Compiler Construction / Compiler Design

Assignment Stage 1

Name and ID of Team Members:

Group 1:
Member 1: Pavan R (2012C6PS331P)
Member 2: Nitin Labhishetty (2012A7PS038P)

Group 2:
Member 3: Saurabh Satnalika (2012A7PS777P)
Member 4: Raghav Jajodia (2012A7PS064P)

(1)Language Features:

- Identifiers: Need to start with alphabet or underscore followed by zero or more alphanumeric or underscores.
- Delimiters: { } ( ) , ; # " "
- Literals:
  - Integer Literals- $-[0\text{-}9]^{+}|[0\text{-}9]^{+}$
  - Real Literals – $-[0\text{-}9]^{*}.[0\text{-}9]^{+}|[0\text{-}9]^{*}.[0\text{-}9]^{+}$
  - Boolean Literals – true| false
- Keywords:
  - Data Type Keywords- int, char, bool, complex, array, struct
  - Conditional Keywords- if, else
  - Looping Keywords- for
  - Jump Keywords- break, return
  - Function Keyword- function
  - Boolean Keywords- true, false
- Data Types:
  - Primitive Data Types- int, real, char, bool, complex
  - Complex Data Types- arrays, struct
- Operators:
  - Arithmetic Operators- +,-, *, /, ^(Exponentiation), %
  - Relational Operators- >, >=, <, <=, !=, ==
  - Bitwise Operators- and ,or, xor, ~, >>, <<
  - Logical Operators- &&, ||, !
  - Slicing Operator- :
  - Reference Operator- .
  - Assignment Operator- =
- Operation defined on Data Types:
  - int, real, char - Arithmetic, Relational, Bitwise
    Evaluation:
    - Operations related to int, real evaluated in C type way.
    - Operations related to char evaluated by using ASCII code of character
  - bool - Logical Operations
  - complex - Arithmetic Operations
  - arrays - Arithmetic Operations, Slicing operation.
    Syntax- arrays(Primitive_Type) ID[Arithmetic Expression];
    Evaluation:
    - In Arithmetic Operations, operation is carried on corresponding elements of arrays and result stored in another array without changing arrays.
    - Slicing Operation returns array of elements in the specified range. E.g. A[1:4] returns array of elements from index 1 to index 3(inclusive). Parameters of slicing can only be arithmetic expression.
    - Arrays of only one dimension can be declared.

- struct – Reference Operation

  Syntax- struct structName{

  #Primitive or array data type declaration

  }
  - Structures inside structures not allowed
  - Structures need to have at least one declaration.
  - Structures can be defined globally or locally in any any compound statement.

- Function:
  - Only Function definition required, no need of Function declaration.
  - Function names are identifiers having same rules as normal identifiers.
  - Nested functions not allowed.
  - Parameter list consists of Primitive and Complex Data Types. Primitive data types and struct are passed by value, arrays are passed by reference.
  - Function can return multiple values but return types need not be specified in function signature.
  - Function call can occur in assignment but cannot occur in expression.

  Function Definition Structure:

  ```
  function functionName(Parameter List){
      #Statements#
      return val1,val2,….,valn;
  }
  ```

- I/O operations: read(), write()
  - In write() specify string to be printed in double quotes. Use format specifier %v for printing variables. E.g. write(“Integer is: %v”,val).
  - In read() specify format specifier %v for variables being read followed by names of variables being read. E.g. read(“%v%v”,val1,val2).
- Scope Rules: Static Scoping with global variables allowed.
- Conditional statement:
  - If else block is specified with curly braces even when statement is one line.
  - Else If block is not allowed.
  - Expression can only be Arithmetic, Logical, Boolean and Bitwise expressions. Assignment statement is not supported.

  Structure:

  ```
  if(Expression){
      #Statements
  }
  else{
      if(Expression){
          #Statements
      }
      else{
          #Statements
      }
  }
  ```

- Assignment Statement:
  - Mixed Mode assignment with implicit conversion from int to char, char to int, int to real, char to real.
  - Explicit conversion can be forced upon by specifying type.
  - Function call can be used for assignment but not in expression. We have designed it this way because function may return multiple values.
  - Multiple assignments are allowed.

