Examen de Compilation

Licence Informatique 3

24 mai 2012

Aucun document n'est autorisé. Durée de l'épreuve : 0x5A minutes. La note finale tiendra trés largement compte de la présentation. Le sujet est composé de 4 exercices indépendants, chaque exercice sera noté sur 6 points.

1 bison

```
%{
#include <stdio.h>
#include <ctype.h>
int yylex( void );
int yyerror(char *msg) {
printf("%s", msg);
return 0;
}
%}

%token NB LT END
%right PLUS

%%
INSTR : EXP END
EXP   : EXP PLUS EXP { }
      | NB { }
      | LT { }
%%

int main( void )
{
printf("\n%d\n", yyparse() );
return 0;
}
int yylex(void)
{
 int car;
 car = getchar();
 if ( isdigit( car ) )  return NB;
 if ( isalpha( car ) )  return LT;
 if ( car == '+' )      return PLUS;
 return END;
}
```

On considère la grammaire :

E –> E + E
E –> NB
E –> LT

où NB et LT sont des terminaux qui représentent respectivement des nombres et des lettres. La grammaire est implantée dans la source `parse.y` ci-dessus.

1. Comment compiler cette source pour obtenir un exécutable `parse` ?
2. Décrire avec précision l'action de cet analyseur.
3. Modifier `parse.y` pour tracer les règles de production dans l'ordre des réductions de l'analyseur syntaxique.
4. Aprés modification, quel sera le résultat de la commande : `echo 1+2+c | ./parse` ?
5. Proposer une modification qui gére les nombres de plusieurs chiffres ?

2 Sémantique des instructions

On considère l'opérateur conditionnel du langage C définit avec les symboles " ?" et " :". On rappelle que l'expression :

$$exp_0 \ ? \ exp_1 \ : \ exp_2$$

est évaluée de la façon suivante. Dans un premier temps, la condition exp_0 est évaluée. Si la valeur de exp_0 est nulle alors exp_2 est évaluée et c'est le résultat de l'expression. Si exp_0 n'est pas nulle alors exp_1 est évaluée et c'est le résultat de l'expression.

On notera par exemple le résultat de l'exécution corrrespondant au programme ci-contre :

```
$ ./a.out 0 1 2 3 4 5 6
$ 3
```

(a) Expliquer pourquoi la grammaire suivante est ambigue :

```
E -> E + E
E -> E ? E : E
E -> nb
```

(b) Ecrire une grammaire non ambigue pour décrire les expression aritmétiques sans ambiguité en respectant les règles du langage C.

```
int main( int argc, char* argv[] )
{
int x = atoi( argv[1]);
int y = atoi( argv[2]);
int z = atoi( argv[3]);
int X = atoi( argv[4]);
int Y = atoi( argv[5]);
int U = atoi( argv[6]);
int V = atoi( argv[7]);
int r;
r = x ? y : z ? X : Y ? U : V ;
printf("%d \n", r );
return 0;
}
```

3 flex

On considère la source `scan.lex` qui termine le texte du sujet.

1. Comment obtenir un exécutable `scan` à partir de `scan.lex` ?
2. Quel est le ésultat de `./scan -c 2 scan.lex` ?
3. Décrire l'action de ce code sur un fichier texte quelconque.
4. Proposer une modification apportant une nouvelle fonctionnalité de votre choix.

4 Gestion des Symboles

```
typedef struct symbole {
  char *key;
  struct symbole * next;
} enrliste, *liste;
```

1. Décrire le rôle d'un gestionnaire de symboles.
2. Ecrire une fonction `liste insert( char *k, liste *l )` qui retourne la position du symbole de clef k dans la liste l. Conformément à l'usage, si le symbole n'est pas présent dans la liste alors une nouvelle entrée est crée dans la liste chainée l.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int  nbc  = 1;
int *val, *cpt;
int  j, nbl = 0;
%}

%%
"#".* ;
[0-9]+ if (!j) nbl++;val[j] = atoi( yytext ); cpt[j++]++;
"\n"   j=0;
.      ;
%%
int
main (int argc, char *argv[])
{
  int opt;
  while ((opt = getopt (argc, argv, "c:")) != -1)
    {
      switch (opt)
        {
        case 'c':
          nbc = atoi (optarg);
          break;
        default:
          fprintf (stderr, "Usage␣:%s␣[-c␣nbc]␣[␣name␣]\n", argv[0]);
          exit (EXIT_FAILURE);
        }
    }

  if (optind < argc)
    stdin = fopen( argv[optind], "r");

  val = (int*) calloc( nbc, sizeof(int) );
  cpt = (int*) calloc( nbc, sizeof(int) );

  yylex();

  fclose(stdin);

  for( j = 0; j < nbc; j++)
        printf("%6.2f␣[%3d]", (float) val[j] / cpt[j] , cpt[j]);
  printf("\nnbl=%d", nbl);

 exit (EXIT_SUCCESS);
return 0;
}
```