
                     beoin
                          pHilf_1: = pHilf_1^.next;
                          pHilf_2: = pHilf_2^.next
                     end; {while}
                pHilf_1: = pHilf_1^.next^.next;
                dispose (pHilf.2)
           end; {else}
end; {procedure}
```

Mir fiel auf, daß großer Wert auf eine ordentliche Form des Programms gelegt wurde.

Während des Programmierens wurde ich gefragt nach der Art möglicher Parameterübergabe (call by value, call by reference) und warum ich mich bei der Prozedure für die Referenzübergabe entschieden hatte.

2. Der nächste Themenkomplex beschäftigte sich mit dem Unit-Prinzip. Wobei auch die Frage gestellt wurde, wie ich dies bei praktischer Programmierung textlich umsetze.. Weitere Frage: Vorteile des Unit-Prinzips.

3. Was verstehen Sie unter Programmverifikation und Testen?

4. Was ist eine Schleifen-Invariante?

5. Erläutern Sie, wann Sie eine iterative Lösung vorziehen und wann Sie eine rekursive Lösung wählen! Würden Sie z. B. bei vorliegender Prozedur eine rekursive Lösung vorziehen?

Beim Punkt 2 hatte ich Schwierigkeiten mit der Antwort, weil ich praktisch nie Units programmiert hatte und nur das aus den Kurseinheiten gelernte zitieren konnte. Das Gewicht dieses Fragenkomplexes für die Gesamtbewertung wurde dann auch etwas zurückgenommen, da meines Wissens in den Kurseinheiten und auch in den Einsendeaufgaben sowohl beim Kurs 1611/1612 aber auch bei 1575 das praktische Implementieren selbst geschriebener Units nicht verlangt wird.

Zum Thema Testen und Verifizieren wurde offensichtlich ein weitestgehend freier Vortrag zum Wesentlichen der beiden Themen verlangt. Obwohl ich mich auf Detailfragen zu diesen beiden Oberbegriffen, wie ich glaube, relativ gut vorbereitet hatte, war ich durch die Aufforderung zum freien Vortrag doch überrascht und wußte nicht so recht, was gehört werden wollte. So würfelte ich einige mir bekannte Fakten zusammen, was aber den Prüfer nicht so recht zufrieden stellte. Er half mir dann mit mehr ins Detail gehenden Fragen weiter. Auf den linken Fuß erwischt wurde ich in diesem Zusammenhang bei der Abfrage der Definition des Begriffes “Semantik”. Auf das Pauken von Begriffsbestimmungen hatte ich mich in der Vorbereitung der Prüfung gar nicht erst eingelassen. So war denn auch Prof. Six die Definition des Begriff nicht scharf genug, und das wurde auch nicht besser, als er mich nach der Definition des Begriffes “Syntax” fragte.

Beim Thema “Schleifen-Invariante” fühlte ich mich wieder wohler, weil diese Frage konkreter war und besser in meine Prüfungsvorbereitungen paßte.

Die Prüfung dauerte dafür, daß nur eine Kurseinheit geprüft wurde, überraschend lang. Nach alten Prüfungsprotokollen glaubte ich, nach 25 Minuten endgültig erlöst zu sein und war zunehmend irritiert, als die Prüfung - wie ich es empfand - gar kein Ende nehmen wollte.

Trotz alledem verlief die Prüfung insgesamt in einer sehr angenehmen, geradezu privaten Atmosphäre. Herr Prof. Dr. Six begann die Prüfung mit einer lockeren Konversion zu meiner Person. Insbesondere interessierte ihn, ob meine gegenwärtige Beschäftigung zu meinem Studium an der Fernuniversität einen Bezug hat, was bei mir nicht der Fall ist. Meine Hoffnung, daraufhin etwas milder geprüft zu werden, hat sich, wie aus dem Protokoll ersichtlich, ganz und gar nicht erfüllt. Mit der Benotung war ich dennoch zufrieden. Ich empfand sie nicht als wohlwollend, wie in anderen Protokollen geschildert, sondern eher als streng, aber gerecht. Ich kann Herrn Prof. Six als Prüfer uneingeschränkt weiterempfehlen.

