TD 1 - Lex / Yacc

Exercice 1

Analyse lexicale	Analyse syntaxique
Ignorer les commentaires d'un programme	Trouver les arguments d'un appel de fonction
Distinguer = et ==	Regrouper des calculs au sein d'une parenthèse
Reconnaître un mot-clé du langage	Touver le verbe dans une phrase en allemand
Regrouper des chiffres en un nombre	Décider si une commande est dans l'alternant d'un if

Exercice 2 - Analyseur lexical

Nombre de mots d'une phrase

```
%noyywrap
%{
    include <stdio.h>
    int nbmots = 0;
%}

mot [a-zA-Z]+
fdl \n

%%

{mot} {nbmots++;}
{fdl} {return 0;}

%%

int main () {
    yylex();
    printf("Il y a %d mots.\n", nbmots);
    return 0;
}
```

Nombre de mots d'un fichier

```
%noyywrap
%{
```

```
    include <stdio.h>
    int nbmots = 0;
%}

mot [a-zA-Z]+
fdf \0

%%

{mot} {nbmots++;}
{fdl} {return 0;}

%%

int main (int argc, char ** argv) {
    FILE * fichier;
    if (argc > 1) {
        fichier = fopen(argv[1], "r");
        if (!fichier) {
            fprintf(stderr, "Erreur à l'ouverture.\n");
            exit(1);
        }
    }
    yyin = fichier;
    yylex();
    printf("Il y a %d mots.\n", nbmots);
    fclose(fichier);
    return 0;
}
```

Plus long mot d'un fichier texte

```
%noyywrap
%{
    include <stdio.h>
    int mbmots = 0;
    char mot[128];
    int max = 0;
%}

mot [a-zA-Z]+
fdf \0

%%
```

```
{mot} { nbmots++;
        if (yyleng > max) {
            max = yyleng;
            strcpy(yytext, mot);
        }
      }
{fdl} {return 0;}

%%

int main (int argc, char ** argv) {
    FILE * fichier;
    if (argc > 1) {
        fichier = fopen(argv[1], "r");
        if (!fichier) {
            fprintf(stderr, "Erreur à l'ouverture.\n");
            exit(1);
        }
    }
    yyin = fichier;
    yylex();
    printf("Il y a %d mots.\n", nbmots);
    printf("Longueur du plus long mot: %d", max);
    fclose(fichier);
    return 0;
}
```

Exerice 3 - Analyseur syntaxique

Q1 - Expressions arithmétiques

Lexer

```
%{
    include "Q1_y.tab.h"
    extern int yylval;
%}

nombre [0-9]+
fdl \n

%%

{nombre} { yylval = atoi(yytext); return NOMBRE; }
[ \t]
```

```
{fdl} { return FDL }
. { return yytext[0]; }

%%
```

Parser

```
%define parse.error verbose
%{
    include <stdio.h>
%}
%token NOMBRE FDL
commencement:
    expression FDL { printf("=%d\n", $$);};
expression:
    NOMBRE {$$ = $1;}
  | '(' expression '+' expression ')' {$$ = $2 + $4;}
  | '(' expression '-' expression ')' {$$ = $2 - $4;}
  | '(' expression '*' expression ')' {$$ = $2 * $4;}
  | '(' expression '/' expression ')'
            { if ($4 != 0) $$ = $2 / $4;
              else fprintf(stderr, "Div by 0\n");
            };

%%

int main (int argc, char ** argv) {
    FILE * fichier;
    if (argc > 1) {
        fichier = fopen(argv[1], "r");
        if (!fichier) {
            fprintf(stderr, "Erreur à l'ouverture.\n");
            exit(1);
        }
    }
    yyin = fichier;
    yyparse();
    fclose(fichier);
    return 0;
}
```

Q2 - Calculatrice à mémoire unique

Lexer

```
%{
    include "Q2_y.tab.h"
    extern int yylval;
%}

nombre [0-9]+
fdl \n

%%

{nombre} { yylval = atoi(yytext); return NOMBRE; }
[ \t]
"MP" {return MP}
"MC" {return MC}
"AF" {return AF}
"M" {return M}
{fdl} { return FDL }
. { return yytext[0]; }

%%
```

