Tentamen

Programspråksteori (5p)

21 mars 1998

Skrivtid: 9.00-15.00. Inga hjälpmedel
Totalpoäng: 40 (Godkänt 20)

Uppgift 1. (3p)

Visa för de nedanstående typdeklarationerna i Pascal vilken mängd av värden de representerar med hjälp av begreppen kartesisk produkt, disjunkt union och avbildning.

```
type    Color = (red, green, blue, yellow);
        Pixel = array [Color] of 0..256;
        Dot = record
                x : 0..480;
                y : 0..320;
                case hasColor : Boolean of
                        true : (c : Pixel);
                        false : ()
        end;
```

Color = { red, green, blue, yellow}
Ej sammansatta värden
Pixel = { red, green, blue, yellow } -> {0, … , 256 }
En avbildning från färgerna till heltal 0-256
Dot = [0, … , 480] x [0, .. 320] x (Pixel + Unit)
Kartesisk produkt där ett element är en disjunkt union

Uppgift 2. (4p)

Vad kännetecknar en deklaration? Beskriv de grundläggande formerna av deklarationer.

En deklaration är en programfras som producerar bindningar

- *Definition, enkel deklaration för att endast producera bindningar*
- *Samtidig deklaration: D_1 and D_2, oberoende*
- *Sekventiell deklaration: D_1; D_2, D_1 följt av D_2*
- *Rekursiv deklaration: refererar till sig själv*

Uppgift 3. (3p)

Förklara vad det innebär att skapa abstraktioner över deklarationer och ge exempel på hur detta stöds i några programspråk.

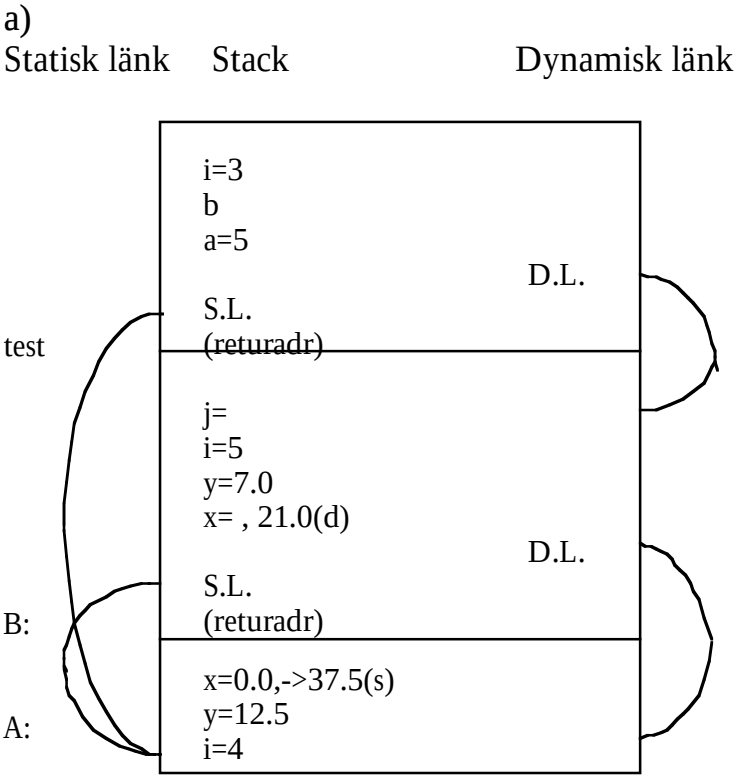
En generisk abstraktion är en abstraktion över en deklaration. En sådan abstraktion har en kropp som är en deklaration, och en instansiering av en generisk abstraktion är en deklaration som producerar bindningar genom att evaluera kroppen.
Ada: generic package (se s. 118 i boken)
ML: funktorer med en liknande effekt (har vi inte tagit upp)

Uppgift 4. (3p)
a) Givet följande programskelett, visa hur anropsstacken ser ut efter anropet av test, om man har statisk räckviddsbindning, visa både statiska och dynamiska länkar.
b) När A anropas, vad skrivs ut i proceduren test och i proceduren A, om man använder sig av statisk respektive dynamisk räckviddsbindning?

```
procedure A;
    integer i; real x, y;
    procedure test (a, b: integer);
        integer i;
    begin
     i:=3;
     x:=float(i)*y;
     write(x,y);
     …
    end test;
    procedure B;
        real x, y; integer i, j;
    begin
     i:=5; y:=7.0;
     call test(i, j);
     …
    end B;
begin
    i:=4; y:=12.5; x:=0.0;
    call B;
    write (x);
end A;
```

