**PROGRAMMING METHODOLOGY**

**ASSIGNMENT**

**GROUP NAME-RITCHY!**

**BY:**

**ANMOLA KUMARI**

**2016UCP1422**

**TARISHI JAIN**

**2016UCP1443**

**# LANGUAGE PROBLEM**

**THIS LANGUAGE USES BELOW MENTIONED OPERATORS TO FIND OUT *CEILING* (CLOSEST GREATER INTEGER) AND *FLOOR* (CLOSEST SMALLER INTEGER) OF EVALUATED EXPRESSION.**

**OUR LANGUAGE INCLUDES USER DEFINED FUNCTIONS AND IF-ELSE STATEMENTS.**

**IT CAN ALSO USE RETURN AND BREAK STATEMENTS AND CAN PERFORM LOOPING (WHILE LOOP).**

***OPERATORS USED ARE* :**

**+ ADDITION**

* **SUBTRACTION**

**/ DIVISION**

**# SQUARE ROOT**

**\*\* POWER**

**>> CEILING**

**<< FLOOR**

**= ASSIGNMENT**

**\* MULTIPLICATION**

**% MODULUS**

# BNF

<s>🡪Start main{<stmt\_list>}End| Start <fun\_decl>main{<stmt\_list>}<fun\_def>End

<fun\_decl> 🡪 <type> ID(<dec>);

<dec>🡪 <type>,<type>

<type>🡪Int|Void|Float

<stmt\_list><stmt>;<stmt\_list>|<if\_stmt><stmt\_list>|

<int\_decl> <stmt\_list>|<float\_decl> <stmt\_list>|<stmt>;

<fun\_def>🡪<type>ID(<arg>){<stmt\_list>}

<arg>🡪<type><id>|<type>ID,<type>ID

<stmt><id>=<expr>|<fun\_call>|<expr>|<print\_stmt>|<iteration\_stmt>|<break\_stmt>|<return\_stmt>

<fun\_call>🡪<id>(<id>,<id>)

<int\_decl>🡪Int <int\_id><int\_t>

<float\_decl>🡪Float <float\_id><float\_t>

<int\_t>🡪 ,<int\_id><int\_t>

<float\_t>🡪 ,<float\_id><float\_t>

<print\_stmt>🡪print(<id>,”<id><id>”,<id>,<id>)|print(<id>)|print(“<id><id>”,<id>)

<if\_stmt>→ if(<cond>){<stmt\_list>}|if(<cond>){<stmt\_list>}else{<stmt\_list}

<cond>→<id> is Int|<id><relop><expr>|<id>

<break\_stmt>→ break

<return\_stmt>→ return|return<expr>

<iteration\_stmt> → while(<cond>){<stmt\_list>}

<relop>→ >|<|<=|>=|==|!=

<expr>🡪<expr>+<term>|<expr>-<term>|<term>|<<<id>|>><id>|#<id>

<term>🡪<term>\*<factor>|<term>/<factor>|<term>%<factor>|<id>\*\*<factor>|<factor>

<factor>🡪(<expr>)|<integer>|<float\_value>|<id>

<integer>🡪<integer><digit>

<float\_value>🡪{digit}+"."{digit}\*|-{digit}+"."{digit}\*

<float\_id>🡪ID

<int\_id>🡪ID

<id>🡪ID

token definition

<digit>🡪{0|1|2|3|4|5|6|7|8|9}

**# BiSON**

* **What is a Parser?**

|  |
| --- |
| *Parsing* is the process of matching grammar symbols to elements in the input data, according to the rules of the grammar. The *parser* obtains a sequence of tokens from the lexical analyzer, and recognizes it's structure in the form of a parse tree. The *parse tree* expresses the hierarchical structure of the input data, and is a mapping of grammar symbols to data elements. Tree nodes represent symbols of the grammar (non-terminals or terminals), and tree edges represent derivation steps.  There are two basic parsing approaches: top-down and bottom-up. Intuitively, a *top-down* parser begins with the start symbol. By looking at the input string, it traces a leftmost derivation of the string. By the time it is done, a parse tree is generated top-down. While a *bottom-up* parser generates the parse tree bottom-up. Given the string to be parsed and the set of productions, it traces a rightmost derivation in reverse by starting with the input string and working backwards to the start symbol. |

* **How to use Bison?**

|  |
| --- |
| Bison is a general-purpose parser generator that converts a grammar description (Bison Grammar Files) for an LALR(1) context-free grammar into a C program to parse that grammar. The Bison parser is a bottom-up parser. It tries, by shifts and reductions, to reduce the entire input down to a single grouping whose symbol is the grammar's start-symbol.  http://alumni.cs.ucr.edu/~lgao/teaching/Img/bison.jpg |

**Format:**  
    %{  
    C Declarations  
    %}  
    Bison Declarations  
    %%  
    Grammar Rules  
    %%  
    Additional C Code

* Useful Bison definitions:  
       %token, %union, %type, %left, %right, %nonassoc, ...
* Format of the grammar rules section:  
      ***result: components ...  
          ;***
* Important data structure and functions:  
      yylval, YYSTYPE, yyerror(), yyparse()
* How to run flex in ubuntu:

flex file\_name.l (this will generate c code of flex code)

gcc lex.yy.c –lfl (to execute c code)

./a.out (object file of c code)

* How to run bison in ubuntu:

yacc –d file\_name.y (this will generate c code of bison code)

gcc y.tab.c –lfl (to execute c code)

./a.out (object file of c code)

* How to run flex and bison together:

flex file\_name.l (this will generate c code of flex code)

yacc –d file\_name.y (this will generate c code of bison code)

gcc lex.yy.c y.tab.c –lfl (to execute c codes)

./a.out (object file of c codes)