Cours de Compilation-Exercices

Master d'Informatique M1 2011–2012

14 septembre 2011

1 Introduction à la compilation

1.1 Erreurs de compilation

Les programmes suivants sont-ils corrects? Si non, indiquer à quel moment de la compilation est détectée l'erreur.

```
//1
class A {
    int x?;
    public static void main(){
    }
}
```

```
//2
class A {
  int x;
  public static void main(){
      String s = "bonjour;
      System.out.println(s);
  }
}
```

```
//3
class A {
    public static void main(){
        int y=3;
        System.out.println(x+y);
    }
}
```

```
//4
class A {
    static void m(int x){
       System.out.println(x+1);
    }
    public static void main(){
        int y=3;
        m(y=y+4);
        m(y==y+4);
    }
}
```

```
//5
class A {
    static int m(int x){
        if (x>0) return 1;
        else
            if (x<=0) return 2;
            else return 3;
        return 4;
    }
    public static void main(){
        m(42);
    }
}
```

```
//6
class A {
    static int m(int x){
        return (((4+x) * x/3 +1);
    }
    public static void main(){
        m(42);
    }
}
```

```
//7
class A {
    static int m(int x){
        return (x+1);
    }
    public static void main(){
        int final=42;
        m(final+1);
    }
}
```

```
//8
class A {
    static int m(int x, boolean b, double f){
        return b?x+(int)f:3;
    }
    public static void main(){
        m(4,3.14);
    }
}
```

```
//9
class A {
    static int m(int x, boolean b, double f){
        return b?x+(int)f:3;
    }
    public static void main(){
        m(4.0,true,3.14);
    }
}
```

```
//10
class A {
    public static void main(){
        short s = 35000;
        System.out.println(s);
    }
}
```

```
//11
class A {
    static int m(int x){
        final int y=x+1;
        y++;
        return y;
    }
    public static void main(){
        m(4);
    }
}
```

```
//12
class A {
    public static void main(String args[]){
        int t[]=new int[2];

        t[2]=42;

        System.out.println(t[2]);
    }
}
```

1.2 Correction des compilateurs

On se donne un langage d'expressions arithmétiques de haut niveau décrit par la grammaire suivante :

⟨expr⟩	::= *entier* | *ident* |
	| ⟨expr⟩ ⟨binop⟩ ⟨expr⟩ | ⟨unop⟩ ⟨expr⟩
	| **let** *ident* = ⟨expr⟩ **in** ⟨expr⟩
	| (⟨expr⟩)
⟨binop⟩	::= + | - | * | /
⟨unop⟩	::= -

On se donne un langage machine (MIPS) :

lw	*reg,adr*	charge le contenu de la cellule *adr* dans le registre *reg*
li	*reg,int*	charge la constante *int* dans le registre *reg*
sw	*reg,adr*	sauve le contenu du registre *reg dans la cellule d'adresse adr*
add	*reg1,reg2,reg3*	*reg1* ←*reg2+reg3*
mul	*reg1,reg2,reg3*	*reg1* ←*reg2*reg3*
div	*reg1,reg2,reg3*	*reg1* ←*reg2/reg3*
neg	*reg*	change le signe du contenu du registre *reg*
addi	*reg1,reg2,int*	*reg1* ←*reg2+int*
move	*reg1,reg2*	met le contenu du registre *reg1 dans le registre reg2*

1. Formaliser la sémantique du langage machine proposé sous la forme d'une fonction de transformation des registres et de la mémoire.

2. Le type des arbres de syntaxe abstraite représentant les expressions est la suivante :

```
type binop = Sum | Diff | Prod | Quot
type expr =
    Cst of int
  | Var of int
  | Binop of binop*expr*expr
  | Opp of expr
```

   ici les identificateurs sont représentés par un entier correspondant à l'adresse machine de la variable.

   Donner une fonction de traduction des arbres de syntaxe abstraite représentant les expressions vers une suite d'intructions du langage machine. Combien de registres sont nécessaires pour cette fonction ?

3. Vérifier que la traduction préserve la sémantique des expressions.

1.3 Auto-Compilation

Supposons que l'on veuille définir un nouveau langage L et écrire un compilateur pour ce langage écrit dans le langage L lui-même (c'est la cas des langages C, Java, OCaml par ex.). Expliquer par quels procédés on peut obtenir un tel compilateur. On pourra introduire un certain nombre de compilateurs et/ou interprètes intermédiaires.

1.4 Compilation croisée

La compilation croisée consiste à écrire un compilateur d'un langage L vers un langage machine N écrit dans le langage machine N alors qu'on ne dispose pas de la machine N mais seulement d'une machine M. On suppose que l'on dispose d'un compilateur C du langage L vers le langage machine M écrit en L et d'un compilateur D du langage L vers le langage machine N écrit en L.

On suppose que par auto-compilation on construit un compilateur C' de L vers M écrit en M. Donner alors la chaîne de compilation à partir de D pour obtenir un compilateur de L vers N écrit en N.

1.5 Interprète

On souhaite développer un interprète pour la machine à pile sur le modèle vu en cours:

Code	Description	*sp*	Condition
`ADD`	`P[sp-2]:=P[sp-2]+P[sp-1]`	`sp-1`	`P[sp-2]`, `P[sp-1]` entiers
`SUB`	`P[sp-2]:=P[sp-2]-P[sp-1]`	`sp-1`	`P[sp-2]`, `P[sp-1]` entiers
`MUL`	`P[sp-2]:=P[sp-2]*P[sp-1]`	`sp-1`	`P[sp-2]`, `P[sp-1]` entiers
`DIV`	`P[sp-2]:=P[sp-2]/P[sp-1]`	`sp-1`	`P[sp-2]`, `P[sp-1]` entiers
`PUSHI n`	`P[sp]:=n`	`sp+1`	`n` entier
`PUSHN n`	`P[sp]:=0 ...P[sp+n-1]:=0`	`sp+n`	`n` entier
`PUSHG n`	`P[sp]:=P[gp+n]`	`sp+1`	`n` entier
`STOREG n`	`P[gp+n]:=P[sp-1]`	`sp-1`	`n` entier
`PUSHL n`	`P[sp]:=P[fp+n]`	`sp+1`	`n` entier
`STOREL n`	`P[fp+n]:=P[sp-1]`	`sp-1`	`n` entier
`WRITEI`	imprime `P[sp-1]` sur l'écran	`sp-1`	`P[sp-1]` entier
`START`	Affecte la valeur de `sp` à `fp`	`sp`	
`STOP`	Arrête l'exécution du programme	`sp`	

On ajoute deux instructions `LABEL lab` pour associer le nom `lab` au point de programme courant et `JZ lab` qui dépile une valeur et change le pointeur de code à l'adresse `lab` si la valeur est zéro.

1. Donner un type CAML pour représenter les arbres de syntaxe abstraite du langage machine.
2. Programmer un interprète des programmes de ce langage qui agira sur l'état de la machine (zone et pointeur de code, pile, pointeur de sommet de pile) et effectuera les commandes d'écriture.