UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	INF3110/4110 — Programmeringsspråk
Eksamensdag:	3. desember 2004
Tid for eksamen:	9.00 – 12.00
Oppgavesettet er på 8 sider.	
Vedlegg:	Ingen
Tillatte hjelpemidler:	Alle

Kontroller at oppgavesettet er komplett før
du begynner å besvare spørsmålene.

Dette oppgavesettet består av 4 oppgaver som kan løses uavhengig av hverandre. Dersom du synes noe i oppgaveteksten er uklart, må du gjøre dine egne forutsetninger; sørg bare for at disse er tydelig angitt.

Lykke til!

Innhold

Oppgave 1 Runtime-systemer (vekt 24%)

Anta at vi har et Java-lignede objektorientert språk med virtuelle metoder som kan redefineres i subklasser. Et typisk fragment av dette språket er som vist i figur 1 på neste side.

Metoder er altså (som i Java) virtuelle og kan redefineres i subklasser hvis ikke annet er spesifisert. I eksemplet over er metoden `p` i klassen `B` en redefinisjon av `p` i klassen `A`. Avhengig av verdien av `ra` (`ra` kan referere til et `A`-objekt eller et objekt av en subklasse til `A`), blir en av disse utført ved kallet `ra.p(1,2)`.

1a

Ola og Kari får en ide om å utvide dette språket med «method extension»: parametre kan legges til, lokale variable kan legges til, og setninger kan legges til setningene i metoden som arves. Her er vi bare interessert i parametre. For eksempel, hvis følgende erklæring var en del av klassen `A` (eller `B`) ovenfor,

(Fortsettes på side 2.)

Figur 1: Kode i et Java-lignende språk

```
class A {
    void p (int i, int j){...}
}

class B extends A {
    void p (int i, int j){...}
}

class C {
    void m() {
        A ra;
        // La ra peke på et objekt.
        ra.p(1,2);
    }
}
```

Figur 2: Bruk av «method extension»

```
class A {
    void p (int i,int j) {...}
}

class B extends A {
    void p extends super.p (int k) {...}
}

class C {
    void m() {
        A ra;
        // La ra peke på et objekt.
        ra.p(1,2);
    }
}
```

```
void pp extends p (int k) {...}
```

ville den definere en metode `pp` som har parametrene `(int i, int j, int k)`.

For metoder som sådan synes dette å være en god ide: generelle metoder kan gjenbrukes til å lage spesielle metoder, på samme måte som arv for klasser.

Ola og Kari ønsker å kombinere deres nye mekanisme («method extension») med redefinisjon av metoder i subklasser. Ideen er å tillate at en metoderedefinisjon kan være en «method extension». Dette gjøres ved å legge til en regel om at en metoderedefinisjon kan være en «extension» av den tilsvarende metode i superklassen. De foreslår syntaks som vist i figur 2.

Gitt at kall av virtuelle metoder fungerer som for Java, så vil denne endringen føre til at antall og typer av metodeparametre ikke kan sjekkes på kompileringstidspunktet, men må sjekkes ved kjøretid.

(Fortsettes på side 3.)

Forklar hvorfor.

1b

Ola og Kari prøver å løse dette problemet ved å innføre følgende begrensninger:

1. Metoder uten parameter kan ikke «extendes» til å få parametre.
2. Bare metoder uten parameter kan kalles «remote».

Den andre begrensningen vil gjøre at kallet `ra.p(1,2)` i eksemplet ovenfor ikke er tillatt.

Disse begrensninger er ikke fornuftige, når man tenker på brukbarheten av språket, men vil de hjelpe? Kan man nå sjekke antall og type av parametre ved kall av virtuelle metoder under kompilering, eller vil man fremdeles måtte gjøre dette ved kjøretid?

Hvis du mener at dette må gjøre ved kjøretid, vis dette ved en skisse av et program (gjerne basert på eksemplet over) med et kall på `p` hvor et sjekk på kjøretid er nødvendig.

Oppgave 2 Skop (vekt 24%)

I det Java-lignende programmet i figur 3 på neste side starter utførelsen ved å utføre `main`-metoden i klassen `Program`.

2a

Hva er verdien av `j` ved (3) og (4)

1. hvis språket har statisk skoping («static scoping»)?
2. hvis språket har dynamisk skoping («dynamic scoping»)?

2b

NB! Denne oppgaven besvares ved å tegne på side 5 i oppgavesettet og levere inn dette arket.

Anta i det følgende at språket har statisk skoping.

Sett inn verdier på variable og de riktige pekere («access link» («static link») og «control link» («dynamic link»)) i de følgende to kjøretidsstakkene («runtime stack») med objekter. Kjøretidsstakken i figur 4 er kjøretidsstakken (og objektene) når `pb`-metoden kalles i linjen markert med (1) i figur 3 på neste side. Kjøretidsstakken med objekter i figur 5 er kjøretidsstakken (og objektene) når `ra.p`-metoden kalles i linjen markert med (2) i figur 3. Sett inn verdier og pekere som de er rett før den aktuelle metode slutter.