  E.g.

  ```
  int i;
  float f;
  f = 4.5;
  i = (float) f;
  i,f = 2,3;
  ```
- Declarations:
  - Primitive or complex data type variables can be declared as specified in above discussion.
  - Multiple declarations allowed. E.g. int i,j,k; is allowed.
  - Declarations and assignments cannot be done together. E.g. int i = 5; is not allowed.
- Iterative Statements:
  - While loop construct is supported.
  - Expression can only be Arithmetic, Logical, Boolean and Bitwise expressions. Assignment statement is not supported.

  Syntax:

  ```
  while(Expression){
      #Statements
  }
  ```
- Expressions:
  - Precedence of operators are as follows (Precedence decreases from top to bottom)

Operations	Example
Parenthesis	(a+2)
Type Casting	(int)2.5
Multiplication,Division	a*b, a/b
Addition,Subtraction	a +b, a-b
Shift Operators	a<<b, a>>b
Relational operators	a<b, a>b. a<=b, a>=b
Equality operators	a==b,a!=b
And operator (Bitwise)	a and b
Xor operator(Bitwise)	a and xor
Inclusive or (Bitwise)	a or b
Logical and	a && b
Logical or	a||b

(2) Lexical Units:

Pattern	Token	Purpose
[_a-zA-Z][0-9a-zA-Z_]*	<TK_ID>	Identifier
{	<TK_LEFTBR>	Delimiter
}	<TK_RIGHTBR>	Delimiter
(	<TK_LEFTPA>	Delimiter
)	<TK_RIGHTPA>	Delimiter
,	<TK_COMMA>	Delimiter
;	<TK_COLON>	Delimiter
“	<TK_QUOTES>	Delimiter
[	<TK_LEFTSQ>	Delimiter
]	<TK_RIGHTSQ>	Delimiter
$[0-9]^{+}$ | -$[0-9]^{+}$	<TK_INTLIT>	Integer Literal
[0-9]*.$[0-9]^{+}$ |-[0-9]*.$[0-9]^{+}$	<TK_REALLIT>	Real Literal
.	<TK_CHARLIT>	Character Literal
true	<TK_TRUE>	Boolean Literal
false	<TK_FALSE>	Boolean Literal
int	<TK_INT>	Keyword Integer
char	<TK_CHAR>	Keyword character
bool	<TK_BOOL>	Keyword boolean
complex	<TK_COMPLEX>	Keyword complex
array	<TK_ARRAY>	Keyword arrays
struct	<TK _STRUCT>	Keyword struct
if	<TK_IF>	Keyword if
else	<TK_ELSE>	Keyword else
while	<TK_WHILE>	Keyword while
break	<TK_BREAK>	Keyword break
return	<TK_RETURN>	Keyword return
function	<TK_FUNC>	Keyword function
main	<TK_MAIN>	Keyword main
/	<TK_DIV>	Arithmetic operators
*****	<TK_MUL>	Arithmetic operators
+	<TK_ADD>	Arithmetic operators
-	<TK_SUB>	Arithmetic operators
>=	<TK_GREQ>	Relational operators

>	<TK_GR>	Relational operators
<=	<TK_LSEQ>	Relational operators
<	<TK_LS>	Relational operators
!=	<TK_NE>	Relational operators
==	<TK_EQ>	Relational operators
and	<TK_BITAND>	Bitwise operators
or	<TK_BITOR>	Bitwise operators
xor	<TK_BITXOR>	Bitwise operators
<<	<TK_BITLSHIFT>	Bitwise operators
>>	<TK_BITRSHIFT>	Bitwise operators
&&	<TK_LOGAND>	Logical operators
||	<TK_LOGOR>	Logical operators
:	<TK_SLIC>	Slicing operator
.	<TK_REF>	Reference operator
=	<TK_ASSIGN>	Assignment Operator

(3) LL1 Grammar:

Start -> Main Externals .
Externals -> External_declaration Externals .
Externals -> e
External_declaration -> Function_definition .
External_declaration -> Declaration .
Main -> tk_main tk_leftpa tk_rightpa Compound_statement .
Function_definition -> tk_func tk_id tk_leftpa Arglist tk_rightpa Compound_statement .
Arglist -> Arg_declaration Arglist_extended .
Arglist -> e
Arglist_extended -> tk_comma Arg_declaration Arglist_extended .
Arglist_extended -> e
Arg_declaration -> Primitive_type_specifier tk_id .
Arg_declaration -> tk_array tk_ls Primitive_type_specifier tk_gr tk_id tk_leftsq Additive_expression tk_rightsq .
Declaration -> Primitive_type_specifier tk_id Extending_declarator tk_colon .
Declaration -> Array_declaration tk_colon .
Declaration -> tk_struct Struct_name Struct_left_factored .
Struct_left_factored -> tk_leftbr Struct_list tk_rightbr .
Struct_left_factored -> tk_id Extending_declarator tk_colon .
Extending_declarator -> tk_comma tk_id Extending_declarator .
Extending_declarator -> e
Primitive_type_specifier -> tk_char .
Primitive_type_specifier -> tk_int .
Primitive_type_specifier -> tk_real .
Primitive_type_specifier -> tk_bool .
Primitive_type_specifier -> tk_complex .
Array_declaration -> tk_array tk_ls Primitive_type_specifier tk_gr tk_id tk_leftsq Additive_expression tk_rightsq
Array_declarator .
Array_declarator -> tk_comma tk_id tk_leftsq Additive_expression tk_rightsq Array_declarator .
Array_declarator -> e
Struct_list -> Primitive_type_specifier tk_id Extending_declarator tk_colon Struct_list_declarator .

Struct_list -> Array_declaration tk_colon Struct_list_declarator .
Struct_list_declarator -> Struct_list .
Struct_list_declarator -> e
Struct_name -> tk_id .
Compound_statement -> tk_leftbr Declarator Many_statements tk_rightbr .
Declarator -> Declaration Declarator .
Declarator -> e
Many_statements -> Statement Many_statements .
Many_statements -> e
Statement -> Compound_statement .
Statement -> Selection_statement .
Statement -> Iteration_statement .
Statement -> Jump_statement .
Statement -> Assignment_statement .
Statement -> Function_call tk_colon .
Jump_statement -> tk_break tk_colon .
Jump_statement -> tk_return Expression Return_values tk_colon .
Assignment_statement -> Variable_list tk_assign Variable_list_left_factored tk_colon .
Variable_list_left_factored -> Expression Multi_assignment .
Variable_list_left_factored -> Function_call Multi_assignment .
Multi_assignment -> tk_comma Variable_list_left_factored .
Variable_list -> Lhs Variable_extender .
Variable_extender -> tk_comma Lhs Variable_extender .
Variable_extender -> e
Lhs -> tk_id Id_left_factored .
Id_left_factored -> tk_ref tk_id .
Id_left_factored -> tk_leftsq Locator tk_rightsq .
Id_left_factored -> e
Locator -> Additive_expression Additive_expression_left_factored .
Additive_expression_left_factored -> tk_slic Additive_expression .
Additive_expression_left_factored -> .
Function_call -> tk_func tk_id tk_leftpa Parameters tk_rightpa .
Parameters -> Expression Return_values .
Parameters -> e
Return_values -> tk_comma Expression Return_values .
Return_values -> e
Selection_statement -> tk_if tk_leftpa Expression tk_rightpa Compound_statement If_left_factored .
If_left_factored -> tk_else Compound_statement .
If_left_factored -> e
Iteration_statement -> tk_while tk_leftpa Expression tk_rightpa Compound_statement .
Expression -> Logical_or_expression e
Logical_or_expression -> Logical_and_expression Logical_or_left_recursion .
Logical_or_left_recursion -> tk_logor Logical_and_expression Logical_or_left_recursion .
Logical_or_left_recursion -> e
Logical_and_expression -> Inclusive_or_expression Logical_and_left_recursion .
Logical_and_left_recursion -> tk_logand Inclusive_or_expression Logical_and_left_recursion .
Logical_and_left_recursion -> e
Inclusive_or_expression -> Exclusive_or_expression Inclusive_or_left_recursion .
Inclusive_or_left_recursion -> tk_bitor Exclusive_or_expression Inclusive_or_left_recursion .
Inclusive_or_left_recursion -> e
Exclusive_or_expression -> And_expression Exclusive_or_left_recursion .
Exclusive_or_left_recursion -> tk_xor And_expression Exclusive_or_left_recursion .
Exclusive_or_left_recursion -> e
And_expression -> Relational_expression And_left_recursion .
And_left_recursion -> tk_bitand Relational_expression And_left_recursion .
And_left_recursion -> e
Relational_expression -> Shift_expression Relational_left_recursion .