Parser

```
%define parse.error verbose
%{
    include <stdio.h>
    int mem = 0;
%}
%token NOMBRE FDL MP MC M AF
start:
    sequence;
sequence:
  | sequence commande FDL
commande:
    MP expression { $$ = mem + $2; }
  | MC { mem = 0; }
  | AF expression { printf("%d", mem); }
expression:
    NOMBRE {$$ = $1;}
  | M { $$ = mem; }
  | '(' expression '+' expression ')' {$$ = $2 + $4;}
  | '(' expression '-' expression ')' {$$ = $2 - $4;}
  | '(' expression '*' expression ')' {$$ = $2 * $4;}
  | '(' expression '/' expression ')'
            { if ($4 != 0) $$ = $2 / $4;
```

```
            else fprintf(stderr, "Div by 0\n");
          };

%%

int main (int argc, char ** argv) {
    FILE * fichier;
    if (argc > 1) {
        fichier = fopen(argv[1], "r");
        if (!fichier) {
            fprintf(stderr, "Erreur à l'ouverture.\n");
            exit(1);
        }
    }
    yyin = fichier;
    yyparse();
    fclose(fichier);
    return 0;
}
```

Q3 - Calculatrice à variables

Lexer

```
%{
    include "Q3_y.tab.h"
    extern int yylval;
%}

nombre [0-9]+
ident [a-z]+
fdl \n

%%

{nombre} { yylval.entier = atoi(yytext); return NOMBRE; }
{ident} { yylval.str = strdup(yytext); return IDENT; }
[ \t]
"MS" {return MS;}
"MC" {return MC:}
"AF" {return AF;}
"M" {return M;}
{fdl} { return FDL; }
. { return yytext[0]; }
```

```
%%
```

Parser

```
%define parse.error verbose
%{
    include <stdio.h>
    int mem = 0;
    struct cleval_t = {char * cle; int valeur;}
    struct tab_cleval_t = {int taille}
    ...
    ...
%}
%token NOMBRE FDL MS M IDENT MC AF
start:
    sequence;
sequence:
  | sequence commande FDL
commande:
    MS IDENT expression { /*Ajouter la clé*/ }
  | MC { mem = 0; }
  | AF expression { printf("%d", mem); }
expression:
    NOMBRE {$$ = $1;}
  | M IDENT{ $$ = /* Récupérer la valeur */ }
  | '(' expression '+' expression ')' {$$ = $2 + $4;}
  | '(' expression '-' expression ')' {$$ = $2 - $4;}
  | '(' expression '*' expression ')' {$$ = $2 * $4;}
  | '(' expression '/' expression ')'
            { if ($4 != 0) $$ = $2 / $4;
              else fprintf(stderr, "Div by 0\n");
            };

%%

int main (int argc, char ** argv) {
    FILE * fichier;
    if (argc > 1) {
        fichier = fopen(argv[1], "r");
        if (!fichier) {
            fprintf(stderr, "Erreur à l'ouverture.\n");
            exit(1);
        }
    }
```

```
    yyin = fichier;
    yyparse();
    fclose(fichier);
    return 0;
}
```

TD2 - Typage

Ex1. Langage d'expressions

```
Expr ::= true | false
      |  n
      |  not(Expr) | Expr and Expr | Expr or Expr
      |  Expr == Expr | Expr < Expr
      |  Expr + Expr | Expr - Expr
      |  Expr * Expr | Expr // Expr
      |  Expr % Expr
```

Q1

Le typage donne une approximation de la cohérence/correction d'un langage.

Exemples:

- true % 10 incorrect, repéré au typage.
- true * (2 < (3 + 1))
- 3 and (2 == 2)

Q2

- Γ : environnement
- P : programme
- T : type

$\Gamma \vdash$ P : t

Jugement de typage:

$$(\text{règle})\frac{\text{prémisse}}{\text{produit}}$$

$$(\text{T})\frac{.}{\Gamma \vdash \mathit{true} : \text{bool}}$$

$$(\text{AND})\frac{\Gamma \vdash e_1 : \text{bool} \;\; \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \text{and} e_2 : \text{bool}}$$

$$(\text{OR})\frac{\Gamma \vdash e_1 : \text{bool} \;\; \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \text{or} e_2 : \text{bool}}$$

$$(\text{NOT})\frac{\Gamma \vdash e : \text{bool}}{\Gamma \vdash \text{not}(e) : \text{bool}}$$

$$(\text{EQ})\frac{\Gamma \vdash e_1 : T \;\; \Gamma \vdash e_2 : TT \in \{\text{bool}, \text{int}\}}{\Gamma \vdash e_1 == e_2 : \text{bool}}$$

$$(\text{INF})\frac{\Gamma \vdash e_1 : int \;\; \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 < e_2 : \text{bool}}$$

$$(\text{INT})\frac{n \in N}{\Gamma \vdash n : int}$$

$$(\text{PLUS})\frac{\Gamma \vdash e_1 : int \;\; \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 + e_2 : int}$$

$$(\text{MINUS})\frac{\Gamma \vdash e_1 : int \;\; \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 - e_2 : int}$$

$$(\text{TIMES})\frac{\Gamma \vdash e_1 : int \;\; \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 * e_2 : int}$$

$$(\text{DIV})\frac{\Gamma \vdash e_1 : int \;\; \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 // e_2 : int}$$

$$(\text{MOD})\frac{\Gamma \vdash e_1 : int \Gamma \;\; \vdash e_2 : int}{\Gamma \vdash e_1 \% e_2 : int}$$