For forenklingens skyld har ikke `main` og `C`-objektet noen «access link» («statisk link»).

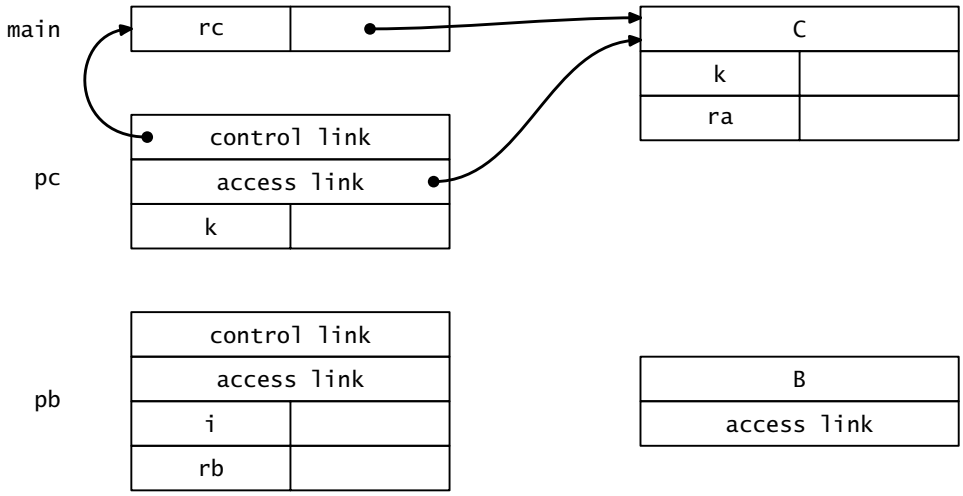
Bemerk at «access link» for et objekt er akkurat som «access link» for en «activation record»: den peker på den instansen som representerer den tekstlig omsluttende blokken (klasse eller metode).

(Fortsettes på side 4.)

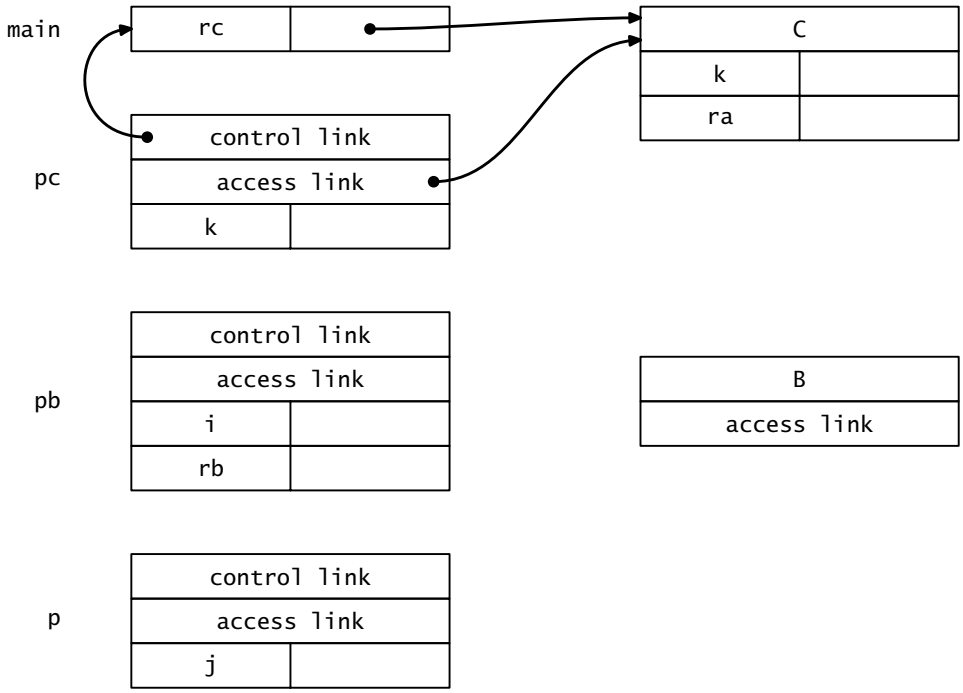
Figur 3: Diverse skop i et Java-lignende språk

```
class Program {
    public static void main(String[] args) {
        C rc=new C();
        rc.pc();
    }
}

class C {
    int k=1;

    class A {
        void p(){ k=3; }
    }

    A ra = new A();

    void pc () {
       int k=2;
       pb();                    // (1)
       ra.p();                  // (2)
    }

    void pb () {
        int i = k;

        class B extends A {
            void p () {
                int j = i;
                                 // (3)
                super.p();
                j = k;
                                 // (4)
            }
        }

        B rb = new B();
        ra = rb;
    }
}
```

(Fortsettes på side 5.)



