Bilkent University
CS 315 Project 1
15.10.2023

Member 1
Tuna Saygın
Section 2
22102566
Member 2
Kenan Zeynalov
Section 2
22101007

GojoLink

1.0 BNF Rules

```
<program>::= <function_list>
<function_list>::= <function_declaration> <function_list> |
<main_declaration>
<function_declaration> ::=  function <function_name> <lp> <param_list>
<rp> <openBrace> <stmt_list><closeBrace>
<main_declaration>::= function
main<lp><rp><openBrace><stmt_list><closeBrace>
<stmt_list> ::= <stmt_list><stmt><end_stmt>|<stmt><end_stmt>
<stmt> ::= <assign_stmt>
        | <if_stmt>
        | <function_calling_stmt>
        | <while_stmt>
        | <input_stmt>
        | <output_stmt>
        | <return_stmt>
        | <comment_stmt>
        | <array_stmt>


<return_stmt> ::= return <arithmetic_expr>


<comment_text>::= <string>
<comment_init> ::= #
<comment_stmt>:: = <comment_init> <comment_text>




<function_calling_stmt> ::= <function_name> <lp> <var_list> <rp>|
<function_name> <lp> <rp>
```

```
<var_list> ::=
<arithmetic_expr>,<var_list>|<var>[],<var_list>|<arithmetic_expr>|<var>
[]
<type>::= int | int[]
<function_name> ::= <var>
<param_list>::= <type> <var> | <type> <var> , <param_list>


<while> ::= while
<openBrace> ::= {
<closeBrace> ::= }
<while_stmt>::= <while> <lp> <logical_expr> <rp> <openBrace>
<stmt_list> <closeBrace>




<if_stmt> ::= <if> <lp> <logical_expr> <rp> <openBrace> <stmt_list>
<closeBrace>
            | <if> <lp> <logical_expr> <rp> <openBrace> <stmt_list>
<closeBrace> <else> <openBrace> <stmt_list> <closeBrace>
            | <if> <lp> <logical_expr> <rp> <openBrace> <stmt_list>
<closeBrace> <elif_stmt> <else> <openBrace> <stmt_list> <closeBrace>
<if> ::= if
<else> ::= else
<else_if> ::= elif
<elif_stmt> ::= <else_if> <lp> <logical_expr> <rp> <openBrace>
<stmt_list> <closeBrace> <elif_stmt>
        | <else_if> <lp> <logical_expr> <rp> <openBrace> <stmt_list>
<closeBrace>
<logical_expr> ::= <logical_expr> <op_logical> <compare> | <compare>
<compare> ::= <arithmetic_expr> <op_logical> <term>
<output_stmt> ::= <cout> <outSymbol> <arithmetic_expr>
                    |<cout> <outSymbol> <string>
<input_stmt> ::= <cin><inSymbol><var>
<inSymbol> ::= >>
<outSymbol> ::= <<
<cout>::= cout
<cin> ::= cin
<string> ::= <char> <string> | <digit> <string> | " " <string>
<string_literal> ::= "\"" <string> "\""


<list_assignment> ::= <var><sb_open><integer><sb_close><op_assign>
<list>
<array_expr> ::= <array_length> | <array_get> | <array_set>
<array_length> ::= <var>::
<array_get> ::= <lp> <var>:<arithmetic_expr> <rp>
<array_set> ::= <var>::<arithmetic_expr> = <arithmetic_expr>
<list> ::= <sb_open> list_items <sb_close>
<list_items> ::=  <list_items>,int| int
```

```
<sb_open>::= [
<sb_close> ::= ]
<array_term> ::= <var>.length<lp><rp> | <var>.get<lp><integer><rp>
<array_stmt> ::= <array_expr>


<assign_stmt> ::= <var> <op_assign> <arithmetic_expr>
                |<list_assignment>
<arithmetic_expr> ::=<arithmetic_expr> <op_addsub> <factor> | <factor>
<factor> ::= <factor> <op_muldiv> <mod_term> |<mod_term>
<mod_term> ::= <mod_term> <op_mod> <expo_term> | <expo_term>
<expo_term> ::= <expo_term> <op_expo> <term> | <lp> <arithmetic_expr>
<rp> | <term>
<term> ::= <var> | <integer> | <function_calling> | <array_term>


<op_logical> ::= <op_and> | <op_or>
<op_arithmetic> ::= <op_addsub> | <op_muldiv>
<op_muldiv> ::= <op_mul> | <op_div>
<op_addsub> ::= <op_add> | <op_sub>
<op_assign>::= =
<op_add> ::= +
<op_sub>::= -
<op_mul> ::= *
<op_div> ::= /
<op_mod> ::= %
<op_expo> ::= **
<op_eq> ::= ==
<op_le> ::= <=
<op_ge> ::= >=
<op_lt> ::= <
<op_gt> ::= >
<op_and> ::= &&
<op_or> ::= ||
<rp> ::= )
<lp> ::= (
<end_stmt> ::= ;


<var> ::= <char><var_suffix>
<var_suffix> ::= <var_suffix><digit>|<var_suffix><char>|<char>|<digit>
<unsigned_int>::= <digit><unsigned_int>|<digit>
<integer> ::= <sign><unsigned_int> | <unsigned_int>
<sign> ::= + | -
<digit> ::= 0|1|2|3|4|5|6|7|8|9
<char> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
P | Q | R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g |
h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y |
z
```