Q3

1. 2 + (3 * 4)

(PLUS)

```
                          3 \in N                    4 \in N
                    (INT)-------------      (INT)-------------
                         \Gamma \vdash 3 : int             \Gamma \vdash 4 : int

       2 \in N          \Gamma \vdash 3 : int    \Gamma \vdash 4 : int
(INT)-------------    (+)-----------------------------
    \Gamma \vdash 2 : int                 \Gamma \vdash 3 * 4

    \Gamma \vdash 2 : int    \Gamma \vdash (3 * 4) : int
(+)---------------------------------
          \Gamma \vdash e_1 + e_2 : int
```

2. $1 < (2 + (3 * 4))$

```
      \Gamma \vdash 1 : int    \Gamma \vdash (2 + (3 * 4)) : int
(INF)---------------------------------------
        \Gamma \vdash 1 < (2 + (3 * 4)) : bool
```

3. $(3 == (2 + 1))$ and $(1 + (2 // 2))$

```
                                    \Gamma \vdash e_1 : int   \Gamma \vdash e_2 : int
          ...                (+)-----------------------------
                                      \Gamma \vdash e_1 + e_2 : int

  \Gamma \vdash (3 == (2 + 1)) : bool    \Gamma \vdash (1 + (2 // 2)) : bool
(AND)-----------------------------------------------------
    \Gamma \vdash (3 == (2 + 1)) and (1 + (2 // 2)) : bool
```

$(1 + (2 // 2))$ n'a pas de règle qui le rende booléen: fini. Le programme n'est pas typable.

4. Non.

Ex2. Langage impératif

```
Expr2 ::= Expr | x
Proc  ::= Com | Com; Proc
Com   ::= Affiche Expr2 | Retient x Expr2
```

Q1.

$$(\text{VAR})\frac{}{\Gamma; x : T \vdash x : T}$$

$$(\text{AFF})\frac{\Gamma \vdash e : T \quad T \in int, bool}{\Gamma \vdash \text{Affiche} e : \text{unit}}$$

$$(\text{RET})\frac{\Gamma \vdash e : T \quad \Gamma \vdash x : T}{\Gamma \vdash \text{Retient x e} : \text{unit}}$$

$$(\text{SEQ})\frac{\Gamma \vdash C1 : \text{unit} \quad \Gamma \vdash P2 : \text{unit}}{\Gamma \vdash C1; P2 : \text{unit}}$$

Q2.

```
...  ...  \Gamma \vdash b :  bool bool \in {int, bool} --------------------
(VAR)----------------- (RET) ---------------- (AFF)-------------------------------
\Gamma \vdash (3 + (2 * 4)) :  int \Gamma \vdash s :  int \Gamma
\vdash C1 :  unit \Gamma \vdash C2 :  unit (RET)-------------------------------------
(SEQ)------------------------------------------------ \Gamma
\vdash R s (3 + (2 * 4)) :  unit \Gamma \vdash R b (saucisse + 2)
== 2 * saucisse; A b :  unit (SEQ)-----------------------------------------------------------
\Gamma;s:int;b:bool \vdash R s (3 + (2 * 4)); R b (saucisse + 2)
== 2 * saucisse; A b :  unit
```

Q3.

Conflit de types. Programme non typable.

```
Com2 ::= Com | Declare x Type
Type ::= int | bool
```

$$(\text{DEC})\frac{\Gamma; x : t \vdash P : unit}{\Gamma \vdash \text{Declare x T;} P : unit}$$

Q4.

$$(\text{DEC})\frac{(\text{SEQ})\frac{s{:}int \vdash Rss{:}unit \quad s{:}int \vdash As{:}unit}{s{:}int \vdash Rss; As{:}unit}}{\Gamma \vdash \text{D s int; R s s; A s ; unit}}$$

Pour garantir que les variables ont été initialisées:

x : T° | T

$$\frac{\Gamma; x : T^\circ \vdash P : \text{unit}}{\Gamma \vdash \text{Decl x T; P} : \text{unit}}$$

Q5.