Figur 4: Kjøretidsstakk ved (1)



Figur 5: Kjøretidsstakk ved (2)

(Fortsettes på side 6.)

2c

Er det mulig å organisere metodekall i en «ren stakk» (en «ren stakk» forstått på den måten at «activation records» kan fjernes ved slutt på en metode)? Forklar hvorfor eller hvorfor ikke.

Oppgave 3 Programmering i ML (vekt 30%)

Vi ønsker å programmere diverse ML-funksjoner som arbeider på et familietre definert med følgende datastruktur:

```
datatype person = NN
                | MAN of (person * person * int)
                | WOMAN of (person * person * int);
```

Varianten `NN` betegner en ukjent person. For `MAN` og `WOMAN` er angitt følgende tre elementer: moren, faren og fødselsåret. Her er eksempler på noen definisjoner av `person`er:

```
val bent = MAN(NN, NN, 1975);
val karen = WOMAN(NN, NN, 1921);
val hilmar = MAN(NN, NN, 1919);
val birger = MAN(karen, hilmar, 1949);
val kirsten = WOMAN(NN, NN, 1947);
val kamilla = WOMAN(kirsten, birger, 1977);
val kristine = WOMAN(kirsten, birger, 1983);
```

3a

Skriv ML-funksjonen

```
fun is_father (p1, p2) = ...;
```

som returnerer `true` om `p1` er far til `p2` og `false` ellers.

3b

Skriv ML-funksjonen

```
fun is_child (p1, p2) = ... ;
```

som returnerer `true` om `p1` er barnet til `p2` og `false` ellers.

3c

Skriv ML-funksjonen

```
fun oldest (p1, p2) = ... ;
```

som returnerer den av `person`ene `p1` og `p2` som er eldst. (Vi kan anta at ingen av dem er `NN`.) Returnér `p1` om de er født samme år.

3d

Skriv ML-funksjonen

```
fun sibling (p1, p2) = ... ;
```

som returnerer `true` om `p1` og `p2` er søsken eller halvsøsken og `false` ellers. (Det er altså nok å ha enten samme mor eller samme far.)

(Fortsettes på side 7.)

3e

Skriv ML-funksjonen

```
fun is_ancestor (p1, p2) = ... ;
```

som returnerer `true` om `p2` er en etterkommer etter `p1`, dvs at `p2` er barn av `p1` eller av en etterkommer av `p1`.

Oppgave 4 Programmering i Prolog (vekt 22%)

Som forrige oppgave dreier denne oppgaven seg om å programmere familier og slektskap. `Person(X,Y,Z,U)` angir at `X` er barn av `Y` (mor) og `Z` (far) og født i år `U`. Her er noen fakta:

```
person(kamilla, kirsten, birger, 1977).
person(kristine, kirsten, birger, 1983).
person(kirsten, herdis, carl, 1947).
person(birger, karen, hilmar, 1949).
person(bent, gerd, jan, 1975).
person(knøttet,kamilla,bent,2006).
```

Regelen `child` angir at `X` er barn av `Y` :

```
child(X,Y) :- person(X,Y,Z,U).
child(X,Y) :- person(X,Z,Y,U).
```

4a

Lag en regel `grandchild(X,Y)` slik at `X` er barnebarn av `Y`.

4b

Nå innfører vi noen nye fakta som skiller mellom menn og kvinner samt om at noen er gift med hverandre.

```
sex(birger,male).
sex(bent,male).
sex(kirsten,female).
sex(herdis,female).
sex(karen,female).
sex(kamilla,female).
sex(hilmar,male).
sex(carl,male).
sex(gerd,female).
sex(jan,male).

married(herdis,carl).
married(carl,herdis,).
married(karen,hilmar).
married(hilmar,karen).
married(kirsten,birger).
married(birger,kirsten).
married(kamilla,bent).
married(bent,kamilla).
```

Bruk disse fakta til å definere relasjonene `father`, `mother`, `grandfather` og `grandmother`:

(Fortsettes på side 8.)

`father(X,Y)`:	X er far til Y
`mother(X,Y)`:	X er mor til Y
`grandfather(X,Y)`:	X er bestefar til Y
`grandmother(X,Y)`:	X er bestemor til Y

4c

Definér `inlaw(X,Y)` og `fatherInlaw(X,Y)`.

`inlaw(X,Y)`:	X er svigerfar eller svigermor til Y (X er far eller mor til Ys ektefelle)
`fatherInlaw(X,Y)`:	X er svigerfar til Y

4d

Definér `ancestor(X,Y)` som finner om Y er en etterkommer etter X.