2.0 Language Construct

2.1 Functions

All of the written language in the program is list of function declarations. Functions will be declared one by one. However, only constraint is that every function can only call the function above.

Main Function

Main function is where the program begins to execute the statements. Main function must be the last function of the function declaration. It cannot be recursive. and the statement list inside main must end with return statement which must return 0 to function properly just like C.

Function Declaration

In order to call a function one must declare or define the function to what to do in the function. Therefore functions need to be declared along with their outputs.

Functions will be declared as:

```
func <func_name> ( <param_list>){
        <stmt_list>
        <return_stmt>;
}
```

Explanations:

“func” is a reserved keyword that is used to state that the statement is a function declaration.
<func_name> is just like variable. All of its instructions will be stored.
<param_list> is set of parameters. It can be either integer or integer array. Integer array parameters are pass by reference meanwhile integer parameters are pass by value.
<stmt_list> is set of instructions that are executed synchronously.
example:

```
func add(int a, int b){
        return a b;
}+
```

Conventions:

- Every function may have no or more than one parameter.
- Every parameter is either an integer or an array data type.
- Every function does not need to end with return. If it is not end with return and ends the statement it will return 0. This decreases the need to use void functions. Also it eliminates any errors that relates with no return value.
- Every function can only call the functions above or itself.
- Every function returns int.(Important!)
- Integer array parameters are passed by reference while integer parameters are passed by value.

Function Calling
Functions can be called as single statements or as a term for arithmetic expressions since it can only return integers. Functions are called as:
Explanation of BNF:
<function_name>(<var_list>);
<function_name>();
Explanations:
<function_name> is the name of the function that is declared.
<var_list> is list of variables that consist of either integer or integer array variables.It can also be integer constants or function calls as well
Examples:
a = add(a[], a+ 2023); #”Assume that add is declared and has one array and one integer parameter”

b = add(arr[], add(a[], a + 21));
Parameters: in the PL integer parameters are pass by value whereas integer array are pass by reference.
Return Type: Because the language only works on integers only integer can be returned. If you think that we didn’t return any integer arrays think again! Our functions get arrays as pass by references. So what you give as an input array function is what you get after the function. Hence, Use pass by reference for array outputs.
In the case of void outputs, just returning a dummy integer is enough. Even though you don’t use it and decreases performance, it increases uniformity in the function calls.

Conventions:
- Every called functions must be declared before the function is called.
- Every function call arguments can support arithmetic expressions as well as other callings.
- Every function call will return an integer. Can be used or not is up to the user.
- Every function call can take array arguments and also may take no arguments.

2.2 Loop Statements

Loop statements repeat the written instruction therefore increases the writability by preventing code redundancy.
The loop statements only supports “while” loop
While Loop
While loop is a loop where until the logical expression that is defined as below is false, continue to do the set of instructions defined in the statement list.
```
while (<logic_expr>){
        <stmt_list>
};
```
Explanation:
  “while” is a reserved keyword that states that the statement is a loop statement.
  <logic_expr> logic expression consists of comparison of integers.
  <stmt_list> set of instructions that will be executed repeatedly.

Conventions:

- While loop repeats the statements/instructions until logic expression turn out to be false.
- While loops can have nested loops.
- Because loop is a statement it ends with semicolon. (It may decrease writability but it increases the writability.)

2.3 Control Flows

Control Flow increases the readability of the code by increasing the comprehensibility of the code.