Relational_left_recursion -> tk_eq Shift_expression Relational_left_recursion .
Relational_left_recursion -> tk_neq Shift_expression Relational_left_recursion .
Relational_left_recursion -> tk_ls Shift_expression Relational_left_recursion .
Relational_left_recursion -> tk_gr Shift_expression Relational_left_recursion .
Relational_left_recursion -> tk_lseq Shift_expression Relational_left_recursion .
Relational_left_recursion -> tk_greq Shift_expression Relational_left_recursion .
Relational_left_recursion -> e
Shift_expression -> Additive_expression Shift_left_recursive .
Shift_left_recursive -> tk_bitlshift Additive_expression Shift_left_recursive .
Shift_left_recursive -> tk_bitrshift Additive_expression Shift_left_recursive .
Shift_left_recursive -> e
Additive_expression -> Multiplicative_expression Additive_left_recursive .
Additive_left_recursive -> tk_add Multiplicative_expression Additive_left_recursive .
Additive_left_recursive -> tk_sub Multiplicative_expression Additive_left_recursive .
Additive_left_recursive -> e
Multiplicative_expression -> Cast_expression Multiplicative_left_recursive .
Multiplicative_left_recursive -> tk_mul Cast_expression Multiplicative_left_recursive .
Multiplicative_left_recursive -> tk_div Cast_expression Multiplicative_left_recursive .
Multiplicative_left_recursive -> e
Cast_expression -> tk_leftpa Root_expression_left_factored .
Cast_expression -> Literal .
Cast_expression -> Lhs .
Root_expression_left_factored -> Primitive_type_specifier tk_rightpa tk_leftpa Expression tk_rightpa .
Root_expression_left_factored -> Expression tk_rightpa .
Literal -> tk_charlit .
Literal -> tk_intlit .
Literal -> tk_reallit .
Literal -> tk_true .
Literal -> tk_false .
Literal -> tk_string_literal .
Multi_assignment -> e

(4) Test Cases:

TEST CASE 1:

```
main(){
        array(int) name[3],res[2];
        name={1,3};
        res[0]=name[0:1];
}
```

TEST CASE 2:

```
main(){

          struct name{
           int i,j;
          }
         name a,b;
         a.i=4;
         a.j=a.i+4;
         b.i,b.j=a.i,a.j;
}
```

TEST CASE 3:

```
main(){
        int n; n=10;
        if(n>7){
```

```
                write(“Big”);
                  }
              else{
                    if(n<2){
                              write(“Small”);
                              }
                    else{
                                write(“Middle”);
                }
            }
}
```

TEST CASE 4:

```
main(){
      int a,b;
      a,b=sqr(2,3);
}
function sqr(int a,int b){
       return a*a,b*b;
}
```

TEST CASE 5:

```
main(){
        int i; i=1;
        while(i<5){
                              if(i>8){
                                break;
                  }
          }
}
```

(5) Derivation

TEST CASE 1

#main(){ }

<start> ::= <main> <externals>

<externals> ::= e

<main> ::= <TK_MAIN><TK_LEFTPA><TK_RIGHTPA><compound-statement>

<compound-statement> ::= <TK_LEFTBR> <declarator><many-statements> <TK_RIGHTBR>

#array declaration. array(int) name[3],res[2];

<declarator>::= <declaration><declarator>

<declarator>::= e

<declaration> ::= <array-declaration> <TK_COLON>

<TK_ARRAY><TK_LEFTPA><primitive-type-specifier><TK_RIGHTPA> <TK_ID><TK_LEFTSQ><additive-expression><TK_RIGHTSQ><array-declarator>

<primitive-type-specifier> ::= <TK_LIT>

<id> ::= <TK_ID>

<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>

<additive-left-recursive> ::= e

<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>

<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal> ::= <TK_INTLIT>
<array-declarator> ::= <TK_COMMA><TK_ID><TK_LEFTPA><additive-expression><TK_RIGHTPA><array-declarator>
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal> ::= <TK_INTLIT>
<array-declarator ::= e

#array assignment. name = {1,3}
<many-statements> ::= <statement><many-statements>
<statement> ::= <assignment-statement>
<assignment-statement> ::= <variable-list>=<variable-list-left-factored>
<variable-list>::= <lhs><variable-extender>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> = e
<variable-extender> = e
<variable-list-left-factored> ::= <multi-expression>;
<multi-expression> ::= <expression><expression-extender>
<expression-extender> ::= e
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal> ::= <array-literal>
<array-literal>::= {<elements>}
<elements> ::= <num-elements>
<num-elements>::= <TK_INT> <int-element-extender>

