



République Algérienne Démocratique et Populaire



Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'informatique

Département d'informatique

Rapport projet en Compilation 2

Mini-Compilateur R

Binôme :

- **HAMRAOUI Samira** **201500010543**
- **REBHI Aicha** **161631095389**

Niveau : 1ère Année Master

Groupe : 2

Spécialité : Ingénierie des Logiciels

Chargé de Tp : M^r Ilyes **KHENNAK.**

Année universitaire : 2019-2020

Introduction :

Le but de ce projet est de réaliser un mini-compilateur du langage R, passant par les différentes phases de compilation à savoir, l'analyse lexical avec FLEX, l'analyse syntaxico-sémantique avec BISON, la génération du code intermédiaire ainsi que la génération du code machine.

I. Environnement de développement :

1. FLEX :

C'est un générateur d'analyseur lexical. Ce dernier est un programme permettant de reconnaître les mots d'un langage donné. Etant donné un flux de chaînes de caractères écrit dans un langage donné, l'analyseur lexical permet de segmenter le flux en lexèmes représentant les entités lexicales de ce langage comme des identifiants de variables, les mots clés, les opérateurs, etc.

Afin de générer l'analyseur lexical d'un langage « L », Flex prend en entrée un fichier descriptif du langage (ayant l'extension « .l ») et génère un fichier (nommé par défaut « lex.yy.c ») qui contient le code source (en C) de l'analyseur lexical. Il suffit de compiler ce code source pour avoir l'exécutable de l'analyseur lexical

2. BISON :

Bison est un parseur qui permet de transformer une grammaire LALR(1) en code C. Il suffit de compiler le code généré (nommé par défaut NomFichier.tab.c) afin de générer un code exécutable qui effectue l'analyse syntaxique. Pour ce faire, Bison prend en entrée un fichier avec l'extension « .y » contenant la grammaire et d'autres instructions nécessaires pour la génération de l'analyseur syntaxique