If/Else & Elseless If Statement

If else statements control the flow of the instructions. When the logical expression is true, it will execute the statements inside “if” keyword. When the logical expression is false, it will execute the statements inside “else”.(Similar to C language)

Elseless if statements will check the logical expression. When the logical expression is true, statements inside “if” will be executed, otherwise program will not execute the statements inside if statement and continue with other statements.

Explanation:

```
if (<logical_expr>){
        <stmt_list>
}else{
        <stmt_list>
};
```

“if” and “else” are reserved keywords that indicate whether the statement inside it will be executed or not according to the output of the logical expression.

<logical_expr> is comparison of integers which can be either true or false.

Example:

```
if( 5>2){
        cout<<”It is working very very well”;
}else{
        cout<<”If it prints this statement pigs can fly as well”;
};
…
if (3<0){
        cout<<”not executed”;
};
```

Conventions:

- else statements may either be used or elseless statements can be used as well.
- every “if” or “else” reserved key word must continue with “{“ and ends with “}”.
- Language cannot support empty statement list. Therefore, between “{“ and “}”, There must be additional statement.
- Logical expressions cannot support integer values. Therefore, any arithmetic expression written as logical expression will cause syntax error.

- Because conditional flow statements are statements, they need semicolon to end the statement. If there is no semicolon syntax error will be given where the semicolon is not put.

ElseIf Statements
Else if statements in GojoLink is supported by the keyword “elif”.
Similar to if/else combined statements you can write it in between if block and else block.
Example:

```
if(tuna>98){ #”reaction of a person in an exam in another world”
        cout<<”Tuna is happy\n”;
}elif(tuna == 97){
        cout<<”Tuna is okay\n”;
}
else{
        cout<<”Tuna will be not happy\n”;
};
```

Conventions for else if usage:

- After else if blocks are finished, conditional flow must end with else. If not it will be syntax error. This implementation is similar to switch statements in java where you are required to put default case.
- you can write as many else if blocks as possible.

2.4 Arithmetic Expressions

```
<arithmetic_expr> ::=<arithmetic_expr> <op_addsub> <factor> | <factor>
```

Arithmetic Expressions are used to do mathematical calculations in the program. All operations such as addition, subtraction, multiplication, division, exponential and module operations are included. There is a operator precedence as compiler understand that in this expression:
a+b*c ←b*c should be calculated first and then proceed to the addition operation. Addition and subtraction have lower level precedence compared to the different operations.
In special conditions like (a+b)*c, expressions inside the parentheses are accepted as the highest precedence despite the next operation given.
It also includes that there are can be repetitive usage of parentheses like (a*(a+b)) * c , as here (a+b) will be done and multiplied with ‘a’ as there is one more parentheses, and later multiplied with C. Additionally, mod operation has higher precedence than multiplication and division and exponential operation has higher precedence than mod operations just like C language. To sum up, operator precedence is:

$$paranthesis > exponential > mod > multiplication = division > substraction = addition$$

Conventions:

- Operator precedence handles the operations are carried out in the desired order.
- Usage of the parenthesis like (a+b) is allowed to show precedence of the operation

- Expression should be correctly executed by having semicolons in the ending to satisfy the syntax.
- When parentheses are used they should be closed properly by the user/developer as they can create syntax errors. Therefore every parentheses must be closed in expressions.

2.5 Data Structures & Variables

Data Structures and Variables increase the functionality of the code by storing and using data accordingly. With Data Structures, we can increase the manipulation of the code such as having 'ARRAYS' as a 'list' a [4] = [0, 0, 0,0]. Here instead of initialization of 4 different new integers, we keep them all in the array and can use them later on for any purpose we want. Variables are used for representing values like int, string, double, etc. in our program. Variables need to be initialized beforehand like this:
<var> count; Here we will give the type of the 'count' variable and use it later.