Den statiska länken pekar på aktiveringsposten *för (den senaste aktiveringen av) textmässigt omslutande enhet. En metod att implementera statiska räckviddsregler, dvs komma åt variabelvärden i textmässigt omslutande enheter som lagras i respektive enheters aktiveringsposter, är att ha en statisk länk, som man då kan följa för att hitta icke-lokala variablers värden.*
Den dynamiska länken pekar på aktiveringsposten *för omslutande enhet. Det finns en pekare (top) som pekar på aktuell aktiveringspost. Den dynamiska länken behövs för att man vid återhopp från en enhet ska kunna sätta denna pekare att peka på rätt ställe. (D. L. används även för att hitta globala variablers värden om språket har dynamisk räckviddsbindning).*
Återhoppsadressen behövs för att man ska veta var i programkoden i den anropande exekveringen ska fortsätta efter ett återhopp från det anropade underprogrammet. (Det är normalt adressen till instruktionen efter anropet).

a)



b) *Utskrift med statisk räckviddsbindning: 37.5, 12.5*
När y (och x) ej finns lokalt så följs statiska länken, variablerna i huvudprogrammet (närmaste omslutande enhet) kommer att användas i test
Utskrift med dynamisk räckviddsbindning: 21.0, 7.0
När y (och x) ej finns lokalt så följs dynamiska länken, variablerna i B *(närmaste anropande enhet) kommer att användas i test.*

Utskrift från A, statisk räckviddsbindning: 0.0 (eftersom test ändrade x i B, x i A är oförändrat)

***Statisk räckviddsbindning** betyder att en variabels räckvidd avgörs före exekveringen och är statisk, kan ej ändras. Den avgörs utgående från programmets textmässiga struktur. I ett blockstrukturerat språk betyder detta att en variabels räckvidd är den enhet den deklarerats i, plus alla enheter som tetmässigt omsluts av denna enhet. Ett ”hål” i räckvidden uppstår om en innanförliggande enhet deklarerar en egen variabel med samma namn. En referens till en variabel syftar alltså på den ”närmaste” deklarationen av denna variabel i någon omslutande enhet.*
***Dynamisk räckviddsbindning** innebär att en variabels räckvidd avgörs under exekveringen. Räckvidden är dynamisk, kan ändras och vara olika vid olika tillfällen. En variabels räckvidd består av alla enheter som anropas, direkt eller indirekt, av den enhet där variabeln har deklarerats. Räckvidden tar slut när en enhet i anropskedjan deklarerar en egen variabel med samma namn. En referens till en variabel syftar alltså på den senaste deklarationen av variabeln man stöter på om man följer kedjan av anropande enheter (d.v.s. den dynamiska kedjan) bakåt.*

Uppgift 5. (4p)

a) Vad menas med en *variabel på högen* (heap variable)?

Den lagras inte på stacken, den kan skapas och tas bort när som helst. Skapas genom ett kommando.

Anonym - åtkomst via pekare

b) Förklara varför sådana är nödvändiga i så många språk. Vilken funktion fyller de som ”vanliga” variabler ej kan klara av? Jämför kortfattat ML och Pascal med avseende på förekomst och användande av heapvariabler.

Behövs för att bygga upp dynamiska strukturer som träd och listor. De är flexiblare än vanliga variabler eftersom man kan bygga upp strukturer under exekveringens gång.

Funktionella språk har värden som inte är uppdaterinsbara, läggs på högen för att det hela tiden måste skapas nya.

Pascal - via new() och pekare (användarinitierad). Används till dynamiska strukturer (listor, träd mm) måste själv allokera

ML - det mesta av allokeringen sker på högen automatiskt, ingen användardef.

Garbage collection

c) Vad är en ”*dangling reference*”? Ge exempel på hur sådana kan uppkomma. Varför kan sådana vara farliga?

Det är en referens till en variabel som inte längre ”lever”, dvs den innehåller adressen till en dynamisk variabel som har deallokerats.

Farligt eftersom positionen kan ha allokerats till en annan dynamisk variabel.Även om man förändrar den dynamiska variabel som ”dinglar” vilket förstör den nya variabeln.