<int-element-extender>::= <TK_COMMA><TK_INT><int-element-extender>
<int-element-extender> ::= e

#Slicing operation with assignment. res[0]=name[0:1];
<many-statements>::= <statement><many-statements>
<many-statements> ::= e
<statement> ::= <assignment-statement>
<assignment-statement> ::= <variable-list><TK_ASSIGN><variable-list-left-factored>
<variable-list>::= <lhs><variable-extender>
<lhs>:= <TK_ID><id-left-factored>
<variable-extender> = e
<id-left-factored> ::= <TK_LEFTSQ><locator><TK_RIGHTSQ>
<locator>::= <additive-expression><additive-expression-left-factored>
<additive-expression-left-factored> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal> ::= <TK_INTLIT>
<variable-list-left-factored> ::= <multi-expression><TK_COLON>
<multi-expression> ::= <expression><expression-extender>
<expression-extender>::= e
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<lhs> ::= <TK_ID><id-left-factored>
<id-left-factored>::=<TK_LEFTSQ><locator><TK_RIGHTSQ>
<locator>::=<additive-expression><additive-expression-left-factored>
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e

<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal> ::= <TK_INTLIT>
<additive-expression-left-factored> ::= <TK_SLIC><additive-expression>
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal> ::= <TK_INTLIT>

TEST CASE 2

<start> ::= <main> <externals>
<externals>::= e
<main>::= <TK_MAIN><TK_LEFTPA><TK_RIGHTPA><compound-statement>
<compound-statement> ::= <TK_LEFTBR> <declarator><many-statements> <TK_RIGHTBR>
<declarator>::= <declaration><declarator>
<declarator>::= <declaration><declarator>
<declarator>::= e

<many-statements>::= <statement><many-statements>
<many-statements>::= <statement><many-statements>
<many-statements>::= <statement><many-statements>
<many-statements>::= e
<declaration> ::= <struct-declaration>
<struct-declaration> ::= <TK_STRUCT> <struct-name> <TK_LEFTBR><struct-list><TK_RIGHTBR>
<struct-name> ::= <id>
<struct-list> ::= <declaration> <declaration-left-factored>
<declaration-left-factored> ::= e
<declaration> ::= <primitive-type-specifier> <id> <primitive-declarator><TK_COLON>
<primitive-declarator>::= <TK_COMMA> <id> <primitive-declarator>
<primitive-declarator>::= e
<primitive-type-specifier> ::= <TK_INT>

<declaration> ::= <struct-name> <id><struct-declarator><TK_COLON>
<struct-declarator>::= <TK_COMMA> <id><struct-declarator>
<struct-declarator>::= e
<struct-name> ::= <id>
<statement> ::= <assignment-statement>
<assignment-statement> ::= <variable-list><TK_ASSIGN><variable-list-left-factored>
<variable-list>::= <lhs><variable-extender>
<variable-extender>::= e
<lhs>:= <id><id-left-factored>
<id-left-factored> ::= <TK_REF><id>
<variable-list-left-factored> ::= <multi-expression><TK_COLON>
<multi-expression>::= <expression><expression-extender>
<expression-extender>::= e
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e

<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INT>

<statement> ::= <assignment-statement>
<assignment-statement> ::= <variable-list><TK_ASSIGN><variable-list-left-factored>
<variable-list>::= <lhs><variable-extender>
<variable-extender>::= e
<lhs>:= <id><id-left-factored>
<id-left-factored> ::= <TK_REF><id>
<variable-list-left-factored> ::= <multi-expression><TK_COLON>
<multi-expression>::= <expression><expression-extender>
<expression-extender>::= e
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= <TK_ADD><multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= <TK_REF><TK_ID>
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e

<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INTLIT>

<statement> ::= <assignment-statement>
<assignment-statement> ::= <variable-list><TK_ASSIGN><variable-list-left-factored>
<variable-list>::= <lhs><variable-extender>
<variable-extender>::= <TK_COMMA><lhs><variable-extender>
<variable-extender>::= e
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= <TK_REF><TK_ID>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= <TK_REF><TK_ID>
<variable-list-left-factored> ::= <multi-expression><TK_COLON>
<multi-expression>::= <expression><expression-extender>
<expression-extender>::= <TK_COMMA><expression><expression-extender>
<expression-extender>::= e
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= <TK_REF><TK_ID>
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e

