

**Team Members:**

Hajar El Boutahiri

Youssef Yousfi

Hiba Aqqaoui

[**Part 2 - Language Design**](https://aui.instructure.com/courses/5704/assignments/40264)

**Professor:**

Violetta Cavalli-Sforza

CSC 3315 02

Part 1: Lexemes:

|  |  |  |  |
| --- | --- | --- | --- |
| Token  (Category) | Regular Expression (formal specification) | Example  Lexeme(s) | Remarks |
| Reserved word  BACK\_RES  MAIN\_RES  SET\_RES  WHILE\_RES  END\_RES  VOID\_RES  INT\_RES  BOOL\_RES  CHAR\_RES  BEGIN\_RES  IF\_RES  ELSE\_RES  CONST\_RES  GOUP\_RES  GOD\_RES  GOL\_RES  GOR\_RES  READ\_RES  PRINT\_RES  PICKF\_RES  BOOL\_LIT | back|main| if [.]+ (else[.]+)+| WHILE|main|SET |END |BEGIN|void| int|bool | const char /\* | char | **if** (a>b) [  b++;  ]  **Else[**  a++;  ]  int main (void){  back 0;  } | back => (Alternative to return in C)  if... else… => for the if-else statement  SET=> to initialize a statement to a value  WHILE => For the loop  END=> Mark the end of the while loop  main=> for the main function  void => for functions that returns nothing and that takes no argument  int => for positive integers  bool => for Boolean values 2 and 3  char => to declare one character  ctchar => to declare string constants  GOUP  GODOWN  GOLEFT  GORIGHT  print  read  =>go in assigned direction  PICKFLOWER |
| Operators  MULT\_OP  ADD\_OP  SUB\_OP  DIV\_OP  EQL\_OP  NEQL\_OP  ASSIGN\_OP  BTOE\_OP  LTOE\_OP  BT\_OP  LT\_OP | /. | +| -| / |= {1,2} |<| >|<=|>= | 1 + 4  5 . 4  b = 2  10 % 2 | . => for multiplication  + => for addition  - => for subtraction  / => for division  = => equality operator  =! =>inequality operator |
| White spaces | [ ]\* | SET a = b ; | “ ” |
| Punctuation  L\_PAREN  R\_PAREN  LCBRK  RCBRK  LSQR  RSQR  S\_COLON  Q\_MARKS  COLON | ,  | ( | )  | { | }  |[| ]  |; | int r **,** j **;**  int main **(**void**)** **{**  **}**  **if (a<b) [**  **]** | () for conditions  {} for statements  ; => mark end of statement  , => Separator  [] => limit conditional statements |
| Positive Decimal Integer Literal  INT\_LIT | 0| [1-9][0-9]\* | SET flower = **15;**  SET grass = **6;**  SET flower\_beds = **9;** | 003 or 02 etc. Are not allowed. In addition to negative integers |
| boolean literal  BOOL\_LIT | True | False | bool a= True; | True  False |
| Strings constants | " [a-zA-Z0-9]\* " | “Hello Flower World” | zero or more character within double quotes |
| ID | [a-zA-Z][\_a-zA-Z0- 9]{0, 15} | Count  Id\_1  incrementor | + An Identifier can’t be a keyword  +Identifiers are case-sensitive  + An identifier can be alphanumeric; however, it must start with a letter or underscore  +Identifier does not include Whitespaces or special characters as: !, @, #, $... |

Part 2: Syntax:

// The code contains global variable declarations / function prototypes, main function, function definitions

Code:: = { < type\_def >  S\_COLON} <main> {< type\_def >  <arg> <function> ) }

//Main Definition

<main>::= INT\_RES  MAIN\_RES L\_PAREN VOID \_RES R\_PAREN  LCBRK <statement> {<statement>}  RCBRK