II. La compilation :

1. Analyse lexicale avec l'outil FLEX.

On définit les entités lexicales à l'aide des expressions régulières afin d'associer chaque programme source leur catégorie lexicale dont il appartient, et de générer la table des symboles. Grâce à l'outil FLEX.

```
%{
#include<stdio.h>
#include "bison.tab.h"
#include <string.h>
#include <math.h>
extern YYSTYPE yylval;
extern int line;
extern int column;
extern YYSTYPE yylval;
extern char id [12];
extern int col;

#define YYSTYPE string;
int cmp=1;
}%
IDF  [A-Z] ([a-z] | [0-9]) *
int  0 | \ ( \ - [1-9] [0-9] * \ ) | [1-9] [0-9] *
float ([0-9] +) ( \ . [0-9] +) | \ ( \ - ( [0-9] +) ( \ . [0-9] +) \ )
char  \ ' . \ '
comment \# . *
boolean FALSE|TRUE
tab \n+\t+
four_spaces \n+"      "+"

%%

"\n" {
    printf("\n");
    column = column+yyleng;yylval.nom = strdup(yytext);
    line++;column=0;
    return (line_br);
}
{comment} {printf("comment \n"); column=1;}
```

```

"if" {printf("if ");column = column+yyleng;yylval.nom = strdup(yytext);return (IF);}
"elseif" {printf("elseif ");column = column+yyleng;yylval.nom = strdup(yytext);return (elseif);}
"else" {printf("else ");column = column+yyleng;yylval.nom = strdup(yytext);return (eLse);}

"+" {printf("+ ");column = column+yyleng;yylval.nom = strdup(yytext);return (plus);}
"-" {printf("- ");column = column+yyleng;yylval.nom = strdup(yytext);return (moin);}
"*" {printf("* ");column = column+yyleng;yylval.nom = strdup(yytext);return (mul);}
"/" {printf("/ ");column = column+yyleng;yylval.nom = strdup(yytext);return (divi);}
">" {printf("> ");column = column+yyleng;yylval.nom = strdup(yytext);return (greater_than);}
"<" {printf("< ");column = column+yyleng;yylval.nom = strdup(yytext);return (less_than);}
"==" {printf("== ");column = column+yyleng;yylval.nom = strdup(yytext);return (equal);}
"!=" {printf("!= ");column = column+yyleng;yylval.nom = strdup(yytext);return (not_equal);}
">=" {printf(">= ");column = column+yyleng;yylval.nom = strdup(yytext);return (gt_equal);}
"<=" {printf("<= ");column = column+yyleng;yylval.nom = strdup(yytext);return (lt_equal);}
"<-" {printf("<- ");column = column+yyleng;yylval.nom = strdup(yytext);return (aff);}
"(" {printf("(" );column = column+yyleng;yylval.nom = strdup(yytext);return (parent_ouvert);}
")" { printf(")");
    column = column+yyleng;yylval.nom = strdup(yytext);
    return (BRACE_R);
}
"{" {
    printf(" {");
    column = column+yyleng;yylval.nom = strdup(yytext);
    return (BRACE_L);
}
"\"" {printf("\" ");column = column+yyleng;yylval.nom = strdup(yytext);return (parent_ferme);}
"[" {printf("[ ");column = column+yyleng;yylval.nom = strdup(yytext);return (cr_ouvert);}
"]" {printf("] ");column = column+yyleng;yylval.nom = strdup(yytext);return (cr_ferm);}
"," {printf(", ");column = column+yyleng;yylval.nom = strdup(yytext);return (virgule);}
"\'" {printf("' ");column = column+yyleng;yylval.nom = strdup(yytext);return (cote);}
"." {printf(". ");column = column+yyleng;yylval.nom = strdup(yytext);return (twopoint);}

"FOR" {printf("FOR ");column = column+yyleng;yylval.nom = strdup(yytext);return (FOR);}
"IN" {printf("IN");column = column+yyleng;yylval.nom = strdup(yytext);return (IN);}
"WHILE" {printf("WHILE ");column = column+yyleng;yylval.nom = strdup(yytext);return (WHILE);}
"INTEGER" {printf("INTEGER ");column = column+yyleng;yylval.nom = strdup(yytext);return (mc_INTEGER);}
"NUMERIC" {printf("NUMERIC ");column = column+yyleng;yylval.nom = strdup(yytext);return (mc_NUMERIC);}
"CHAR" {printf("CHAR ");column = column+yyleng;yylval.nom = strdup(yytext);return (mc_CHAR);}
"LOGICAL" {printf("LOGICAL ");column = column+yyleng;yylval.nom = strdup(yytext);return (mc_LOGICAL);}
"IFELSE" {printf("IFELSE");column = column+yyleng;yylval.nom = strdup(yytext);return (IFELSE);}

(int) {printf("integer ");column = column+yyleng;yylval.entier = atoi(yytext);return (INTEGER);}
(float) {printf("Numeric ");column = column+yyleng;yylval.reel = atof(yytext);return (NUMERIC);}
(char) {printf("char ");column = column+yyleng;yylval.nom = strdup(yytext);return (CHAR);}
{IDF} {
    printf("IDF ");
    column = column+yyleng;
    if(yyleng<l2) { yylval.nom = strdup(yytext);
        return (idf);}
    else printf("la taille de IDF est superieur la taille maximale\n");
}

" " {printf(" ");column = column+yyleng;}
. {printf("erreur lexical a la ligne %d\n undefined token %c \n",line,yytext[0]);exit(0);}

%%
int yywrap () {return 1;}

```

Figure II.1 : Définition des entités lexical sous FLEX.

2. Analyse syntaxico-sémantique avec l'outil BISON.

Pour compiler l'analyseur syntaxico-sémantique, on doit d'abord écrire la grammaire associée au langage qu'on a défini déjà en FLEX. On doit donc spécifier dans le fichier BISON, les différentes règles de grammaires, ainsi que les règles de priorités des opérateurs, tout en rajoutant avec, leurs routines sémantiques.

a. L'analyse syntaxique :

Dans cette partie, on a défini la structure générale grammaticale du langage.