Gedächtnisprotokoll Vordiplomsprüfung Praktische Informatik

Datum: **August 1995**
Prüfer: Prof. Dr. Six
Kurs: **16 12, Version WS 94/95**
Dauer: ca. 25 Minuten
Note: **1,0**

Prüfungsablauf:

Nach ein klein wenig Smalltalk zu Beginn (wie weit im Studium, welche Ziele werden mit dem Studium verfolgt) begann die Prüfung mit der Programmieraufgabe. Danach folgten die Fragen in etwa in der Reihenfolge wie auf der nächsten Seite aufgeführt. Zwischendurch kamen von Prof. Six noch ein paar weitere von der direkten Prüfung abweichende Fragen, z. B. “Arbeiten Sie in Ihrer Firma viel mit Zeigern oder eher weniger?” oder "Verwenden Sie Bibliotheken in Ihrer Firma?“.

Programmieraufgabe:

Es sind 2 Listen ListA und ListB mit der üblichen Typdefinition gegeben. ListB soll nach dem ersten Listenelement von ListA eingefügt werden. Dazu skizzierte Prof. Six beide Listen auf einem Blatt Papier und verdeutlichte durch Linien (= Zeiger), wie ListB in ListA eingefügt werden sollte.
Auf meine Nachfrage hin wurde noch festgelegt, daß ListA aus mindestens einem Element besteht, von ListB hingegen ist nichts bekannt.

```
procedure ListZweitEinfuegen (ListA, ListB : tRefElement);
   var
   hilfl, hilf2 : tRefElement;
begin
   { nur wenn ListB nicht leer ist, ist überhaupt etwas zu tun}
   if (ListB <> nil) then
   begin
      hilf1 := ListA^. next;          { next - Zeiger von ListA merken }
      ListA^.next := ListB;           { ListB hinter das 1. Element von ListA hängen }
      hilf2 := ListB;
      while (hilf2^.next <> nil) do     { ListB mit Hilfszeiger durchlaufen }
         hilf2 := hilf2^.next;
      hilf2^.next := hilfl;           { Ende von ListB zeigt auf das 2. Element von ListA }
   end
end;
```

Zu Beginn meines Entwurfs hatte ich nur eine Hilfsvariable vorgesehen und merkte dann, daß ich zwei Hilfsvariablen benötigte. Darauf versuchte Prof. Six, mich auf einen geschickteren Weg zu lotsen, wobei ich mit der einen Variablen ausgekommen wäre. Da ich seinen Hinweis aber nicht gleich verstand, meinte er nur “Machen Sie ruhig weiter wie Sie denken, ich will Sie nicht durcheinanderbringen”. Nachdem die Prozedur dann fertig war, erklärte er, daß man zuerst das Ende von ListB auf das zweite Element von ListA

hätte zeigen lassen können, um danach den Anfang von ListB hinter das erste Element von ListA zu hängen. Auf die spätere Benotung hatte das aber keinen Einfluß.

Fragen:

- Sie haben im Prozedurkopf Wert-Parameter verwendet. Hätten Sie nicht Referenz-Parameter nehmen müssen, da Sie die Liste doch verändern?
  - Nein, da der Listenanfang nicht verändert wird.
- Warum verwendet man überhaupt Prozeduren bzw. Funktionen?
  - Mehrfachverwendbarkeit im gleichen Programm, dadurch Vermeidung von Code-Redundanz
  - Strukturierung des Programms, Stichwort “prozedurale Programmentwicklung”
  - Wiederverwendbarkeit in anderen Programmen, Stichwort “Bibliothek”, sinnvoll nur bei “sauberer” Programmierung, d.h. Datenaustausch nur über die Parameter und nicht über globale Variablen
- Was versteht man unter dem Modulkonzept? Wie ist die Abgrenzung zum Prozedurkonzept?
  - Aufbau: Schnittstelle, Implementationsteil, ggf. Initialisierungsteil
  - Ein Modul stellt eine Einheit aus Datenstrukturen und Operationen dar, während durch Prozeduren nur Operationen zusammengefaßt werden.
  - Der Zugriff auf die Datenstrukturen eines Moduls erfolgt nur über die in der Modul-Schnittstelle definierten Prozeduren und Funktionen, während bei Prozeduren auch über globale Variablen zugegriffen werden kann (aber nicht zugegriffen werden sollte).
  - Stichwort “Geheimnisprinzip”
- Könnte man eine (externe) Prozedur, die mit ihrer Umgebung nur über die Schnittstelle in Verbindung tritt, als Modul bezeichnen?
  - Ja.
- Was versteht man unter partieller Korrektheit?
  - Definition 13.2 und weitergehende Erläuterungen insbesondere den Unterschied zur totalen Korrektheit, Terminierung
- Wie zeigt man die partielle Korrektheit eines Programmes?
