CS F363 - COMPILER CONSTRUCTION

Group No: 29
Rohith Kumar Gattu - 2019A7PS0049H
Rohan Rao Nallani - 2019A7PS0048H
Kasina Satwik - 2019A7PS0011H
Srikar Sashank M - 2019A7PS0160H

Language specifications

The language “Kidney” is strongly typed and the primitive data types used are integers and floating points. The language supports arithmetic operations like addition, subtraction, multiplication, division, modulo operation, etc.

Data Types: Int, Float, String.

Key Words: Int, Float, If, Else, For, While, Return, String, Void, Array, main

Statements:

Declaration statement: The data type of the variable and the name should be specified while declaring the variable. The length of the variable name should be not more than 15 characters and should contain one alphabet followed by one numeric character.
Ex: Int abc123;

Assignment statement : The symbol ‘:’ is used as assignment operator to assign values to the variables
Ex: Int abc123 : 10;

Array Declaration: There are two types of declaration of array

1. Int arr[10]
2. Int _arr[100]

Return Statement: A return statement is the last statement in any function definition. A function not returning any value simply causes the flow of execution control to return to the calling function using the following statement based on the return type.
Ex:
Return;
Return 0;
Return ‘a’;
Return “abc”;

Conditional Statements:
IF :

```
If(x1<10){
        ?print(100);
}
```

IF-Else:

```
If(x1<10){
        ?print(100);
}
Else{
        ?print(200);
}
```

Iterative Statements: There are two types of iterative statements.

For Loop:

```
For(Int x1:0;x1<10;x1:x1+1){

}
```

WHILE LOOP:

```
While(x1<10){
        x1:x1+2;
        ?print(x12);
}
```

Function Declaration : The name of any function should start with '@'. The function ends with a '~' . Any function should have no more than 1000 variables to it.
Ex:

```
Int @sum(Int a, Int b){
        Return a + b;
}~
```

?read() for reading the user input from input stream.
?print() for writing or printing some data to output stream.

Comments:
There are two types of comments available

i)single line comments:
For commenting a single line use #
ii)multiline comments:
For commenting multiple lines start with a $ and end with a $

Operators:
Arithmetic Operators:

1. a1+b1
2. a1-b1
3. a1*b1
4. a1/b1
5. a1%b1

Logical Operators:

1. a1&&b1
2. a1||b1
3. !a1

Bitwise Operators:

1. a1&b1
2. a1|b1
3. a1^b1

Comparsion Operators:

1. a1<b1
2. a1<=b1
3. a1>b1
4. a1>=b1
5. a1==b1
6. a1!=b1

Assignment Operator:

1. a1:b1

STRUCTURE OF EXECUTION:

First input code is given in a file and the name of the that file is given as an input to the newlexer.cpp program.
Now a new code is generated in a new file named removedComments.txt
Now removedComments.txt is parsed and we get all the tokens from it as output.

Sample Code:

```
# program to take two positive integers from the user and print sum of all values in between #them, both inclusive.

Int @main(){
        Int a1;
        Int a2;
        ?read(a1);
        ?read(a2);
        Int sum1 : 0;
        For(Int i1:a1; i1<=a2; i1 = i1 + 1){
                sum1 : sum1 + i1;
        }
        ?print(sum1);
        Return 0;
}~

# sample code 2:
# kidney program to find the length of the given string.
Int @findlength(String str123){
   Int count123 : 0;
   Int ii1 : 0;
   While(str123[i1] >= "a" && str123[i1] <= "z"){
      count123 : count123 + 1;
      ii1 : ii1 + 1;
   }
   Return count123;
}~
Int @main(){
   String str123 : "Hello kidney";
   Int len123 : @findlength(str123);
   ?print(len123);
}~
```

OUTPUT for SampleCode2:

```
10000
TK_GLOBALFUNC
Global
Token 1, TK_Int, string Int, line number 2
Token 11000, TK_FUNCT, string findlength, line number 2
Token 69, TK_SMOPEN, string (, line number 2
Token 8, TK_String, string String, line number 2
Token 11001, TK_VARIABLE, string str123, line number 2
Token 70, TK_SMCLOSE, string ), line number 2
Token 67, TK_FLOPEN, string {, line number 2
Token 1, TK_Int, string Int, line number 3
Token 11002, TK_VARIABLE, string count123, line number 3
Token 66, TK_ASSIGN, string :, line number 3
Token 73, TK_NUM, string 0, line number 3
Token 401, TK_SEMICOLON, string ;, line number 3
Token 1, TK_Int, string Int, line number 4
Token 11003, TK_VARIABLE, string ii1, line number 4
Token 66, TK_ASSIGN, string :, line number 4
Token 73, TK_NUM, string 0, line number 4
Token 401, TK_SEMICOLON, string ;, line number 4
Token 6, TK_While, string While, line number 5
Token 69, TK_SMOPEN, string (, line number 5
Token 11001, TK_VARIABLE, string str123, line number 5
Token 71, TK_SQOPEN, string [, line number 5
Token 11003, TK_VARIABLE, string ii1, line number 5
Token 72, TK_SQCLOSE, string ], line number 5
Token 62, TK_GEQ, string >=, line number 5
Token 75, TK_STR, string "a", line number 5
Token 51, TK_LOGAND, string &&, line number 5
Token 11001, TK_VARIABLE, string str123, line number 5
Token 71, TK_SQOPEN, string [, line number 5
Token 11003, TK_VARIABLE, string ii1, line number 5
Token 72, TK_SQCLOSE, string ], line number 5
Token 64, TK_LEQ, string <=, line number 5
Token 75, TK_STR, string "z", line number 5
Token 11000, TK_FUNCT, string findlength, line number 13
Token 69, TK_SMOPEN, string (, line number 13
Token 10001, TK_VARIABLE, string str123, line number 13
Token 70, TK_SMCLOSE, string ), line number 13
Token 401, TK_SEMICOLON, string ;, line number 13
Token 82, TK_PRINT, string ?print, line number 14
Token 69, TK_SMOPEN, string (, line number 14
Token 10002, TK_VARIABLE, string len123, line number 14
Token 70, TK_SMCLOSE, string ), line number 14
Token 401, TK_SEMICOLON, string ;, line number 14
Token 68, TK_FLCLOSE, string }, line number 15
Token 76, TK_FUNCTEND, string ~, line number 15
```