- **L'affectation :**

Dans cette partie, on a traité les cas de l'affectation avec les trois formes de déclaration :

Forme 1 : Type <List-IDF> (figure II.2.a.1)

Forme 2 : Type IDF \leftarrow Valeur (figure II.2.a.2)

Forme 3 : IDF \leftarrow Valeur (figure II.2.a.3). De ce cas-là, la déclaration peut être une affectation.

Tous en spécifiant les type des variables déclarées. On a traité aussi les déclarations des tableaux (figure II.2.a.4).

Quelques exemples de déclarations :

```
|mc_INTEGER idf
{
    if(ts == NULL) ts = ts_create(ts,100);

    if(ts_get(ts,$2.nom) == NULL ){
        ts_value_t* value = (ts_value_t*) malloc(sizeof(ts_value_t));
        strcpy(value->EntityName, $2.nom);
        strcpy(value->EntityCode, "variable");
        strcpy(value->EntityType, "int");
        strcpy(value->Entityvalue, "1");
        strcpy(value->Entityvaleur,"");
        ts_put(ts,value->EntityName,value);
    }else{
        {
            printf("\n Erreur line:%d Syntatique :: IDF deja declare idf::%s\n",line,$2.nom);
            exit(0);
        }
    }
}
```

Figure II.2.a.1 : Déclaration d'une variable de forme 1 de type INGER.

```
|mc_NUMERIC idf aff OPERATION_ARITH(  
  
if(ts == NULL) ts = ts_create(ts,100);  
  
    if(ts_get(ts,$2.nom) == NULL ){  
    if(strcmp($4.type,"float") == 0){  
        ts_value_t* value = (ts_value_t*) malloc(sizeof(ts_value_t));  
        strcpy(value->EntityName, $2.nom);  
        strcpy(value->EntityCode, "variable");  
        strcpy(value->EntityType, "float");  
        strcpy(value->Entityvalue, "1");  
        strcpy(value->Entityvaleur, $4.nom);  
        ts_put(ts,value->EntityName,value);  
        insertQuad("<-", $4.nom, "", $2.nom);  
    }else{  
  
        printf("\n Erreur line:%d  Synatgique :: incompatible type  %s <- %s      expected NUMERIC not %s \n",line,$2.nom,$4.nom,$4.type);  
        exit(0) ; }  
    } else{  
  
        printf("\n Erreur line:%d  Synatgique :: double declaration  NUMERIC  %s <- %s \n",line,$2.nom,$4.nom);  
        exit(0);  
    }  
}
```

Figure II.2.a.2 : Déclaration d'une variable de forme 2 de type NUMERIC.

```

AFFECTATION : idf aff OPERATION_ARITH {
    if(ts == NULL) ts = ts_create(ts,100);

    if(ts_get(ts,$1.nom) == NULL ){
        insertQuad("<-$3.nom","", $1.nom);
        ts_value_t* value = (ts_value_t*) malloc(sizeof(ts_value_t));
        strcpy(value->EntityName, $1.nom);
        strcpy(value->EntityCode, "variable");
        strcpy(value->EntityType, $3.type);
        strcpy(value->Entityvalue, "1");
        strcpy(value->Entityvaleur, $3.nom);

        ts_put(ts,value-> Entityvalue,value);
    }else{
        ts_value_t* value =(ts_value_t*)ts_get(ts, $1.nom);
        if(strcmp(value->EntityType,$3.type) == 0){
            insertQuad("<-$3.nom","", $1.nom);
        }
        else{
            printf("\n Erreur line:%d  Syntatique :: incompatible type    %s <- %s expected %s \n",line,$1.nom,$3.nom,value->EntityType);
            exit(0);
        }
    }
}

```

Figure II.2.a.3 : Déclaration d'une variable de forme 3.

```

|mc_INTEGER idf cr_ouvert INTEGER cr_ferm {

    if(ts == NULL) {ts = ts_create(ts,100);    }
    char* name      = $2.nom;

    if(ts_get(ts,name) == NULL ){
        insertQuad("<-$3.nom","", $1.nom);
        ts_value_t* value = (ts_value_t*) malloc(sizeof(ts_value_t));
        strcpy(value->EntityName, name);
        strcpy(value->EntityCode, "Tab");
        strcpy(value->EntityType, "int");
        strcpy(value->Entityvalue, inttostr($4));
        ts_put(ts,value->EntityName,value);
        name = strcat($2.nom,inttostr(i));

    }else{
        printf("\n Erreur line:%d  Syntatique :: idf %s déjà déclaré \n",line,$2.nom);
        exit(0);
    }
    for(int i=1;i< $4; i++){ pour les quad    }
}

```

Figure II.2.a.4: Déclaration d'un tableau des INTEGER.