<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= <TK_REF><TK_ID>

TEST CASE 3

#main() { }
start> ::= <main><externals>
<externals> ::= e
<main>::= <TK_MAIN><TK_LEFTPA><TK_RIGHTPA><compound-statement>

#int n;
<compound-statement> ::= <TK_LEFTBR><declarator><many-statements><TK_RIGHTBR>
<declarator>::= <declaration><declarator>
<declaration> ::= <primitive-type-specifier> <TK_ID><primitive-declarator><TK_COLON>
<primitive-type-specifier> ::= <TK_INT>
<primitive-declarator> ::= e
<declarator> ::= e

#n=10;
<many-statements>::= <statement><many-statements>
<statement>::= <assignment-statement>
<assignment-statement> ::= <variable-list><TK_ASSIGN><variable-list-left-factored>
<variable-list>::= <lhs><variable-extender>
<lhs>::= <TK_ID><id-left-factored>
<id-left-factored> ::= e
<variable-list-left-factored> ::= <multi-expression><TK_COLON>
<multi-expression>::= <expression><expression-extender>
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e

<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal> ::= <TK_INT>
<expression-extender>::= e

#if(n>7){ }
<many-statements> ::= <statement><many-statements>
<statement> ::= <selection-statement>
<selection-statement> ::= <TK_IF><TK_LEFTPA><expression><TK_RIGHTPA><compound-statement><if-left-factored>
<expression> ::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression><logical-or-left-recursion>
<logical-or-left-recursion>::= e
<logical-and-expression> ::= <inclusive-or-expression><logical-and-left-recursion>
<logical-and-left-recursion>::= e
<inclusive-or-expression>::= <exclusive-or-expression><inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression><exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression><and-left-recursion>
<and-left-recursion> ::= e
<equality-expression> ::= <relational-expression><equality-left-recursion>
<equality-left-recursion> ::= e

#n<5 Derivation
<relational-expression> ::= <shift-expression><relational-left-recursion>
<relational-left-recursion> ::= <TK_GR><relational-left-factored>
<relational-left-factored> ::= <shift-expression> <relational-left-recursion>
<shift-expression> ::= <additive-expression><shift-left-recursive>
<shift-left-recursive>::= e
<additive-expression> ::= <multiplicative-expression><additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression><multiplicative-left-recursive>
<multiplicative-left-recursive> ::= e
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INT>
<shift-expression> ::= <additive-expression><shift-left-recursive>
<shift-left-recursive>::= e
<additive-expression> ::= <multiplicative-expression><additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression><multiplicative-left-recursive>
<multiplicative-left-recursive> ::= e
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= e

#write("Big");
<compound-statement> ::= <TK_LEFTBR><declarator><many-statements><TK_RIGHTBR>
<declarator>::= e
<many-statements>::= <statement><many-statements>
<statement>::= <function-call><TK_COLON>
<many-statements> ::= e

<function-call>::= <TK_ID><TK_LEFTPA><parameters><TK_RIGHTPA>
<parameters>::= <expression>
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <string-literal>

#else{ }
<if-left-factored> ::= <TK_ELSE><compound-statement>
<compound-statement> ::= <TK_LEFTBR><declarator><many-statements><TK_RIGHTBR>
<declarator>::= e
<many-statements>::= <statement><many-statements>|e
<many-statements> ::= e
<statement>::= <selection-statement>

#else{ if(n<2){ } else{ } }
<selection-statement> ::= <TK_IF><TK_LEFTPA><expression><TK_RIGHTPA><compound-statement><if-left-factored>
<expression> ::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression><logical-or-left-recursion>
<logical-or-left-recursion>::= e
<logical-and-expression> ::= <inclusive-or-expression><logical-and-left-recursion>
<logical-and-left-recursion>::= e
<inclusive-or-expression>::= <exclusive-or-expression><inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression><exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression><and-left-recursion>
<and-left-recursion> ::= e
<equality-expression> ::= <relational-expression><equality-left-recursion>
<equality-left-recursion> ::= e

#n<5 Derivation
<relational-expression> ::= <shift-expression><relational-left-recursion>
<relational-left-recursion> ::= <TK_GR><relational-left-factored>
<relational-left-factored> ::= <shift-expression> <relational-left-recursion>
<shift-expression> ::= <additive-expression><shift-left-recursive>