  - Mit Hilfe der Hoare-Regeln (sprich Hohr-Regeln!)
- Wie sehen die Hoare-Regeln aus?
  - Der Aufbau ist im Prinzip wie ein Bruch. Im “Zähler” stehen eine oder mehrere Prämissen. Wenn sämtliche Prämissen erfüllt sind, gilt auch der “Nenner”.
- Welche Eigenschaften muß / sollte eine Schleifen-Invariante haben?
  - Sie muß natürlich erst einmal die Invarianten-Eigenschaft erfüllen. Außerdem sollte sie aber die Funktion der Schleife möglichst genau darstellen. Eine Schleifen-Invariante "true" ist z. B. nicht sinnvoll (aber möglich).
- Ist die Schleifen-Invariante eindeutig?
  - Nein.
- Wann gilt die Schleifen-Invariante?
  - Sie gilt vor Beginn der Schleife und bei jeder Beendigung des Schleifenrumpfes, insbesondere also nach Beendigung der Schleife.
- Die Schleifen-Invariante gilt also vor Beginn der Schleife und bei Beendigung der Schleife. Wie sieht es zwischendurch aus, d.h. während der Bearbeitung des Schleifenrumpfes?
  - Dort gilt die Schleifen-Invariante normalerweise nicht.
- Warum gelten die Hoare-Regeln?
  - Bei dieser Frage wurde u. a. die while-Regel näher betrachtet. Dabei wurden u. a. die Unterschiede von “Zähler” und “Nenner” herausgearbeitet (im “Zähler” steht nur der Schleifenrumpf, im “Nenner” die gesamte Schleife, . ..)

Gedächtnisprotokoll Vordiplomsprüfung Praktische Informatik

Datum: August 1995
Prüfer: Prof. Dr. Six
Kurs: 1612, Version WS 94/95
Dauer: ca. 25 Minuten
Note: 1,0

Prüfungsablauf:

Nach ein klein wenig Smalltalk zu Beginn (wie weit im Studium, welche Ziele werden mit dem Studium verfolgt) begann die Prüfung mit der Programmieraufgabe. Danach folgten die Fragen in etwa in der Reihenfolge wie auf der nächsten Seite aufgeführt. Zwischendurch kamen von Prof. Six noch ein paar weitere von der direkten Prüfung abweichende Fragen, z. B. "Arbeiten Sie in Ihrer Firma viel mit Zeigern oder eher weniger?" oder "Verwenden Sie Bibliotheken in Ihrer Firma?".

Programmieraufgabe:

Es sind 2 Listen ListA und ListB mit der üblichen Typdefinition gegeben. ListB soll nach dem ersten Listenelement von ListA eingefügt werden. Dazu skizzierte Prof. Six beide Listen auf einem Blatt Papier und verdeutlichte durch Linien (= Zeiger), wie ListB in ListA eingefügt werden sollte.
Auf meine Nachfrage hin wurde noch festgelegt, daß ListA aus mindestens einem Element besteht, von ListB hingegen ist nichts bekannt.

```
procedure ListZweitEinfuegen (ListA, ListB : tRefElement);
   var
   hilf1, hilf2 : tRefElement;
begin
   { nur wenn ListB nicht leer ist, ist überhaupt etwas zu tun }
   if (ListB <> nil) then
   begin
      hilf1 := ListA^.next;          { next - Zeiger von ListA merken }
      ListA^.next := ListB;          { ListB hinter das 1. Element von ListA hängen }
      hilf2 := ListB;
      while (hilf2^.next <> nil) do     { ListB mit Hilfszeiger durchlaufen }
         hilf2 := hilf2^.next;
      hilf2^.next := hilf1;          { Ende von ListB zeigt auf das 2. Element von ListA }
   end
end;
```

Zu Beginn meines Entwurfs hatte ich nur eine Hilfsvariable vorgesehen und merkte dann, daß ich zwei Hilfsvariablen benötigte. Darauf versuchte Prof. Six, mich auf einen geschickteren Weg zu lotsen, wobei ich mit der einen Variablen ausgekommen wäre. Da ich seinen Hinweis aber nicht gleich verstand, meinte er nur "Machen Sie ruhig weiter wie Sie denken, ich will Sie nicht durcheinanderbringen". Nachdem die Prozedur dann fertig war, erklärte er, daß man zuerst das Ende von ListB auf das zweite Element von ListA

hätte zeigen lassen können, um danach den Anfang von ListB hinter das erste Element von ListA zu hängen. Auf die spätere Benotung hatte das aber keinen Einfluß.