// Global Variable declaration

<type\_def>::= (CONST\_CHAR | BOOL\_RES | INT\_RES) ID

// function definition

<arg>::=  L\_PAREN (<type\_def > { COLON < type\_def >} )  | VOID\_RES R\_PAREN

<void> ::= VOID \_RES ID  <arg>

<function>::= LCBRK <statement> {<statement>}  RCBRK

// The different statements that a function may contain

<statement>::= <initialize> | <assign> | <conditional statement> | <loop> | <function\_call> | <back>| <predef\_func>

//declare a variable inside a function

<initialize>::= <type\_def> S\_COLON

// assign a value to a variable

<assign>::= SET\_RES ID <EQL\_OP> ( (  ID | | INT\_LIT) [<operator> (  ID | | INT\_LIT) ] ) | ( Q\_MARKS (<sentence> ) Q\_MARKS ) | ( <function\_call> ) ) S\_COLON

<operator>::= ADD\_OP| SUB\_OP| DIV\_OP|MULT\_OP|MOD\_OP

<sentence>::= ID { } { ID { } }

// if.... else... statements

<conditionalstatement>::= IF L\_PAREN <condition> R\_PAREN LSQR THEN <statement> {<statement>} RSQR [ ELSE\_RES LSQR <statement> {<statement>} RSQR ]

//Put a condition; as an example b > c

<condition>::= (ID| INT\_LIT) <conditionaloperator> (ID| INT\_LIT)

<conditionaloperator>::= BT\_OP | LTOE\_OP| BTOE\_OP| LT\_OP | EQL\_OP | NEQL\_OP

//While loop

<loop>::= WHILE\_RES L\_PAREN <condition> R\_PAREN BEGIN\_RES <statement>

{<statement>} END\_RES S\_COLON

// call a function inside our program

<function\_call>::= ID L\_PAREN [ (ID| INT\_LIT){COLON (ID| INT\_LIT) } ] R\_PAREN S\_COLON

// call predefined functions inside our program

<predef\_func>::= ((PRINT\_RES | READ\_RES) L\_PAREN ID R\_PAREN S\_COLON) **|** ( GOU\_RES| GOD\_RES| GOL\_RES| GOR\_RES| PICKF\_RES )L\_PAREN R\_PAREN S\_COLON

// Back statement similar to return statement in C

<back>::= BACK\_RES [<back\_statement>] S\_COLON

back\_statement>::= ID | INT\_LIT

Part 3:

* **A full set of lexemes for closed categories and representative examples categories like user-defined identifiers and constants. These will be formally specified as regular expressions that will be used by the lexer (as shown above).**

\* lexemes:

User defined identifiers: flower, flowerbed, wall, pathway.

User defined constants: WIDTH, HEIGHT, flowerbedloc, flowerloc, grassloc

Predefined functions: PICKFLOWER( ), GOUP( ), GODOWN( ), GOLEFT( ), GORIGHT( ), read(), print()…

Predefined Words: int, main, void, back, char, bool, ctchar , WHILE, END, SET, if, else.

User defined functions: FINDFLOWER(), IFWALLTURN()…

Part 4:

* + **Examples for every grammar rule you used to describe the structure of your program. The parts of the example that are described by rules for non-terminals in the RHSs can be expanded with the examples you give for those non-terminals.**

