TP de compilation

Luc Brun

16 septembre 2017

1 TP1

Le but de ce premier TP est de définir un interpréteur permettant de manipuler les nombres complexes.

1.1 Interpreteur simple

En vous inspirant du cours constuisez un interpreteur simple permettant de manipuler des expressions arithmétiques entre nombres reels. La structure du fichier Lex sera la suivante :

```
% {
#include <stdio.h>
% }
%option noyywrap nounput batch debug

/* La définition des motifs dont nb (pour nombre) */
%%
{nb} {
  yylval=atof(yytext);
  return NB;
}
\n {printf("car rec: %c\n",yytext[0]);return EOL;}
. {printf("car rec: %c\n",yytext[0]);return yytext[0];}
%%
```

La structure du fichier Bison sera la suivante :

```
%{
  #include <stdio.h>
#include <ctype.h>
  int yylex(void);
#define YYSTYPE double
  int yyerror(const char*);
  %}

%error-verbose

%token NB
%token EOL
%%
input:
/* Ligne vide */
|input ligne
;
ligne: expr '\n' {printf("%f\n",$1);}
     | EOL {}
      ;
/* Le reste de la grammaire*/
%%

#include "lex.yy.c"

int main()
{
  return yyparse();
}

int yyerror( const char *s )
{
fprintf( stderr, "Erreur: %s\n", s);
return 0;
}
```

La compilation de ce premier interpréteur se fera avec gcc. On pourra examiner les conflits et plus généralement le fonctionnement de l'interpréteur en utilisant l'option « - -report all » de bison. Utilisez la priorisation des opérateurs (%left et %right) pour lever les éventuels conflits.

1.2 Interpréteur de nombres complexes

Définissez ou récupérez à partir de la STL une classe complexe permettant de manipuler les nombres complexes. Si vous choisissez de définir la classe, il faudra nottament implémenter :

— Les opérateurs +,-,/,*,<<,=
— Les constructeurs par défaut, par copie et avec les parties réelles et imaginaires (en double).

Modifiez l'interpréteur précédent pour qu'il manipule des complexes plutôt que des doubles.

On concevra dans l'analyseur lexical deux actions prenant en paramètre une partie réelle et une partie imaginaire et renvoyant dans chaque cas le token NB. Il faudra initialiser yylval de façon appropriée dans chaque cas.

L'analyseur grammatical ne doit pas être modifié si ce n'est :

1. L'utilisation de complex plutôt que double pour YYSTYPE,
2. Le remplacement des instructions C (printf, stdio.h) par leurs équivalents C++.

La compilation de cet interpréteur se fera avec g++. On ajoutera à l'interpréteur les fonctionalités suivantes :

1. calcul du module. Par exemple : module 1+i doit renvoyer $\sqrt{2}$,
2. la possibilité de calculer des exponentielles complexes $e^z = e^{Re(z)}.e^{Im(z)i}$. e(1+i) doit donc renvoyer $e\cos(1) + ie\sin(1)$.

2 TP2

L'objectif de ce second TP est d'ajouter à notre interpréteur d'expressions complexes la possibilité d'initialiser des variables et d'utiliser celles-ci dans des expressions. Un exemple de session pourrait par exemple être :

```
a=1+i
a=a/(1-i)
```

bien entendu le nom de variable « i » est à proscrire.

Ceci pose deux problèmes :

1. Il nous faudra gérer une table des symboles pour stocker les valeurs des variables.
2. Jusqu'à présent les valeurs retournées par l'analyseur lexical à l'analyseur syntaxique n'avaient qu'un seul type (double puis complexe dans les exemples précédents). A présent, l'analyseur lexical doit retourner simultanément des noms de variables (des strings) et des valeurs (des complexes).

Le premier problème n'est pas très important et peut se résoudre simplement en utilisant le type unorderedmap de la STL. Le second problème en revanche est plus épineux. Si nous programmions en C, on utiliserait la directive :

```
%union
{
string var;
complex val;
}
```

Malheureusement, C++ interdit de déclarer des types complexes dans des unions. Nous avons toujours la possibilité de passer par des pointeurs. Toutefois cette solution nous oblige à gérer à la main des allocations/deallocations de variables. Heureusement, la dernière version de la STL implémente le type `variant` (disponible à partir de la version 7 de g++).

1. Définissez un type variant egal à un complexe ou une string,
2. Modifiez YYSTYPE pour qu'il corresponde à ce type,
3. Modifiez vos analyseurs lexical (pour reconnaitre des variables) et syntaxique pour que les variables puissent être affectées et intégrées à des expressions. On utilisera pour cela une `unordered_map` en veillant au cas des variables non initialisées dans les expressions (à vous de définir une politique).
4. Ajoutez une expression `display var` à votre grammaire pour afficher le contenu d'une variable.

Remarque : Faites attention d'utiliser la version 7 du compilateur pour disposer du type variant.