LEXEMES	MEANING	TOKEN	TokenID
+	+(plus)	TK_ADD	54
-	-(subtraction)	TK_SUB	55
*	*(multiplication)	TK_MUL	56
/	/(division)	TK_DIV	57
%	%(modulo)	TK_MOD	58
&	&(bitwise and operator)	TK_LOGAND	51
&&	&&(logical and operator)	TK_AND	50
|	|(bitwise or operator)	TK_LOGOR	53
||	||(logical or operator)	TK_OR	52
<	<(less than)	TK_LET	63
>	>(greater than)	TK_GET	61
:	=(assignment operator)	TK_assign	66
;	;(semicolon)	TK_semicolon	401
<=	<=(less than or equal to)	TK_LEQ	64
>=	>=(greater than or equals operator)	TK_GEQ	62
==	==(equals operator)	TK_EQUALS	65
^	^(XOR operator)	TK_XOR	49
!	!(not operator)	TK_NOT	59
!=	!=	TK_NOTEQ	60
@@	EOF - End of File	TK_EOF	420
{	Left flower bracket	TK_FLOPEN	67

}	Right flower bracket	TK_FLCLOSE	68
(	Left small bracket	TK_SMOPEN	69
)	Right small bracket	TK_SMCLOSE	70
[	Left square bracket	TK_SQOPEN	71
]	Right square bracket	TK_SQCLOSE	72
[0-9][0-9]*	Integer values (Length is limited to 15 digits)	TK_NUM	73
[0-9][0-9]*.[0-9][0-9]*	Decimal values (Length is limited to 8 before the point and 10 after the point)	TK_DECIMAL	74
“*”	String can be anything in between “ “	TK_STR	75
Int	Integer can atmost have 10 digits	TK_Int	1
Float	Floating point numbers with real part atmost 8 digits and floating part atmost 10 digits	TK_Float	2
If	Conditional statement	TK_If	3
Else	Conditional statement	TK_Else	4
For	Iterative statement	TK_For	5
While	Iterative statement	TK_While	6
Return	For returning any datatype or even void	TK_Return	7
String	String keyword	TK_String	8
Void	keyword	TK_Void	9
Array	Keyword	TK_Array	10
main	Keyword,this must be there in a program	TK_main	11

@[a-z][a-z]*	Function starts with @ and only small alphabets	TK_FUNCT	Depends
~	Ending of the function	TK_FUNCTEND	76
?read	Reading from the input stream	TK_READ	81
?print	Writing to the output stream	TK_PRINT	82
[a-z][a-z]*[0-9][0-9]*	Variable starts with a-z and must have atleast two alphabets and atmost 15alphabets followed by atmost 15digits	TK_VARIABLE	Depends on scope
Error	Error token	TK_ERROR	999

GRAMMAR :

1. <program> = <other><main>
2. <main>=<data type>TK_main<body>
3. <other> =<function><other>|eps
4. <function>=<data type>TK_funcid TK_open<input>TK_closed<body>
5. <input>=<data type>Tk_varid|<data type>Tk_varid Tk_comma<input> | eps
6. <body>=TK_flowerOpen <stmts><return> Tk_flowerClosed
7. <return>=TK_return <literal> TK_semicolon
8. <literal>=Tk_num|Tk_alphabet|Tk_decimal
9. <stmts>=<stmt>|<stmt><stmts>
10. <stmt>=<declaration>|<if>|<if else>|<for>|<while>|<assignment>
11. <declaration>=<var_type>TK_var TK_semicolon
12. <var_type>=TK_int|TK_float|TK_string

13. <data type>=<var_type>
14. <if>=TK_if Tk_smallOpen<expr>TK_smallClose<body>
15. <if else>=<if>TK_else<body>
16. <for>=Tk_for Tk_smallOpen <spl_stmt> TK_smallClose<body>
17. <spl_stmt>=<stmt>TK_semicolon<stmt>TK_semicolon<stmt>TK_semicolon
18. <while>=Tk_while Tk_smallOpen<expr>TK_smallClose<body>
19. <expr>=<expr><operator><expr>|<basicExpression>|Tk_smallOpen<expr> TK_smallClose
20. <basicExpression>=Tk_varid|Tk_num|Tk_alphabets|Tk_decimal
21. <operator>=<arithmetic>|<logical>|<bitwise>|<comparison>
22. <comparison>=TK_lt|TK_le|TK_gt|TK_ge|TK_equals|TK_noteq
23. <arithmetic>=TK_add|TK_sub|TK_mul|TK_div|Tk_mod
24. <logical>=TK_and|TK_or|TK_not
25. <bitwise>=TK_bitwise_and|TK_bitwise_or|TK_xor
26. <assignment>=Tk_varid <assignmentOp> <expr> TK_semicolon
27. <assignmentOp>=TK_assign