<shift-left-recursive>::= e
<additive-expression> ::= <multiplicative-expression><additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression><multiplicative-left-recursive>
<multiplicative-left-recursive> ::= e
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INT>
<shift-expression> ::= <additive-expression><shift-left-recursive>
<shift-left-recursive>::= e
<additive-expression> ::= <multiplicative-expression><additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression><multiplicative-left-recursive>
<multiplicative-left-recursive> ::= e
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= e

#write("Small");
<compound-statement> ::= <TK_LEFTBR><declarator><many-statements><TK_RIGHTBR>
<declarator>::= e
<many-statements>::= <statement><many-statements>
<statement>::= <function-call><TK_COLON>
<many-statements> ::= e
<function-call>::= <TK_ID><TK_LEFTPA><parameters><TK_RIGHTPA>
<parameters>::= <expression>
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <string-literal>

#else{ write("Middle")}

<if-left-factored> ::= <TK_ELSE><compound-statement>
<compound-statement> ::= <TK_LEFTBR><declarator><many-statements><TK_RIGHTBR>
<declarator>::= e
<many-statements>::= <statement><many-statements>
<statement>::= <function-call><TK_COLON>
<function-call>::= <TK_ID><TK_LEFTPA><parameters><TK_RIGHTPA>
<parameters>::= <parameters>::= <expression><return-values>
<return-values>::= e
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal> ::= <string-literal>

Test Case 4
<start> ::= <main> <externals>
<externals> ::= <external-declaration><externals>
<externals>::= e
<external-declaration> ::= <function-definition>
<main>::= <TK_MAIN><TK_LEFTPA><TK_RIGHTPA><compound-statement>
<compound-statement> ::= <TK_LEFTBR><declarator><many-statements> <TK_RIGHTBR>
<declarator>::= <declaration><declarator>
<declarator>::= e
<many-statements>::= <statement><many-statements>
<many-statements>::= e

<declaration> ::= <primitive-type-specifier> <TK_ID> <primitive-declarator><TK_COLON>
<primitive-declarator>::= <TK_COMMA> <TK_ID> <primitive-declarator>
<primitive-declarator>::= e
<primitive-type-specifier> ::= <TK_INT>
<assignment-statement> ::= <variable-list><TK_ASSIGN><variable-list-left-factored>
<variable-list-left-factored> ::= <function-call><TK_COLON>
<variable-list>::= <lhs><variable-extender>
<variable-extender>::= <TK_COMMA><lhs><variable-extender>
<variable-extender>::= e
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= e
<lhs>:= <TK_ID><id-left-factored>

<id-left-factored> ::= e
<function-call>::= <TK_ID><TK_LEFTPA><parameters><TK_RIGHTPA>
<parameters>::=<expression><return-values>
<return-values> ::= <TK_COMMA><expression><return-values>
<return-values>::=e

<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INTLIT>

<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INTLIT>

<function-definition> ::= function <TK_ID>(<arglist>) <compound-statement>
<arglist> ::= <arg-declaration> <arglist-extended>
<arglist-extended> ::= <TK_COMMA> <arg-declaration> <arglist-extended>
<arglist-extended> ::= e
<arg-declaration> ::= <primitive-type-specifier> <TK_ID>
<primitive-type-specifier>::= <TK_INT>
<arg-declaration> ::= <primitive-type-specifier> <TK_ID>
<primitive-type-specifier>::= <TK_INT>
<compound-statement> ::= <TK_LEFTBR> <declarator><many-statements> <TK_RIGHTBR>
<declarator>::= e
<many-statements>::= <statement><many-statements>
<many-statements>::= e
<statement> ::= <jump-statement>
<jump-statement> ::= <TK_RETURN><expression><return-values>;
<return-values> ::= <TK_COMMA><expression><return-values>
<return-values> ::= e

<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <cast-expression><multiplicative-left-recursive>
<multiplicative-left-recursive> ::= <TK_MUL><cast-expression><multiplicative-left-recursive>
<multiplicative-left-recursive> ::= e
<cast-expression> ::= <root-expression>
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<root-expression> ::= <literal>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= e
<literal>::= <TK_INTLIT>

<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>

<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <cast-expression><multiplicative-left-recursive>
<multiplicative-left-recursive> ::= <TK_MUL><cast-expression><multiplicative-left-recursive>
<multiplicative-left-recursive> ::= e
<cast-expression> ::= <root-expression>
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<root-expression> ::= <literal>
<lhs>:= <TK_ID><id-left-factored>
<id-left-factored> ::= e
<literal>::= <TK_INTLIT>