|  |  |
| --- | --- |
| **Code:** | **GRAMMAR RULES USED:** |
| int glob; | Code:: = { < type\_def >  S\_COLON} <main> {< type\_def >  <arg> <function> ) }  <type\_def>::= (INT\_RES) ID |
| int main (void) { | => In this part of the code, we used the following grammar for the main Function:  Code:: = { < type\_def >  S\_COLON} <main> {< type\_def >  <arg> <function> ) }  <main>::= INT\_RES MAIN\_RES L\_PAREN VOID \_RES R\_PAREN LCBRK <statement> {<statement>} RCBRK |
| bool a; | => In this part of the code, we used the following parts of the EBNF:  <statement>::= <initialize>  <initialize>::= <type\_def> S\_COLON  <type\_def>:: = BOOL\_RES ID |
| SET ab= pick (5, 10); | => In this part of the code, we used the following parts of the EBNF:  <statement>::= <assign>  <assign>::= SET\_RES ID <EQL\_OP> <function\_call>S\_COLON  <function\_call>::= ID L\_PAREN [ INT\_LIT{COLON (INT\_LIT) } ] R\_PAREN S\_COLON |
| back 0; | For this statement we used this part of the EBNF:  <statement>::= <back>  <back>::= BACK\_RES[<back\_statement>] S\_COLON  <back\_statement>::= INT\_LIT |
| } | => In this part of the code, to end the main function we used the following part of the EBNF:  <main>::= INT\_RES  MAIN\_RES L\_PAREN VOID \_RES R\_PAREN  LCBRK <statement> {<statement>}  RCBRK |
| bool PICKFLOWER (int flowerposition, int dimension) { | For this part of the function definition, we used the following parts of the EBNF:  Code:: = { < type\_def >  S\_COLON} <main> {< type\_def >  <arg> <function> ) }  <arg>::=  L\_PAREN (<type\_def > { COLON < type\_def >} )  | VOID\_RES R\_PAREN  <type\_def>::= (CONST\_CHAR | BOOL\_RES | INT\_RES) ID  <void> ::= VOID \_RES ID  <arg>  <function>::= LCBRK <statement> {<statement>}  RCBRK |
| int count; | For this initialisation we used those parts of the EBNF:  <statement>::= <initialize>  <initialize>::= <type\_def> S\_COLON  <type\_def>:: = INT\_RES ID |
| ctchar hello;  SET hello = “Hello Flower World”;  ctchar found;  SET found = “Flower picked”; | For this initialisation we used this part of the EBNF:  <statement>::= <initialize>  <initialize>::= <type\_def> S\_COLON  <type\_def>:: = CONST\_CHAR ID  <assign>::= SET\_RES ID EQL\_OP Q\_MARKS (<sentence> ) Q\_MARKS |
| print (hello); | In this part, we used the following parts of our EBNF:  <statement>::= <predef\_func>  <predef\_func>::= ((PRINT\_RES | READ\_RES) L\_PAREN ID R\_PAREN S\_COLON) |
| SET count = 0; | For this statement we used the following parts of the EBNF:  <statement>::= <assign>  <assign>::= SET\_RES ID EQL\_OP INT\_LIT S\_COLON |
| WHILE (count < dimension ) | For this statement we used the following parts of the EBNF:  <loop>::= WHILE\_RES L\_PAREN <condition> R\_PAREN BEGIN\_RES <statement>  {<statement>} END\_RES S\_COLON  <condition>::= ID <conditionaloperator> ID  <conditionaloperator>::= LT\_OP |
| BEGIN | For this statement we used the following parts of the EBNF:  <loop>::= WHILE\_RES L\_PAREN <condition> R\_PAREN BEGIN\_RES <statement>  {<statement>} END\_RES S\_COLON |
| if (count == flowerposition)[ | => For this condition we used this part of the EBNF:  <statement>::= <conditional statement>  <conditionalstatement>::= IF L\_PAREN <condition> R\_PAREN LSQR THEN <statement> {<statement>} RSQR [ ELSE\_RES LSQR <statement> {<statement>} RSQR ]  <condition>::= (ID| INT\_LIT) <conditionaloperator> (ID| INT\_LIT)  <conditionaloperator>::= EQL\_OP |
| print (found); | In this part, we used the following parts of our EBNF:  <statement>::= <predef\_func>  <predef\_func>::= ((PRINT\_RES | READ\_RES) L\_PAREN ID R\_PAREN S\_COLON) |
| back 2; | => For this statement we used this part of the EBNF:  <statement>::= <back>  <back>::= BACK\_RES [<back\_statement>] S\_COLON  <back\_statement>::= INT\_LIT |
| ] | =>To end an if statement according to our grammar we need a right square bracket: RSQR:  => For this condition we used this part of the EBNF:  <conditionalstatement>::= IF L\_PAREN <condition> R\_PAREN LSQR THEN <statement> {<statement>} RSQR [ ELSE\_RES LSQR <statement> {<statement>} RSQR ] |
| else [ | => For an Else if statement according to our grammar we need a left square bracket: LSQR:  <conditionalstatement>::= IF L\_PAREN <condition> R\_PAREN LSQR THEN <statement> {<statement>} RSQR [ ELSE\_RES LSQR <statement> {<statement>} RSQR ]  => For else, we don’t need a conditionnal statement |
| SET count = count + 1; | => In this part of the code, we used the following parts of the EBNF:  <statement>::= <assign>  <assign>::= SET\_RES ID EQUAL ( ID | INT\_LIT) { <operator> (ID| INT\_LIT)}  <operator>::= ADD\_OP |
| ] | =>To end an else statement according to our grammar we need a right square bracket: RSQR:  <conditionalstatement>::= IF L\_PAREN <condition> R\_PAREN LSQR THEN <statement> {<statement>} RSQR [ ELSE\_RES LSQR <statement> {<statement>} RSQR ] |
| END; | => In our grammar, a loop is marked by END;  <loop>::= WHILE\_RES L\_PAREN <condition> R\_PAREN BEGIN\_RES <statement>  {<statement>} END\_RES S\_COLON |
| back 3; | => For this statement we used this part of the EBNF:  <statement>::= <back>  <back>::= BACK\_RES [<back\_statement>] S\_COLON  <back\_statement>::= INT\_LIT |
| } | => To end a function definition in our grammar we need a RCBRK:  <function>::= LCBRK <statement> {<statement>} RCBRK |