- **Identificateur :**

```

VALUE : INTEGER { $$nom = inttostr($1); $$type= "int"; }
| NUMERIC { $$nom = floattostr($1); $$type= "float"; }
| CHAR { $$nom = strdup($1.nom); $$type="char"; }
| BOOLEEN { $$nom = strdup($1.nom); $$type="boolean"; }
| idf { if(ts != NULL) {
    if(ts_get(ts, $1.nom) != NULL){
        ts_value_t* value =(ts_value_t*)ts_get(ts, $1.nom);

        $$nom = strdup($1.nom);
        $$type = value->EntityType;
    }else{
        printf("\n Erreur line:%d    syntaxique IDF non declaré    Lors de laaffectation \n  IDF : :nom : %s",line, $1.nom);
        exit(0); } }else {
printf("\n Erreur line:%d    syntaxique IDF non declaré :: AFFECTATION NON CORRECTE \n  IDF :: nom: %s",line, $1.nom);
        exit(0);
    }
}
;

```

Figure II.2.a.5: Les identificateurs.

- **Condition :**

On a traité tous les cas des conditions.

-Condition IF (figure II.2.a.6)

```

IF_STATEMENT : IF  parent_ouvert  CONDITON  parent_ferme  LINEBREAK  BRACE_L  INSTRUCTION  BRACE_R  LINEBREAK { //printf("IF_S ");
    $$nom = strdup($3.nom);
    int last_quad = depiler();
    char quad_chaine[12] ;
    sprintf(quad_chaine,"%d",quad_list_size+1);

    MAJQuad(last_quad,quad_chaine); } ;

```

Figure II.2.a.6: Condition IF.

-Condition IFELSE (figure II.2.a.7)

```

IFELSE_STATEMENT : idf aff IFELSE parent_ouvert parent_ouvert CONDITON parent_ferme virgule  VALUE virgule VALUE parent_ferme{}

```

Figure II.2.a.7: Condition IFELSE.

-Condition ELSEIF (figure II.2.a.8)

```

ELSEIF : elseif CONDITON BRACE_L  INSTRUCTION  BRACE_R  LINEBREAK{
    int last_quad = depiler();
    char quad_chaine[12];
    sprintf(quad_chaine,"%d",quad_list_size+1);
    //printf("MAJQUAD");
    MAJQuad(last_quad,quad_chaine); } ;

```

Figure II.2.a.8: Condition ELSEIF.

-Condition ELSE (figure II.2.a.9)

```
ELSE : LINEBREAK eLse BRACE_L INSTRUCTION BRACE_R | eLse LINEBREAK BRACE_L INSTRUCTION BRACE_R LINEBREAK;
```

Figure II.2.a.9: Condition ELSE.

- **Opération logique :**

```
OPERATION_LOGIC : VALUE OPERATOR_LOGIC VALUE {
    if(IsCompatible($1.type,$3.type)){
        //printf("%s - %s\n",$1.nom,$3.nom );
        sprintf(tempTCond,"Tcond%d",nTempTCond);
        nTempTCond ++;
        $$nom=strdup(tempTCond);
        insertQuad("-", $1.nom,$3.nom,tempTCond);
        tempTCond[0]='\0';
    }else{
        printf("\n Erreur line:%d syntagique incompatible type ')\n",line);
        exit(0);
    }
};

OPERATOR_LOGIC : greater_than {BR = strdup("BMZ");} | less_than {BR = strdup("BPZ");} |
equal {BR = strdup("BNZ");} | not_equal {BR = strdup("BZ");} |
gt_equal {BR = strdup("BM");} | lt_equal {BR = strdup("BP");} ;
```

Figure II.2.a.10: Les opérateurs logiques.

