

Analyseur lexicale:

```
%{
#include <string.h>
#include <stdlib.h>
#include "syntaxe.h"
extern char s[10];
extern int l;
extern int lineNumber,c;
}%
%option noyywrap
nbr [0-9]
entier {nbr}+
identif [a-zA-Z_][0-9a-zA-Z_]*[\s]?
%%
Ecrire {return ECRIRE;}
Algorithme {return ALGO;}
Entier {return ENTIE;}
":" {return DP;}
"+" {return PLUS;}
"-" {return MOIN;}
"*" {return MULT;}
"/" {return DIVS;}
"(" {return PAR_O;}
")" {return PAR_F;}
var {return VAR;}
debut { return DEBUT; }
fin { return FIN; }
[" "\t] { /* rien */ }
{entier} {l=atoi(yytext);return ENTIER; }
{identif} { strcpy(s, yytext);return IDENTIF; }
"=" { return AFFECT; }
";" { return PTVIRG; }
"," {return VIRG;}
"\n" { ++lineNumber; }
. { return yytext[0]; }
%%return 0;
```

Analyseur syntaxique et sémantique:

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
extern FILE* yyin; //file pointer by default points to terminal
int yylex(void); // défini dans progL.cpp, utilise par yyparse()
void yyerror(const char * msg);
int lineNumber; // notre compteur de lignes
char vars[10][20];
int c=0,i;
char s[10];
char tokens[12][8]={"debut","fin","var"," ","","=",";","+", "-", "*", "/"};
void isUsable(char *s,char t[][20]);
void identifTest(char *s,char t[][8],char v[][20]);
char d[255],ll[255],name[200],mm[200];
int l;
FILE *file;
%}

%token DEBUT FIN VAR VIRG// les lexemes que doit fournir yylex()
%token IDENTIF ENTIER AFFECT PTVIRG ALGO ECRIRE
%token PLUS MOIN MULT DIVS PAR_O PAR_F ENTIE DP
%start program // l'axiome de notre grammaire
%%

program : ALGO IDENTIF {strcpy(name,s);file=fopen(strcat(s,".c"),"w");
        fprintf(file,"#include<stdio.h>\n void main() {\n");}declarations DEBU
T listInstr FIN
;
listInstr : listInstr inst
| inst
;
declaration: IDENTIF {identifTest(s,tokens,vars);
        fprintf(file,"int %s ;\n",s);} VIRG declaration
        | IDENTIF{identifTest(s,tokens,vars);
fprintf(file,"int %s ;\n",s);}

declarations: VAR declaration DP ENTIE PTVIRG declarations
        |VAR declaration DP ENTIE PTVIRG;
inst: IDENTIF{isUsable(s,vars);strcpy(d,s);} AFFECT {strcat(d,"=");}
        expression PTVIRG{strcat(d,"");fprintf(file,"%s\n",d);}
        | ECRIRE PAR_O IDENTIF {isUsable(s,vars);}
PAR_F PTVIRG{strcpy(mm,"%d");fprintf(file,"printf(\"%s\\",%s);",mm,s);}

;

expression: expression PLUS{strcat(d,"+");} terme
| expression MOIN{strcat(d,"-");} terme
| terme

;
terme: terme MULT{strcat(d,"*");} facteur
```

```
| terme DIVS {strcat(d,"/");}facteur
| facteur
;

facteur: PAR_0 expression PAR_F{strcpy(l1,"(");strcat(l1,d);strcat(l1,"");
strcpy(d,l1);}
| ENTIER {itoa(l,l1,10);strcat(d,l1);}
| IDENTIF{isUsable(s,vars);strcat(d,s);}
;
%%
void yyerror( const char * msg){
    printf("\nline %d : %s", lineNumber, msg);
}
void createIdentif(char *s,char t[][20])
{
    strcpy(t[c++],s);
}

int isToken(char *s,char t[][8]){
    int i;
    for(i=0;i<10;i++)
        if (strcmp(s,t[i]) == 0)
            return -1;
    return 0;
}

int isDeclared(char *s,char t[][20]){
    int i;
    for(i=0;i<c;i++)
        if (strcmp(s,t[i]) == 0)
            return -1;
    return 0;
}

void isUsable(char *s,char t[][20]){

    if (isDeclared(s,t) == 0)
    {
        yyerror("variable not declared\n");
        exit(-1);
    }
}

void identifTest(char *s,char t[][8],char v[][20]){

    if (isToken(s,t) == -1)
    {
```

```
        yyerror("invalide name for a variable\n");
        exit(-1);
    }
    if (isDeclared(s,v) == -1)
    {
        yyerror("double declaration of variable\n");
        exit(-1);
    }

    createIdentif(s,v);
}

int main(int argc,char ** argv){
if(argc>1) yyin=fopen(argv[1],"r"); // check result !!!
lineNumber=1;
char s[255],n1[200],n2[200];long i;
if(!yyvsparse()){
fprintf(file,"}");
fclose(file);
strcpy(n1,name);
    strcpy(n2,name);
    strcpy(s,"gcc ");
    strcat(s,strcat(n2,".c "));
    strcat(s,"-o ");
    strcat(s,strcat(n1,".exe"));
    i=system(s);
    i=system(n1);
}
return(0);
}
```

Fichier .bat: contient les commandes

```
flex -olexical.c lexical.l
bison -d -osyntaxe.c syntaxe.y
gcc -o prog lexical.c syntaxe.c
prog<code.txt
```

l'exemple (teste):

Algorithme somme

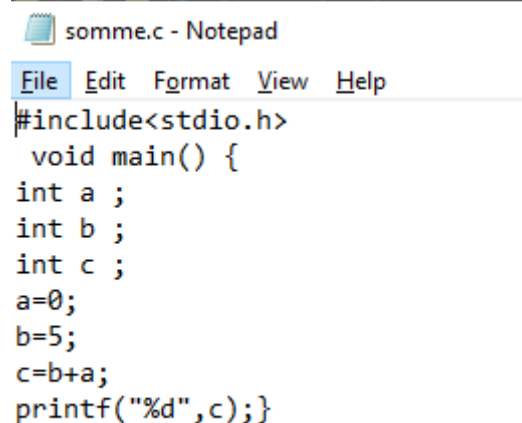
```
var a,b,c : Entier;
```

```
debut  
a=0 ;  
b=5;  
c=b+a;  
Ecrire(c);  
fin
```

Resultat:

```
D:\s6\compilation\tp 5>flex -olexical.c lexical.l  
D:\s6\compilation\tp 5>bison -d -osyntaxe.c syntaxe.y  
D:\s6\compilation\tp 5>gcc -o prog lexical.c syntaxe.c  
D:\s6\compilation\tp 5>prog0<code.txt  
5
```

Le fichier générée:



```
somme.c - Notepad  
File Edit Format View Help  
#include<stdio.h>  
void main() {  
int a ;  
int b ;  
int c ;  
a=0;  
b=5;  
c=b+a;  
printf("%d",c);}
```