CS 315

Project 2

Report

Argert Boja 21503525
Ndriçim Rrapi 21500342
Section 3

LANGUAGE NAME : B#

BNF DESCRIPTION

```
start:            stmt ENDSTMT
           | funct_declare ENDSTMT start
           | call_predicate ENDSTMT start
           | stmt ENDSTMT start
           | stmt ENDSTMT COMMENT start
           | COMMENT
           | PROGEND
           ;

stmt:             bool_var ASSIGN bool_expr
           | bool_var ASSIGN BOOLEAN
           | bool_const ASSIGN BOOLEAN
           | int_var ASSIGN LRB ath_expr RRB
           | int_var ASSIGN INT
           | string_var ASSIGN string_expr
           | string_var ASSIGN STRING
           | cond
           | loop_stmt
           | OUTPUT
           | INPUT
           | RETURN
           ;

bool_expr:        call_predicate
           | bool_var connectives bool_expr
           | bool_var connectives bool_var
           | bool_const
           | connectives VARIABLE
           | LRB int_var comparison int_var RRB
           ;

ath_expr:   int_var operation int_var;

operation: ADD | SUB | MULT | DIV;

string_expr:     string_var ADD string_var;


cond: IFSTMT LRB  bool_expr RRB LCB  start RCB cond_tail
      | IFSTMT LRB  bool_expr RRB  cond_tail
      ;

cond_tail: ELSETHEN LCB start RCB
          | ELSETHEN
          ;
```

```
loop_stmt:   LOOP bool_expr LCB start RCB;

funct_declare:   PREDICATEINST LRB param_list RRB LCB start RCB;

bool_var:  BOOLEANTYPE VARIABLE
           | BOOLEAN
           | VARIABLE
           ;

int_var:        INTEGERTYPE VARIABLE
           | INT
           | VARIABLE
           ;

string_var:      STRINGTYPE VARIABLE
           | VARIABLE
           |  STRING
           ;

bool_const:       BOOLEANTYPE CONSTANTS
           | CONSTANTS
           ;

call_predicate:  PREDICATEFUNCT LRB arg_list RRB;

param_list:       PARAMETER
           | PARAMETER COMMA param_list
           |
           ;

arg_list:    ARGUMENT
           | ARGUMENT COMMA arg_list
           |
           ;

connectives:      AND | OR | IMPLY | IFF | EBNB | NEGATION;

comparison:       BEQ | LT | LTE | GT | GTE;
```

General Description of B#

Types

In our language **B#** we have three variable types, namely **booleantype**, **stringtype** and **integertype** and **arraytype.**

Our **booleantype** variable resembles the usual boolean variable we are used to see in languages like JAVA with **TRUE** and **FALSE** as values, yet with one small little addition: ***AMBIGUOUS***. When we cannot define whether the expression is neither TRUE nor FALSE we declare it as AMBIGUOUS in **B#**. Say the expression *"Each of us saw her duck"* – It is not clear whether the word "duck" refers to an **action** of ducking or a duck that is a **bird**. In this case this statement is declared as *AMBIGUOUS.* Also **booleantype** can be declared as constant by putting the keyword **const** as follows **booleantype const.** A constant booleantype's value cannot be changed after it is given a boolean value.

As for the **stringtype** and **integertype** we have not added any special attribute to them since our language is mostly based on proportional logic and these types of variables are simply used to enhance the options of boolean logical expressions.

We decided that our **arraytype** variable would be dynamic so a simple example of its syntax in the language would be:

```
arraytype #arr1[]; // dynamic array | size not decided
#arr[0] = TRUE; // assigning values
```

Conditionals

B# also supports conditional statements and the structure of the conditional body is given in the following example

```
in case( #bool1 &  #bool2) // boolean expression
    #arr[0] = TRUE;
otherwise
   #arr[0] = FALSE;
```

Loops

B# also supports loop statements. We use the reserved keyword **until** to denote the start of a loop. An example of a loop in **B#** would be as follows:

```
until ( #input1 >= 0 ) {
   display "Enter a new number";  //prints a message on the screen
  #input1 = scanin;   // input is assigned value of input from keyboard
}
```

Predicate functions

In B# we can use predicate functions to evaluate boolean logical expressions and return values from them. A simple example of declaring a predicate function would be:

```
declare predictWhatIGetFromProject2() {  // reserved words declare predict followed by alphanumerics
        booleantype #IGet100 = TRUE;
        booleantype #OrNot = FALSE;
       booleantype #value = #IGet100 | #OrNot ;
       return #value;
}
```

Then using its return value as follows:

```
booleantype #myGradeIs100 = predictWhatIGetFromProject2();
```

NONTERMINALS

Following there is a brief explanation of our main nonterminals:

- **start**
  This nonterminal defines the start of the program. It will basically point to either a predicate function (call or declare) or to the **stmt** .
- **stmt**
  It defines all of the statements within our language construct which include boolean, integer and string variable assignments as well as loop statements (**loop_stmt**) , conditionals (**cond**) , **OUTPUT** and **INPUT** statements and the **RETURN** statements for returning from functions.

- **bool_expr**
  It defines all of the propositional logic expressions within our language including **comparisons** and **connectives.**

- **comparisons**
  Includes comparison operation like >= , < , <= , > , >=

- **connectives**
  Includes propositional logical operations like AND, IMPLY , IFF, NEGATION, OR, EBNB.

- **ath_expr**
  It defines all of the arithmetic operations(+,-,/,*) that can be done between integers in **B#.**

- **cond** and **cond_tail**
  These nonterminals define the grammar rule of our conditional statements.
- **loop_stmt**
  It defines the grammar rule of our loop statements
- **funct_declare**
  It defines the grammar rule of declaration of predicate function in our language.
- **call_predicate**
  It defines the grammar rule for calling previously declared predicate functions which means in the flow of the code within **B#** we first use the **funct_declare** rule to declare a functions and then we use **call_predicate** to call this declared function and use the boolean value it returns .

TOKENS

Below is a description of our non-trivial tokens used in designing **B#**

- **IF_STMT** & **ELSETHEN**

These tokens define the syntax for conditional statement, respectively **incase** and **elsethen**. They are used each in **cond** and **cond_tail** nonterminals respectively to assist in the grammar for conditionals

- **LOOP**

This token represents the start of the loop namely starting with **until** literals, and it is used in **loop_stmt** nonterminal.

- **COMMENT**

A token that defines a comment within the code

- **VARIABLE**

A token that defines the construction of the names of our variables. Ex: #var1. Used in **bool_expr** and **ath_expr** to define the start of the name of a variable.

- **AND , OR , IFF , EBNB, NEGATION , IMPLY**

These tokens define all of the propositional logic operations within our language and are used in the **connectives** non-terminal to define propositional logical expressions.

- **PREDICATEINST** , **PREDICATEFUNCT**

These tokens define the instantiation and declaration of predicate functions syntax. They are used respectively in **call_predicate** and **funct_declare** non-terminals to assist in the syntax of defining these functions.

- **INTEGERTYPE, STRINGTYPE, BOOLEANTYPE**

These tokens define the reserved words we used for the different types of our variables. Used throughout the grammar.

- **BEQ, LT, GTE, GT**

These tokens define the reserved literals for comparison operations between integers and are used in **comparisons** non-terminal

- **LRB, RRB, LCB, RCB**

These tokens define the brackets used in our language and are used throughout the grammar.

RULES

The precedence in our grammar is defined by brackets.

Every statement must end with a semicolon.

We had problems with ambiguity and we resolved them by removing some of our double recursions that we had throught the grammar