- **Opération arithmétique :**

Ici on a traite tous les cas des opérateurs arithmétiques talque l'addition, la soustraction, la multiplication, la division ainsi que le modulo. Dans cette exemple (Figure II.2.a.11), ce figure le traitement d'opérateur d'addition.

```
OPERATION_ARITH :
    OPERATION_ARITH plus OPERATION_ARITH {
        if(IsCompatible($1.type,$3.type)){
            sprintf(tempC,"T%d",nTemp);
            nTemp++;
            $$nom=strdup(tempC);
            $$type = strdup($1.type);
            tempC[0]='\0';
            insertQuad ("+", $1.nom,$3.nom,$$.nom);
        }else{
            printf(" Erreur line:%d syntagique type no compatiblite vous voulez additioner %s avec %s :') ",line,$1.type,$3.type);
            exit(0);
        }
    } |
```

Figure II.2.a.11: Les opérateurs arithmétique.

b. L'analyse sémantique :

Cette partie, concerne la sémantique du grammaire (compatibilité des type, les double déclaration).

3. Gestion de la table des symboles.

Cette table de TS été déjà créé lors de la phase 1, on a utilisé les méthodes de Hachage pour la création de cette table. Elle contient l'ensemble des variables et constantes définies par le programmeur.

Nous avons utilisé l'enregistrement suivant :

```
typedef struct {
    char EntityName[20];
    char EntityCode[20];
    char EntityType[20];
    char Entityvalue[20];
    char Entityvaleur[20];
} ts_value_t;

//symbolstable element structure
typedef struct ts_elem_t {
    struct ts_elem_t *next;
    void *data;
    char key[];
} ts_elem_t;

//symbolstabe structure
typedef struct {
    unsigned int capacity;
    unsigned int e_num;
    ts_elem_t **table;
} ts_t;

//Structure used for iterations
typedef struct {
    ts_t *ht;
    unsigned int index;
    ts_elem_t *elem;
} ts_elem_it;
```

Figure II.3.1: Structure utilisé de la table des symboles.

- Les Fonctions/Procédure utilisé :

Nom de fonction/Procédure	Fonctionnalité
ts_t *ts_create(ts_t *symbols_t ,unsigned int capacity)	Création de la table des symboles
void *ts_put(ts_t *symbolst, char *key, void *data)	Ajouter des éléments dans la TS
void *ts_get(ts_t *symbolst, char *key)	Récupérer des éléments à partir de la TS
void *ts_update_value(ts_t *symbolst, char *key, char *type)	Modifier la TS

void *ts_remove(ts_t *symbolst, char *key)	Suppression des éléments de la TS
--	-----------------------------------

Exemple d’affichage des TS :

```

*****
*               Table des symboles               *
*****

```

NomEntite	TYPE	NATURE	TAILLE	VALEUR
A	int	variable	1	5
B	int	variable	1	0
C	int	variable	1	6
X	int	variable	1	6
Y	int	variable	1	T1

Figure II.3.2 : Affichage d’une TS.

4. Génération de code intermédiaire.

Pour ce faire, on a recours à créer des quadruplets.

- Les Fonctions/Procédure utilisé :

Nom de fonction/Procédure	Fonctionnalité
void quadAppend(Quadruplet quad)	Insérer les quadruplet dans la liste
void insertQuad (char* o,char* o1,char* o2,char* r)	Insérer dans les quadruplet
void MAJQuad (int indice , char* operand)	Mise à jour dans le quadruplet
void afficherQuad()	Affichage du quadruplet

Exemple d’affichage du quadruplet :

```
*****
*****QUADRUPLET ***
1 - ( <- , 6 , , X )
2 - ( <- , 6 , , C )
3 - ( BPZ , @6 , ( , ) )
4 - ( / , X , 2 , T1 )
5 - ( <- , T1 , , Y )
6 - ( - , Y , 1 , Tcond1 )
7 - ( BPZ , @9 , Tcond1 , )
8 - ( <- , 5 , , A )
9 - ( <- , 0 , , B )
10 - ( FIN , , , )
```

Figure II.4.1: Affichage quadruplet.