Uppgift 6. (3p)

Givet förljande program

```
PROGRAM Voo;
    VAR
    i : INTEGER;
    v : ARRAY [1..2] OF INTEGER;
    PROCEDURE Doo (x,y : INTEGER);
    BEGIN
      y := x;
      x := y + 2;
      i := i - 1;
      x := x+y;
    END

BEGIN
    i := 1;
    v[1] := 2;
    v[2] := 3;
    Doo(v[i], i);
    WriteLn (i, v[1], v[2]);
END.
```

Tala om vad som skrivs ut och varför om parametrarna är

a) value-parametrar

b) variable/referens-parametrar

c) value-result-parametrar

a) 0 2 3

Doo förändrar inte parametrarna, däremot förändras värdet av i *som är global variabel.*

b) 1 5 3
Varibel/referens-parametrar, då sker ingen kopiering, utanformell parameter refererar direkt till den variabel som representeras av aktuell parameter. y *och* i *blir alltså två namn på samma variabel i Doo.*
c) 2 6 3
En värde-resultatparameter fungerar så att värdet av aktuell parameter kopieras till formell parameter som fungerar som lokal variabel. När proceduren exekverat klart kopieras värdet av formell parameter över till aktuell parameter.

Uppgift 7. (3p)

Vi har talat om tre olika varianter av polymorfism under kursen:
- ad hoc-polymorfism
- inclusion polumorfism
- parametriserad polymorfism

Beskriv dessa tre former och ge exempel på hur de fungerar.
*ad hoc-polymorfism: overloading, ett (litet) antal olika abstraktioner med samma id (t ex +, *). Tillåts om man entydigt kan bestämma typinfo.*
inclusion polymorfism: arv, t ex undertyper ärver de operationer som finns för övertyper
parametriserad polymorfism: En abstraktion utför liknande saker oberoende av argumenttyp. eof i Pascal, ML t ex fun id(x: 'a) = x.

Uppgift 8. (3p)

a) Vad är ett *undantag* (an exception) i programspråkssammanhang? Vad gör man när ett undantag uppträder?
Ett undantag signalerar att något oförutsett inträffat och att programmet inte kan fortsätta på normalt sätt. När ett sådant inträffar tar man hand om undantaget och specificerar vad man vill ska hända istället för det "normala".
b) Vilka mål vill man uppnå med att införa undantag i ett programspråk?
Man vill få ett robust program som inte bara stannar vid första bästa "fel".

Uppgift 9. (3p)

Ge tre olika kriterier som man bör ta hänsyn till när man väljer ett programspråk för ett större programvaruprojekt. Förklara vad de innebär.
Abstraktioner, över funktioner, procedurer, abstrakta datatyper
Modularitet, separatkompilering, återanvändning
Information hiding, skilja mellan vad och hur

Uppgift 10. (4p)

a) Förklara vad som menas med ett språks syntax och semantik. Förklara även skillnaden mellan abstrakt och konkret syntax.
Syntax handlar om form och notation, medan semantik har att göra med meningen/betydelsen av varje sats i programmet (och för konstruktioner)
Förklara även skillnaden mellan abstrakt och konkret syntax.
Ingår EJ.
b) Ange några fördelar med att använda en formell semantisk metod vid beskrivning av ett språk.
Ingår EJ.

Uppgift 11. (4p)

Vi har diskuterat fyra olika *semantiska principer* som en språkdesigner bör ha i åtanke för att få ett regelbundet språk. Redogör för två av dessa och hur dessa påverkar språkdesign. (Det är inte namnen som är det viktiga utan principerna som sådana.)

Typkompletthetsprincipen

–Alla typer i språket ska ha samma status, parametrar och funktioner t ex ska inte begränsas avseende typer

Abstraktionsprincipen

–Möjlighet att abstrahera över inte bara uttryck (funktioner) och kommandon (procedurer), utan även över deklarationer (Ada generic) och typer (MLs parametriserade typer)

Principen för överensstämmelse

–För deklarationer finns en motsvarande form av parameter och tvärtom. Underlättar abstraktion, dvs att omvandla ett block till t ex en funktion

Kvalifikationsprincipen

–Block ska ingå i alla ”semantiskt meningsfulla ” syntaktiska klasser.

–T ex blockkommandon, blockuttryck (mindre vanligt), blockdeklarationer.

Uppgift 12. (3p)

Är objektorienterade språk bra som nybörjarspråk? Motivera! (Det är motiveringen som bedöms.)

Frågan är hämtad från redovisningarna av lab2. Man kan tänka sig att komma fram till olika slutsatser beroende på vilka karakterisktika man väljer att använda i sin motivering.