Fragen:

- Sie haben im Prozedurkopf Wert-Parameter verwendet. Hätten Sie nicht Referenz-Parameter nehmen müssen, da Sie die Liste doch verändern?
  - Nein, da der Listenanfang nicht verändert wird.
- Warum verwendet man überhaupt Prozeduren bzw. Funktionen?
  - Mehrfachverwendbarkeit im gleichen Programm, dadurch Vermeidung von Code-Redundanz
  - Strukturierung des Programms, Stichwort "prozedurale Programmentwicklung"
  - Wiederverwendbarkeit in anderen Programmen, Stichwort "Bibliothek", sinnvoll nur bei "sauberer" Programmierung, d.h. Datenaustausch nur über die Parameter und nicht über globale Variablen
- Was versteht man unter dem Modulkonzept? Wie ist die Abgrenzung zum Prozedurkonzept?
  - Aufbau: Schnittstelle, Implementationsteil, ggf. Initialisierungsteil
  - Ein Modul stellt eine Einheit aus Datenstrukturen und Operationen dar, während durch Prozeduren nur Operationen zusammengefaßt werden.
  - Der Zugriff auf die Datenstrukturen eines Moduls erfolgt nur über die in der Modul-Schnittstelle definierten Prozeduren und Funktionen, während bei Prozeduren auch über globale Variablen zugegriffen werden kann (aber nicht zugegriffen werden sollte).
  - Stichwort "Geheimnisprinzip"
- Könnte man eine (externe) Prozedur, die mit ihrer Umgebung nur über die Schnittstelle in Verbindung tritt, als Modul bezeichnen?
  - Ja.
- Was versteht man unter partieller Korrektheit?
  - Definition 13.2 und weitergehende Erläuterungen insbesondere den Unterschied zur totalen Korrektheit, Terminierung
- Wie zeigt man die partielle Korrektheit eines Programmes?
  - Mit Hilfe der Hoare-Regeln (sprich Hohr-Regeln!)
- Wie sehen die Hoare-Regeln aus?
  - Der Aufbau ist im Prinzip wie ein Bruch. Im "Zähler" stehen eine oder mehrere Prämissen. Wenn sämtliche Prämissen erfüllt sind, gilt auch der "Nenner".
- Welche Eigenschaften muß / sollte eine Schleifen-Invariante haben?
  - Sie muß natürlich erst einmal die Invarianten-Eigenschaft erfüllen. Außerdem sollte sie aber die Funktion der Schleife möglichst genau darstellen. Eine Schleifen-Invariante "true" ist z. B. nicht sinnvoll (aber möglich).
- Ist die Schleifen-Invariante eindeutig?
  - Nein.
- Wann gilt die Schleifen-Invariante?
  - Sie gilt vor Beginn der Schleife und bei jeder Beendigung des Schleifenrumpfes, insbesondere also nach Beendigung der Schleife.
- Die Schleifen-Invariante gilt also vor Beginn der Schleife und bei Beendigung der Schleife. Wie sieht es zwischendurch aus, d.h. während der Bearbeitung des Schleifenrumpfes?
  - Dort gilt die Schleifen-Invariante normalerweise nicht.
- Warum gelten die Hoare-Regeln?

- Bei dieser Frage wurde u. a. die while-Regel näher betrachtet. Dabei wurden u. a. die Unterschiede von "Zähler" und "Nenner" herausgearbeitet (im "Zähler" steht nur der Schleifenrumpf, im "Nenner" die gesamte Schleife, ...)

Protokoll: Vordiplom Praktische Informatik, Kurs 1612

Prüfer: **Prof. Dr. Six**
Kursversion: WS 93/94
Note: 1,0

Termin: April '95
Dauer: ca. 30 Minuten

Programmieraufgabe:
Es ist das Maximum eines Feldes von integer-Zahlen durch Divide-and-Conquer-Verfahren zu bestimmen, also rekursiv duch Teilung des Feldes in zwei etwa gleich große Teile.

