20171179

COMPILERS

Language Design and Definition Document

Programming Language - MiniC

This language is case-sensitive.

Keywords: **bool, int, uint, char, break, continue, if, else, false, true, for, while, cond, return, void, and, or, not, EOF, read, print, readfile**

White spaces can occur between any tokens.

Single line comment: starting the line with //

Multiple line comment: wrapping the comment within /* and */

(**x:** x in bold means it is a terminal token)

(<x>: this means x is a non-terminal token)

(x*: means zero or more occurrences of x separated by commas)

(x+: means one or more occurrences of x separated by commas)

(|: Alternatives)

Macro Syntax:

- <prog> : (variabl_decl ;)* function_decl* **void main** () <block>;
- <type> : **int** | **uint** | **bool** | **char** | **void** ;
- <variable_decl>: <type> <id>;

  | <type> <id>[(<**int_literal**> | <id> | <expr>)];

  | <type> <id>[(<**int_literal**> | <id> | <expr>), (<**int_literal**> | <id> | <expr>)];
- <function_decl>: <type> <id> (<variable_decl>*) <block> ;
- <block>: { <statement>* }

  | { <statement>* return; }

  | { <statement>* return expr ;}
- <statement> : <variable_decl>*

  | <expr>;

| <if_decl>

| <for_decl>

| <while_decl>

| <cond_decl>

| **read();**

| **print(**<expr>**);**

| readfile(<id>**);**

| **break**;

| **continue**;

- <if_decl>: **if** (<expr>) <block>;

  | **if** (<expr>) <block>; **else** <block>;
- <for_decl>: **for**(<expr1>; <expr2>;<expr3>) <block>;
- <while_decl>: **while**(<expr>) <block>;
- <cond_decl>: **cond**(expr1) expr2:expr3
- <expr>: <**int_literal**>

  | <**char_literal**>

  | <**bool_literal**>

  | <**EOF**>

  | <id>

  | <id> (<parameter_decl>*)

  | <expr> <bin_op> <expr>

  | <expr> ‘=’ <expr>

  | - <expr>

  | ! <expr>

  | (<expr>)
- <parameter_decl>: <id>

  | <id>, <parameter_decl>

  | <**int_literal**>

| <**int_literal**>, <parameter_decl>

| <**char_literal**>

| <**char_literal**>, <parameter_decl>

| <**bool_literal**>

| <**bool_literal**> , <parameter_decl>

- <bin_op>: <arith_op> | <rel_op> | <eq_op> | <bool_op>
- <arith_op>: + | - | * | / | %
- <rel_op>: < | > | <= | >=
- <eq_op>: == | !=
- <bool_op>: **and** | **or | not**

Micro Syntax:

- <int_literal>: <digit><digit>*
- <char_literal>: '<alpha>'
- <bool_literal>: **true** | **false**
- <EOF>: '-1' (represents end of file)
- <digit>: **0** |**1** | **2** | ... | **9**
- <id>: <alpha><alpha>*
- <alpha>: **a** | **b** | ... | **z** | **A** | ... | **Z**

Data Types

The basic data type of the language:

- **int (Signed Integer)**
- **uint (Unsigned Integer)**
- **bool**
- **char**

1D and 2D array can be created of any of the above data types. Arrays are indexed from 0 to N-1 where N > 0 is the size of the array. To index the array the bracket notation is used. When the **void** is used as one of the return types of a function then it signifies that the function does not have a return type.

Operations

All operations are binary operation. Following are the types of operations that are compatible:

- Arithmetic Operation: + | - | * | / | %
- Boolean Operation: and | or | not
- Comparison Operation: < | > | >= | <= | == | =!

For boolean operation **and**, and **or** if the result of the first operand determines the value of the whole expression then the second operand is not executed.

Statements

If

Firstly the expression is evaluated. If the expression comes out to be true then the if block is computed and if the expression comes out to be false then the else block is computed.

Eg:

```
If (x == 2)
{
        Statement 1;
}
else
{
        Statement 2;
}
```

For

There are 3 expressions in the initial sentence of the a for loop. The first expression defines the initial value of the variable, the second expression defines the termination condition, and the third expression defines the incremental change that should be applied on the variable for the loop to iterate. The loop block is executed if the termination condition comes out to be false. After each iteration, the termination condition is checked.

Eg:

```
for (expr1; expr2; expr3)
{
        Statements;
}
```

While

While loop has one expression which is computed each time before the execution of the block. If the expression comes out to be true then the block is executed else the while loop is terminated.

Eg:
```
while (expr1)
{
        Statements;
}
```

Cond? Op1: Op2

The condition operation has 3 expressions. Expression1 is computed, if it comes out to be true then expression2 is computed and returned else expression3 is computed and the corresponding value is returned.

Eg:
```
value = cond(expr1) expr2:expr3
```

Break

It contains inside the block of the for loop or the while loop. The execution of the loop stops at this statement and it continues with the statement after the block.

Eg:
```
for (expr1; expr2; expr3)
{
        Statement1;
        break;
        Statement2;
}
```

Continue

It contains inside the block of the for loop or the while loop. The execution for that iteration of the loop stops and it begins by starting the next iteration of the loop.

Eg:
```
for (expr1; expr2; expr3)
{
        Statement1;
        continue;
        Statement2;
}
```

Functions

- Function Declaration: Firstly the type of the return value of the function is defined. The return value can be of type: int, bool, char, or void. Then the identifier of the function and then the parameter declaration is done. In the case of the return, the type is void then the return statement doesn't return anything.

  Eg:
  ```
  <type> <id> ( <type> <id>*)
  {
          Statements;
          return <type>;
  }
  ```
- The function call is done by the id of the function following the expression which denotes the parameter of the function.

  Eg:
  ```
  <id> = <id> (<expr>*);          if a return value id expected

  <id> (<expr>*);                 if the return type of the function id void.
  ```
- The program should contain the main function whose return type is void. The execution of the program starts with the execution of the main function.

  Eg:
  ```
  void main()
  {
          Statements;
  }
  ```

Input / Output

- Input: Reads integer and char values at a time and stores it to the respective type of variable. Eg: <id> = read();
- Output: it prints the value of the expression given to it. Eg: print(<expr>);
- For reading a file use function readfile() with the file name as the input. Eg: readfile("abc.txt"). EOF represents the end of the file. The readfile commands read the data in the form of an array of char and accordingly stores that in the variable.

Semantic Rule

- No Identifiers are declared twice in the same space. No Identifiers are used before it is declared.
- Type check for each assignment, or expression.
- Type of the expression for the if condition operation should be bool. Similarly, the 2nd expression for the for loop should be bool and the expression for the while loop should also be bool.
- The number, type and order of parameters in a method call must be the same as the one declared at the time of method declaration.
- The program contains a definition for a method main that has zero parameters defined. The execution of the program starts with the execution of this function.
- The int_literal in an array declaration must be greater than zero.