Projet de compilation – L3 Informatique
Un compilateur pour un petit langage de programmation

Nicolas Bedon, Arnaud Lefebvre

25 février 2017

1 Présentation générale

On vous demande d'écrire un compilateur générant en sortie du code assembleur pour SIPro, et prenant en entrée un fichier texte contenant un programme écrit dans un petit langage impératif. Voici un exemple de programme écrit dans ce petit langage :

```
// Une variable globale
int g;

int fact(int a) {
  if (a==0)
    return g;
  else
    return a*fact(a-1);
}

int ack(int m, int n) {
    if (m==0)
       { return n+1; }
    if (n==0)
       return ack(m-1,1);
    return ack(m-1,ack(m,n-1));
}

// La fonction main est obligatoire pour faire un programme
int main() {
  print "Hello gentle world!\n";
  g=1;
  print "fact(5)=";
  print fact(5);
  print "\n"; // Affichage: fact(5)=120
  g=3;
  print "fact(5)=";
  print fact(5);
  print "\n"; // Affichage: fact(5)=360
  print 1+2*3==6+2;
  print "\n"; // Affichage: false
  print "ack(3,2)=";
  print ack(3,2);
  print "\n"; // Affichage: ack(3,2)=29
```

```
  print "Goodbye cruel world!\n";
  return 0;
}
```

La boucle `while` n'apparaît pas dans l'exemple, mais devra également être supportée. Sa syntaxe est `while (` *expression booléenne* `)` *instruction*.

Vous constaterez, dans l'exemple, que le petit langage de programmation supporte les variables globales, les variables locales, les fonctions récursives. Le langage est déclaratif : toute variable ou fonction doit avoir été *définie* avant d'être utilisée. Les variables et paramètres de fonctions sont soit des entiers (`int`) soit des booléens (`bool`). On est ici moins exigeant qu'en C, où il suffit que les fonctions et variables soient *déclarées* avant d'être utilisées. Contrairement au C, `print` est ici une instruction qui affiche soit un entier, soit un booléen, soit une chaîne de caractère suivant le type de son opérande. Les chaînes de caractères sont nécessairement des constantes : il n'existe pas de mot-clef correspondant au type des chaînes de caractères, il n'est donc pas possible d'avoir une variable de type chaîne de caractères, ou une fonction retournant une chaîne de caractère ou prenant en argument une chaîne de caractères. Il n'existe pas non plus d'opérateur sur les chaînes de caractères.

Votre compilateur supportera la notion de bloc d'instructions, et sera capable de réaliser du masquage :

```
// f(1,2);
void f(int a, int b) {
  print a; // 1
  print b; // 2
  {
    int a;
    int b;
    a=3;
    b=4;
    print a; // 3
    print b; // 4
    {
      int a;
      a=5;
      print a; // 5
      print b; // 4
    }
  }
  print a; // 1
  print b; // 2
  {
    int a;
    int b;
    a=7;
    b=8;
    print a; // 7
    {
      int a;
      a=9;
      print a; // 9
      print b; // 8
    }
    print b; // 8
  }
  print a; // 1
```

```
  print b; // 2
}
```

Il sera capable de détecter les erreurs de bon sens (syntaxe, typage, nom indéfini, etc).

Finalement, votre compilateur devra être capable de gérer les fonctions locales à d'autres fonctions (comme vu en cours).

```
// f(1) affiche 121
void f(int a) {
  void g(int b) {
    print a;
    print b;
  };
  print a;
  g(2);
}
```

On rappelle qu'ici le travail est plus complexe à réaliser qu'un simple masquage. Quand une variable `a` est utilisée dans une fonction `g` locale à d'autres fonctions, le nom `a` fait référence à la variable locale `a` (si elle existe) de l'enregistrement d'activation le plus récent de la fonction `f` englobant le plus précisément possible `g` et contenant `a` (si cet enregistrement d'activation n'existe pas alors `a` peut éventuellement être une variable globale). Le compilateur a donc ici pour rôle de déterminer l'existence de `f`, de calculer la différence d de niveau d'imbrication entre `g` et `f`, et de générer le code assembleur qui suit d liens d'accès pour retrouver l'enregistrement d'activation contenant `a`.

2 Ce qu'il vous est demandé

Vous devez écrire en C, et en utilisant *flex* et *bison* pour les parties concernant les analyses lexicales et syntaxiques, un compilateur ayant les capacités présentées dans la section précédente.

Bien entendu, votre projet devra être écrit le plus proprement possible : algorithmique adaptée, code clair et commenté.

Vous pouvez étendre le projet si vous le souhaitez, en rajoutant des fonctionnalités par exemple. Cependant, ne le faites que si la base qui vous est demandée est implantée et fonctionne correctement : il est préférable d'avoir un projet qui fait correctement le minimum plutôt que d'avoir un projet étendu dont le minimum demandé ne fonctionne pas. Vous n'implanterez les fonctions locales à d'autres fonctions (partie un peu plus difficile) que quand le reste de ce qui est demandé (plus facile) fonctionnera bien.

Votre projet devra être rendu avec un jeu d'exemples illustrant le mieux possible ses fonctionnalités, ainsi qu'un rapport contenant un rapport de développement et un manuel d'utilisation.

Il devra être développé individuellement ou par binôme, et rendu au plus tard le *A FIXER 2017* au soir, dans une archive au format tar gzippé de nom FrancoisDupontJacquesDurant.tar.gz pour un binôme (si Francois Dupont et Jacques Durant sont vos noms), envoyée en pièce jointe à un courriel de sujet "Projet de compilation L3 Info" à l'adresse de courriel de votre chargé de TP (Nicolas.Bedon@univ-rouen.fr, Arnaud.Lefebvre@univ-rouen.fr). Si vous avez fait le projet en binôme à cheval sur deux groupes de TP, votre projet est à envoyer aux deux chargés de TP. Vous vous mettrez en copie du courriel pour vérifier que vous n'oubliez pas la pièce jointe. L'extraction du fichier d'archive devra produire un répertoire de nom FrancoisDupontJacquesDurant contenant le code source de votre projet, un makefile, des jeux d'exemples et un rapport de projet.

Votre projet fera l'objet d'une soutenance sur machine.