Part 5:

* + **Examples of simpler and more complex programs. For example:**

1. A program that moves the agent a few steps only:

int main(void){

int positionX;

int positionY;

SET positionX = 3;

SET positionY = 7;

SET i = 0;

WHILE (i<2)

BEGIN

GORIGHT();

SET i = i + 1;

END;

SET i = 0;

WHILE (i<3)

BEGIN

GOUP();

SET i = i + 1;

END;

back 0;

}

1. A program that has an agent pick all flowers from a flowerbed:

int main(void){

int positionX;

int positionY;

int LLx;

int LLy;

int ULx;

int ULy;

int URx;

int URy;

int LRx;

int LRy;

SET LLx = 8;

SET LLy = 1;

SET ULx = 8;

SET ULy = 3;

SET URx = 12;

SET URy = 3;

SET LRx = 12;

SET LRy = 1;

SET positionX = 8;

SET positionY = 0;

SET i = 0;

WHILE (i < LRx)

BEGIN

PICKFLOWER();

GORIGHT();

SET i = i + 1;

END;

GORIGHT();

GOUP();

GOUP();

SET i = 0;

WHILE (i < URy)

BEGIN

PICKFLOWER();

GOUP();

SET i = i + 1;

END;

GOUP();

GOLEFT();

GOLEFT();

SET i = 0;

WHILE (i < LRx)

BEGIN

PICKFLOWER();

GOLEFT();

SET i = i + 1;

END;

GOLEFT();

GODOWN();

GODOWN();

SET i = 0;

WHILE (i < LRx)

BEGIN

PICKFLOWER();

GODOWN();

SET i = i + 1;

END;

back 0;

}

1. A program that navigates the whole Flower World, picking all the flowers and exits:

int positionX;

int positionY;

int flowerCount;

int main(void){

SET positionX = 0;

SET positionY = 0;

SET flowerCount = 0;

ctchar Exit;

SET Exit = "Left The Flower World with ";

ctchar Flowers;

SET FlowerCount = " Flowers";

SET flowerCount = traverse(positionX, positionY);

print(Exit );

print(flowerCount);

back 0;

}

1. a program that cannot exit because the path to the exit is blocked by flowerbeds:

int main(void){

int positionX;

int positionY;

SET positionX = 0;

SET positionY = 0;

bool foundExit;

ctchar noExit;

SET noExit = "Exit Blocked by Flowerbeds";

SET foundExit = traverse(positionX, positionY);

if (foundExit == 3) [

print(noExit);

]

back 0;

}