



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> <return_stmt> <end_stmt> <closeBrace>
<main_declaration> ::= function
main<lp><rp><openBrace><stmt_list><return_stmt><end_stmt><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>
           | <function_calling_stmt>
           | <array_stmt>

<return_stmt> ::= return <arithmetic_expr>

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

<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> ::= if
<else> ::= else
<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> #arr[3] = [11,13,03]
<array_expr> ::= <var>.<array_properties>
<array_properties> ::= .assign<lp><integer>,<integer><rp>
    | .length<lp><rp>
    | .get<lp><integer><rp>
<list> ::= <sb_open> list_items <sb_close> # [1,2,4,7,8]
<list_items> ::= <list_items>,<int>| <int> # 1,2,3,4,5,7,9
<sb_open> ::= [
<sb_close> ::= ]
<array_term> ::= <var>.<length><lp><rp> | <var>.<get><lp><integer><rp>
<array_stmt> ::= <var>.<array_properties><end_stmt>

<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 ends with a return. This decreases the possibility of errors that are caused by no return statements errors.
- Every function can only call the functions above or itself.
- Every function returns int.
- 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:

```
<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));

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.

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.

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.

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 run-time errors or different results because of improper usage.

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 an '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 or etc. in our program. Variables need to be initialized before hand like this:

<var> count; Here we will give the type of the 'count' variable and use it later on.

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, index of every array's first item is 0. So the last element will be length-1. There are 2 important operations in integer array. These are array initialization, array properties.

Array Initialization:

<arr_name>[<size of array>] = [0,1,2,..];

Explanation:

<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:

- <array_name>.get(index): This will return the integer item that is stored in the specific index. index argument inside get is an int
- <array_name>.assign(index, value): This will assign the integer *value* to specified *index*. Both *index* and *value* is integer. .

2.6 Comment Statement

Comment statement is used for increasing the readability of the code by allowing the developer to express his ideas while writing the code. Developer will be able to explain code according to his own desires. Commenting on the code will be available by putting ' # " word ' symbol, followed by a single quotation and closing will quotation. It will not affect the main purpose of the code as it is not recognized by the compiler and ignored. Mainly, when the end of the lines is reached, comment 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.

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:

- Neither 'cin' or 'cout' statements accept function calls.
- There is no concatenations. Therefore, either a variable or a string can be printed.
- All the variables that are printed by cout or gett 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 concatenated.