Test Case 5
#main(){ }
<start> ::= <main> <externals>
<externals> ::= e
<main> ::= <TK_MAIN><TK_LEFTPA><TK_RIGHTPA><compound-statement>
<compound-statement> ::= <TK_LEFTBR> <declarator><many-statements> <TK_RIGHTBR>

#int i;
<declarator>::= <declaration><declarator>
<declarator>::=e
<declaration> ::= <primitive-type-specifier> <TK_ID> <primitive-declarator><TK_COLON>
<primitive-type-specifier>::= <TK_INT>
<primitive-declarator>::=e

#i=1;
<many-statements> ::= <statement><many-statements>
<statement>::=<assignment-statement>
<assignment-statement>::=<variable-list><TK_ASSIGN><variable-list-left-factored>
<variable-list> ::= <lhs><variable-extender>
<variable-extender>::=e
<lhs> ::= <TK_ID><id-left-factored>
<id-left-factored>::=e
<variable-list-left-factored> ::= <multi-expression><TK_COLON>
<multi-expression> ::= <expression><expression-extender>
<expression-extender>::=e
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>
<relational-left-recursion> ::= e
<shift-expression> ::= <additive-expression> <shift-left-recursive>

<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INT>

#while(Expression)
<many-statements> ::= <statement><many-statements>
<statement> ::= <iteration-statement>
<iteration-statement>::=<TK_WHILE><TK_LEFTPA><expression><TK_RIGHTPA><compound-statement>

#Expression. i<5
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>

<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<lhs>::=<TK_ID><id-left-factored>
<id-left-factored>::=e

<relational-left-recursion>::= <TK_LS> <relational-left-factored>
<relational-left-factored>::=<shift-expression> <relational-left-recursion>
<relational-left-recursion>::=e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INT>

#if
<compound-statement>::= <TK_LEFTBR> <declarator><many-statements> <TK_RIGHTBR>
<declarator>::=e
<many-statements>::=<statement><many-statements>
<many-statements>::=e
<statement>::=<selection-statement>
<selection-statement>::= <TK_IF><TK_LEFTPA><expression><TK_RIGHTPA><compound-statement><if-left-factored>
<if-left-factored>::=e
<expression>::= <logical-or-expression>
<logical-or-expression> ::= <logical-and-expression> <logical-or-left-recursion>
<logical-or-left-recursion> ::= e
<logical-and-expression> ::= <inclusive-or-expression> <logical-and-left-recursion>
<logical-and-left-recursion> ::= e
<inclusive-or-expression>::= <exclusive-or-expression> <inclusive-or-left-recursion>
<inclusive-or-left-recursion> ::= e
<exclusive-or-expression> ::= <and-expression> <exclusive-or-left-recursion>
<exclusive-or-left-recursion> ::= e
<and-expression> ::= <equality-expression> <and-left-recursion>
<additive-left-recursive> ::= e
<equality-expression> ::= <relational-expression> <equality-left-recursion>
<equality-left-recursion> ::= e
<relational-expression> ::= <shift-expression> <relational-left-recursion>

<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <lhs>
<lhs>::=<TK_ID><id-left-factored>
<id-left-factored>::=e

<relational-left-recursion>::= <TK_GS> <relational-left-factored>
<relational-left-factored>::=<shift-expression> <relational-left-recursion>
<relational-left-recursion>::=e
<shift-expression> ::= <additive-expression> <shift-left-recursive>
<shift-left-recursive> ::= e
<additive-expression> ::= <multiplicative-expression> <additive-left-recursive>
<additive-left-recursive> ::= e
<multiplicative-expression> ::= <power-expression> <multiplicative-left-recursive>
<power-expression> ::= <cast-expression><cast-expression-left-factored>
<multiplicative-left-recursive> ::= e
<cast-expression-left-factored> ::= e
<cast-expression> ::= <root-expression>
<root-expression> ::= <literal>
<literal>::= <TK_INT>

#compound of if; break
<compound-statement>::= <TK_LEFTBR> <declarator><many-statements> <TK_RIGHTBR>
<declarator>::=e
<many-statements> ::= <statement><many-statements>
<many-statements>::=e
<statement>::=<jump-statement>

<jump-statement>::=break<TK_COLON>