Variables
Starts with alphabetical characters followed by alphanumerical characters. It can also consist of 1 alphabetical character as well. In the language camel case is the preferred choice so we do not support "-" or "_".
Correct usage: anyVariable12445
Wrong usage: x_24
Integer : used for numerical storage of a value.
Integer Array
Integer Array is a collection of data that will be separated by ",". Just like C, the index of every array's first item is 0. So the last element will be length-1. There are 2 important operations in an integer array. These are array initialization and array properties.
Array Initialization:
In the PL array is initialized by writing the array name followed by the array size in brackets and equal sign and followed by list of items as followed by semicolon.
In case the size of array is bigger than the number of items in the list, then the list will be put to array by starting at index 0. All the remaining uninitalized values will be 0. Therefore if you want an array with no initializations, please write arr[100] = [];
Explanation:
<arr_name>[<size of array>] = [0,1,2,..];
<arr_name> is name of the array

Conventions:
- The size of the array specified in the left hand side must be greater or equal to the size of the array.
- If the size of the array specified in the left hand side is greater than the right handside, then the uninitialized items in the array will all be set to 0.
- This is the only way to initialize array and it does not support assigning array to another array.
- Variable types can not be more than one as it should be specified as 'int' or 'string'. Both of them at the same time cannot be used.

- Variables need to be initialized in the run time.

Integer Array Properties:
There are 3 properties that are predefined in the language.These are getting the value at index, seting the value at index and getting the length of the array.

- getting the value at specific index:
  Examples: (arr:12) will return the integer value that is stored in index 12. Parentheses are used to resolve ambiguities between integers in arithmetic expressions.
  (arr:n) will return the value at index 10.
- setting a value in specified index: This operation will assign the integer *value* to specified *index*.
  Example: arr::3 = 5; or arr::n+1 = 3;#"assuming that n is defined"
  This statement will put 5 to index location 3. In runtime if this number is greater than the array size it gets a runtime error.
- getting the length of the array: this operation will get the length of the array.
  Example: arr:: will return the length of the array.

Reason why there is no n dimensional arrays:
N dimensional arrays are not specified in the project prompt, it only stated data structure that store integer values. Therefore, because the targeted user of the programming language as CS students, it does not need to make matrix calculations extensively.
Even though it decreases writability to not have n dimensional arrays, the writability is traded with rapid development and memory efficiency.

2.6 Comment Statement

Comment statement is used to increase the readability of the code by allowing the developer to express his ideas while writing the code. The developer will be able to explain code according to his own desires. Commenting on the code will be available by putting the " # " word ' symbol, followed by a double quotation and closing quotation. It will not affect the main purpose of the code as it is not recognized by the compiler and is ignored. Mainly, when the end of the lines is reached, comments will not influence the rest of the code in the next lines. For example:
int c = a + b #" xxx this is used for the […………….] purposes "

Conventions:

- Developer need to be precise with his commenting as wrong commenting style can influence the code
- Comment usage between the code like a+#b will ruin the code as previously mentioned.
- between "#" and string there can't be any empty spaces. Otherwise it will invoke a syntax error.
- comments are single line. Which decreases writability but increases the readability by organization.

- You can put comment anywhere in the statement. Even in the middle of the statement. It will work properly.

2.7 Input/Output Statements
By having input/output statements we allow developers to interact with the program properly. Input statements are used for getting a value from the user and storing it in a variable. For example:
cin >> a ; here it takes a value from the terminal from the user and stores it in a variable called ‘a’; this variable keeps it and can be used later on.
Output statements are used to print necessary information from the user. For example:
cout << “ String “; is printing a given word in the quotation mark to the user.
cout << a ; also prints the value inside the variable ‘a’

Conventions:

- There is no concatenations. Therefore, either a variable or a string can be printed.This decreases writability but increases the development to make our lives easier.
- All the variables that are printed by cout or get from input by cin must be initialized.

3.0 Description of NonTrivial Tokens

Comments

<comment_text> - comments in this language are done by putting ‘ # ‘ symbol and they have no influence on the program.
Motivation - comments are important to give a sense of information about the piece of code a person writes.
Constraints - there is no specific constraint. Line should start with # to work as a comment.

Identifiers

<var> - this token is an identifier which includes function names, variable names, and other user defined names. Any combination of the letters or digits are accepted as an identifier. They only should not be starting by a digit.
Motivation - they are used for naming variables with good readability
Constraints - need to start with letter only

Literals

<integer> - integer represents and includes positive and negative whole numbers such as -5, 1, 0.
<string> - strings are written between double quotations (“ “) and represent a whole word.
Motivation - they are both used to show textual and numerical values.
Constraints - integers should not be initiated by putting any decimal point like 4.0, as it is a different type of literal. Strings are only recognized when they are between quotations marks (“ “)

Reserved Words

<if> - it is a conditional statement that takes an expression from the user and proceeds to complete inside the statement.
Motivation - to have strict rules to initiate some blocks of code
Constraints - there is not one. if should be used properly as with logical expressions
<else> - works cooperatively with the ‘if’ statement to give a conditional statement that if the previous condition is not satisfied, do this action.
Motivation - if the first rule does not pass, the second condition is helping with the handling condition.
<while> - this token is used for a loop statement that executes blocks of the code repetitively according to the condition given by the user.
Motivation - essential for repetitive tasks that are complex and need to be done through a few times of cycle. It helps to avoid the same blocks of code again and again.