5. Génération de code machine.

Génération en assembleur :

```

|
TITLE : Test

PILE SEGMENT stack
    100 DD dup (?)
PILE ENDS

DATA SEGMENT
    X DW 0
    C DW 0
    ( DW 0
    Y DW 0
    A DW 0
    B DW 0
DATA ENDS

CODE SEGMENT
BEGIN:
    ASSUME SS: PILE, DS: DATA, CS: CODE
    Mov AX,DATA
    Mov DX,AX

    POP AX
    CMP AX , 0
    JGE etiq [ 6 ]
    Mov AX,X
    DIV AX,2
    PUSH AX

etiq [ 6 ] :
    Mov AX,Y
    SUB AX,1
    PUSH AX
    POP AX
    CMP AX , 0
    JGE etiq [ 9 ]

etiq [ 9 ] :
END

```

Figure V : Code assembleur du langage R.

6. Traitement des erreurs.

Cette phase concerne le plus la partie d'analyse sémantique, l'erreur est affichée pour tous grammaire non respecte, qui ne convient pas avec les règles applique.

Exemple : si on déclare une variable deux fois d'un même programme, il nous affichera une erreur de double déclaration. Comme suit :

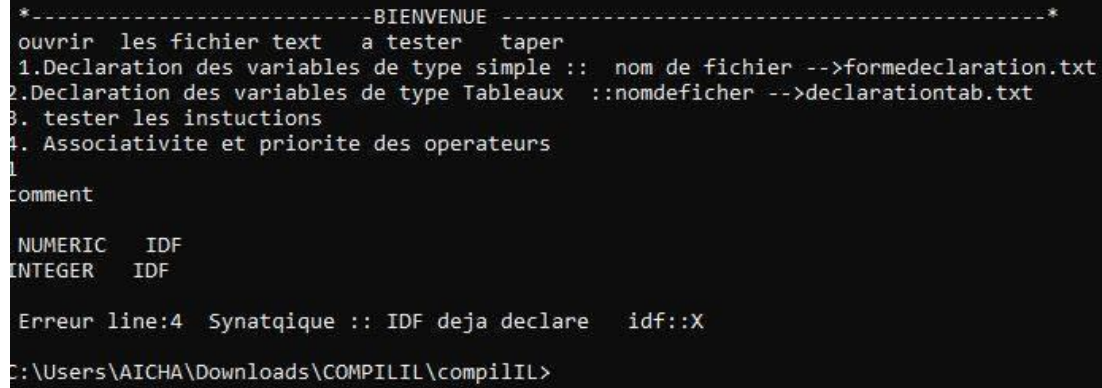
```

# Forme 1
    NUMERIC X
    INTEGER X

```

→ Ceci est une double déclaration

L'erreur ci-dessus est affiché

A screenshot of a terminal window with a black background and white text. The text shows a menu-driven program with options to open a text file, declare simple variables, declare array variables, test instructions, and test operator associativity and precedence. The user has entered '1' to open a file, then 'NUMERIC IDF' and 'INTEGER IDF'. The program then displays an error message: 'Erreur line:4 Syntatique :: IDF deja declare idf::X'. The terminal path is 'C:\Users\AICHA\Downloads\COMPILIL\compilIL>'.

```
*-----BIENVENUE -----*
ouvrir les fichier text a tester taper
1.Declaration des variables de type simple :: nom de fichier -->formedeclaration.txt
2.Declaration des variables de type Tableaux ::nomdeficher -->declarationtab.txt
3. tester les instuctions
4. Associativite et priorite des operateurs
1
Comment
NUMERIC IDF
INTEGER IDF

Erreur line:4 Syntatique :: IDF deja declare idf::X
C:\Users\AICHA\Downloads\COMPILIL\compilIL>
```

Figure VI : Gestion des erreurs.

-Fin rapport-