Easy. A faire.

Q6.

TD4 - Sémantique opérationnelle à grand pas

Notation:

$$\vdash T, S \Downarrow v$$

- T le terme/programme à évaluer
- S l'état du système à l'évaluation (mémoire)
- v une valeur

Entiers binaires

Q1.

- T: expressions (bien typées)
- S, S': $\emptyset$
- v: faux, vrai, z, s(..)

Q2.

$$\frac{}{\vdash z \Downarrow z}$$

$$\frac{\vdash N \Downarrow v}{\vdash s(n) \Downarrow s(v)}$$

$$\frac{\vdash e_1 \Downarrow \text{faux}}{\vdash \text{et}(e_1, e_2) \Downarrow \text{faux}} \quad \frac{\vdash e_2 \Downarrow \text{faux}}{\vdash \text{et}(e_1, e_2) \Downarrow \text{faux}} \quad \frac{\vdash e_1 \Downarrow \text{vrai} \quad \vdash e_2 \Downarrow \text{vrai}}{\vdash \text{et}(e_1, e_2) \Downarrow \text{vrai}}$$

$$\frac{\vdash e_1 \Downarrow z \quad \vdash e_2 \Downarrow v_2}{\vdash \text{plus}(e_1, e_2) \Downarrow v_2} \quad \frac{\vdash e_1 \Downarrow v_1 \quad \vdash e_2 \Downarrow v_2 \quad \vdash \text{plus}(v_1, v_2)}{\vdash \text{plus}(e_1, e_2) \Downarrow s(v_3)}$$

$$\frac{\vdash e_1 \Downarrow z \quad \vdash e_2 \Downarrow z}{\vdash \text{egal}(e_1, e_2) \Downarrow \text{vrai}} \quad \frac{\vdash e_1 \Downarrow s(v_1) \quad \vdash e_2 \Downarrow s(v_2) \quad \vdash \text{egal}(v_1, v_2) \Downarrow \text{vrai}}{\vdash \text{egal}(e_1, e_2) \Downarrow \text{vrai}}$$

Q3.

- $s(s(s(z)))$

$$\dfrac{\dfrac{\dfrac{\vdash z \Downarrow z}{\vdash s(z) \Downarrow s(z)}}{\vdash s(s(z)) \Downarrow s(s(z))}}{\vdash s(s(s(z))) \Downarrow s(s(s(z)))}$$

Q5.

$$\frac{}{\vdash z \Downarrow 0} \quad \frac{\vdash N \Downarrow v}{\vdash s(N) \Downarrow v+1}$$

$$\frac{\vdash e_1 \Downarrow v_1 \quad \vdash e_2 \Downarrow v_2}{\vdash \text{plus}(e_1, e_2) \Downarrow v_1 + v_2}$$

...

Langage impératif

Q1.

- T: expressions, commandes, programme
- S: σ var $\to$ val
- v: entiers, void

Q2. Call by value

$$\frac{}{\vdash N, \sigma \Downarrow N, \sigma} \quad \frac{\sigma(x) = v}{\vdash x, \sigma \Downarrow v, \sigma} \quad \frac{\vdash E, \sigma \Downarrow v, \sigma}{\vdash x = E, \sigma \Downarrow void, (\sigma'; x \to v)}$$

$$\frac{\vdash e_1, \sigma \Downarrow v_1, \sigma_1 \quad \vdash e_2, \sigma_1 \Downarrow v_2, \sigma_2}{\vdash (e_1 + e_2), \sigma \Downarrow v_1 + v_2, \sigma_2} \quad \frac{\vdash e_1, \sigma \Downarrow v_1, \sigma_1 \quad \vdash P_2, \sigma_1 \Downarrow v_2, \sigma_1}{\vdash (e_1; P_2, \sigma) \Downarrow v_2, \sigma_2}$$

Q4. Call by name

$$\frac{}{\vdash (x = E, \sigma) \Downarrow (void, \sigma[x \to E]}$$

$$\frac{\sigma(x) = E \vdash (E, \sigma) \Downarrow (v, \sigma')}{\vdash (x, \sigma) \Downarrow (v, \sigma')}$$

Bonus: Call by need

$$\frac{}{\vdash (x = E, \sigma) \Downarrow (void, \sigma[x \to E]}$$

$$\frac{\sigma(x) = E \vdash (E, \sigma) \Downarrow (v, \sigma')}{\vdash (x, \sigma) \Downarrow (v, \sigma'[])}$$