gegeben:
```
type tindex = 1..N;
     tfeld = array[tindex] of integer;

function maximum (var inFeld: tfeld; inL, inR: tindex): integer;
var LMax, RMax: integer;
    mitte: tindex;
begin
  if inL = inR then
    maximum := inFeld[inL]
  else
  begin
    mitte := (inL + inR) div 2;
    LMax := maximum (inFeld, inL, mitte);
    RMax := maximum (inFeld, mitte + 1, inR);
    if LMax > RMax then
      maximum := LMax
    else
      maximum := RMax
  end
end;
```

(Die Hälften dürfen sich deshalb nicht überlappen, weil sonst die Rekursion nicht abbricht.)

Wir sprachen kurz über mein Informatikstudium und meine Vorkenntnisse aus dem Informatikunterricht in der Schule. Schwerpunkte der Prüfung waren dann das Modul-Konzept und Syntax/Semantik/Verifikation/Testen. Die Atmosphäre war gut.

Einige Fragen waren z.B.:
- Was versteht man unter **modularer Programmierung**, welche Kennzeichen hat sie ?
  (z.B. Gliederung, Wiederverwendbarkeit, Änderungsfreundlichkeit, Sicherheit)
- Welche Vorteile hat das Modul-Konzept **gegenüber der prozeduralen Gliederung** ?
- Was versteht man unter dem **Geheimnisprinzip**, was sind dessen Vorteile ?
- Nachfrage: Was gehört in den interface-Teil ? (type etc., aber keine Variablen)

- Was bedeuten Syntax und Semantik ?
- Wie kann die **Syntax** beschrieben werden ?
  (z.B. durch **Syntaxdiagramme** oder kontextfreie Grammatiken)
- Wie sind **kontextfreie Grammatiken definiert** ?
- Wie wird die **Semantik** eines Programms konkret beschrieben ?
  (Vers.93/94: operationale, denotationale und **axiomatische Semantik** kurz erläutert)
- Wie sieht eine **Programmformel** konkret aus ?
  (Vor- u. Nachbedingung sind mathematische Ausdrücke, evtl. quantifiziert)
- Wie funktioniert ein **Korrektheitsbeweis** (abstrakt) ?
  (Zusammenfügen der elementaren Programmformeln nach den Hoare-Regeln)
- Was versteht man unter **Testen**; was unterscheidet es vom Korrektheitsbeweis ?
- Welche **Testmethoden** gibt es ?
  (statische / dynamische, kontrollflußorientierte / funktionale,
  Anweisungs- / Zweig- / Pfadüberdeckung)
- Wie testet man Schleifen ?
  (**boundary-interior**: kein-, ein- und zweimalige Ausführung des Schleifenrumpfes;
  die zweimalige Ausführung repräsentiert die Wiederholung)

Mündliche Vordiplomprüfung Kurs 01612 „Konzepte imperativer Programmierung“

Prüfer:	Prof. Dr. Hans-Werner Six
Beisitzer:	Dr. Sonja Branskat
Datum:	21.07.2004
Dauer:	25 min
Note:	1,7

Programmieraufgabe:

Die übliche Typdeklaration für eine Lineare Liste. Es sollte eine Prozedur geschrieben werden, die ein Element in dieser Liste sucht und durch Änderung der Verkettung an das Ende der Liste hängt. Die Lösung musste nicht vollständig ausprogrammiert werden. Die meisten Fragestellungen wurden in der Diskussion besprochen.
Als erstes war die grafische Darstellung der Änderung in der Liste erwünscht, danach die Diskussion über die Sonderfälle (leere Liste, gesuchtes Element am Anfang oder am Ende), Art der Parameterübergabe und die Erläuterung des Algorithmus.

Theoretischer Teil:

Was ist Syntax?
Wie wird die Syntax von Pascal erklärt?
Was sind die Unterschiede zwischen einem Syntaxdiagramm und einer Grammatik?
Gehört die Deklaration einer Variablen zur Syntax oder Semantik?
Was ist unter Semantik zu verstehen?
Erläutern Sie den Begriff einer Programmformel.
Welche Anweisungen können mit der Programmformel beschrieben werden?
Und die anderen?
Was verstehen Sie unter Korrektheit eines Programms?
Erläutern Sie die partielle und totale Korrektheit eines Programms.
Wobei ist die Terminierung eines Programms besonders wichtig?

Prof. Six ist ein sehr freundlicher und sympathischer Prüfer. Er lässt den Prüfling auf einem falschen Weg nicht „verhungern“, sondern führt ihn behutsam auf den richtigen Weg zurück, ohne das bei diesem Nervosität aufkommen kann.
Seine Benotung ist sehr wohlwollend.
Ich kann Prof. Six nur wärmstens als Prüfer empfehlen.