SLE language for dummies:

**Run the program si.exe with a filename as argument. The filename should have an extension of “sle” or “SLE” and SLE language written in it.**

## Structures and enumerators

|  |  |  |
| --- | --- | --- |
| enum bool | {TRUE=1,FALSE=0} | Simple Boolean enum |
| enum Type | {INT,FLOAT,ERROR} | Represents the sle language types, INT for int, FLOAT for real and ERROR for error purposes |
| struct number | Int ival float fval Type type | Represents a number in the sle language, containing both real value and int value, but the type field indicates whether it’s a real or an int. |
| struct node | char id[16] number num bool visited struct node\* next | A node of a linked list that we use to store the variables in the SLE language. The visited field represents whether an error message was already printed for this variable. |
| struct undefined | char id[16] struct undefined\* next | A node in a linked list which saves the id of variables that were not declared. |

## GLOBAL VARIABLES

|  |  |
| --- | --- |
| node \* head | A linked list to preserve the ids and values of all declared variables. |
| undefined \* errors | A linked list to preserve the ids of all undefined variables identified by the compiler. |

## FUNCTIONS

|  |  |
| --- | --- |
| void insertToSymbolTable(const char \* id,number num) | Inserts a new symbol with the given id and type similar to num.type, to the symbol table. The parameter num’s value is uninitialized. |
| void updateSymbol(const char \* id, number rhs); | Will update a symbol with the given id inside the symbol table to have the value equal to the value of rhs. If the types of the found symbol and rhs are different a compilation error will be invoked.  If id is not found, a compilation error will be invoked. |
| number GetValueFromSymbol(const char\* id); | Returns the value and type of a symbol found by the given id. If id is not found, a compilation error will be invoked. |
| number operatorMUL(number lhs, char opr,number rhs); | Returns the answer of multiplication/division/mod operation between two numbers. If both lhs’s and rhs’s types are int, the returned number’s type will be int, real otherwise. \* mod operation operates on two given numbers of type int, otherwise a compilation error is invoked.  \*If dividing 0, a compilation error will be invoked. |
| number operatorADD(number lhs, char opr,number rhs); | Returns the answer of addition/subtraction between  two numbers. If both lhs’s and rhs’s types are int, the returned number’s type will be int, real otherwise. |
| void printExpression(number num); | Responds to the put command of SLE. Will output the value of a symbol of any type to the screen. |
| node \* findById(node \* head ,const char\* id); | Find a symbol inside the symbol list, returns the actual node in the list if found, NULL otherwise. |
| undefined\* findError(const char\* id); | Find a symbol that invoked an error in the current compilation, returns the node if found, NULL otherwise. |

## FUNCTIONS

|  |  |
| --- | --- |
| lps.y | Input file for the bison parser generator. will contain the grammar rules for the sle language defining the tokens for the lexical analyzer and calling the right code when a token is parsed. Also in charge of error recovery. |
| Sle.l | Input file for the flex lexical analyzer. will contain the regular expressions for each token and will assign additional information to the token using the global variable yylval which the bison uses. |
| Lps.tab.h / lps.tab.c | The automatically generated files from bison. |
| Lex.yy.c | The automatically generated file from flex |
| main.c | Contains the functions implementations and the Actual program that runs when executing si.exe |
| <filename>.lst | Generated by si.exe, contains the parsed code of the sle language that was actually read by the parser. |

# Process Description

bison

lps.y

flex

gcc

sle.l

lps.tab.h

lps.tab.c

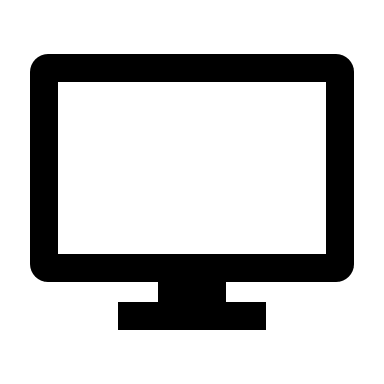
lex.yy.c

Main.c

si.exe

file.sle

file.lst



output