Constraints - it can only be used for loop purposes.
<return> - it returns a specific value to the function when execution of the code is completed.
Motivation - when a developer wants to finish a program in one place, it is directly used to finish the function and return desired result to the caller.
Constraints - it can not be used outside of the function.
<main> - this token is used in the name of the function in the beginning. By using this word one time with ‘ function ‘, the user will initiate his written code and compile accordingly.
Motivation - easy to understand by any programmer of a different language. Enhances the readability.
Constraints - can only be initiated one time in the compile time of the program.
<cin> - used for inputting a string or integer statement by the user when asked in the compile time. This is used by developers during the coding stage.
Motivation - C based printing style to have easy, writable, and readable input statements.
Constraints - limited to only input statements.
<cout> - it is used to print statements to the terminal. Developers will be using this by entering ‘cout >> ‘ and accordingly write string, integer or variable value to print.
Motivation - can print any variable or literal we want. Additionally, the orthogonality is increased by the support of both printing strings and integer variables This increases readability along with writability.
Constraints - limited to the output statement. function calls are not specified yet. Cannot be concatanated.

3.0 Description of NonTerminal Tokens

program: Porgram is the whole code piece that consist of list of function declarations.
function_list: Function list is the list of function declarations that ends with main function where the program executed just like C.
function_declaration: Declared functions such as func foo(int[] arr, int x){...}
main_declaration: declaration of main function where the program starts.
param_list: parameter list of any function declaration. Example,in func main(int x, int[] a){...} declaration inside parentheses is where the param_list token belongs.
stmt_list: list of statements. Recursively runs the program. Every statement ends with semi colon and this token rule is what provides that.
stmt: Every type of statement reduces to stmt token. (return_statement, assignment statement, etc.)
input_stmt: Input statement with cin >> x
output_stmt: Output statement cout<<”hello world”;
return_stmt: return statement for functions that returns an integer.
function_calling: It is both a statement such as in foo(); or can be reduced to arithmetic expression token as in the assignment operation. x = foo();
var_list: variable list of function. foo(x,arr[]). Inside parentheses is where this token lies.
array_stmt: array statement only consist of seting the value of an specific index as arr::5 = 12;

while_stmt: while statement is for the loop statement.
if_stmt: it is for the conditional statements. there are two types that can be reduced. First is elseless if, and if with else if and elses. (else if is optional)
elif_stmt: consist of both else and else if statements. elif is the token name for else if and with recursive rule one can both write statements without elif tokens or with elif tokens. It always ends with else.
assign_stmt: There are two rules that can be reduced to that statement. First one is variable assignment. other is array initialization.
list_assignment: Deals with array initialization.
list: it is collection of list items surrounded by square bracket. SBO (Square Brackets Open) SBC (Square Brackets Close).
list_items: it is collection of integers separated by COMMA token (represent the char comma).
arithmetic_expr: tokens that are used in arithmetic operations such as multiplication division modulus expo. It is recursive so user can represent 1+2%3 .
logical_expr: token that deals with all logical expression recursively.
term: basic expressions that has a integer value inside. These are integer constant variable, function calling or array properties.
array_get: token than deals with getting the value in the specific index of the array.
array_length: token that deals with getting the length of the array.
type: representing whether the parameter is array or an integer in parameter list in function declaration.

4.0 Inferences about the project that we applied

We as designers of the program thought that printing only the first occurrence of the syntax error is better practice than printing all the errors. According to the P. Madore, our brain is evolved to do single tasking, and every time we switch from one task to another task there is a switch cost [1]. By regarding this article, we decided that one must need to focus on a single error at a time rather than focusing on all the errors. As a result, our programming language is only print single statement **intentionally**.

References

[1] Madore KP, Wagner AD. Multicosts of Multitasking. Cerebrum. 2019 Apr 1;2019:cer-04-19. PMID: 32206165; PMCID: PMC7075496.