Decaf Language Specifications

Sayantan Jana

September 2020

1 Introduction

MiniC is a simple programming language created as a part of Compilers course, which is almost identical to C (contains a subset of features allowed in C) and some modified features from C made for convenience of parsing and case handling . Below mentioned is the language specification, the syntax, rules and constructs .

2 Data Types

- int(signed integer)
- uint(unsigned integer)
- bool
- char
- 1D array of integers , array of boolean values.
- 2D array of integers , array of boolean values.
- 1D array of characters : string

3 Operators and Precedence

The types of operators are as follows :

- arithmetic operators : +, -, *, /, % , respectively for sum, difference, multiplication, division and modulus . Note that since the allowed data types do not include float as of now, here integer division is performed . BODMAS rule is followed here, i.e., first brackets are evaluated , followed by modulo, divide, muliplication, addsub (addition and subtraction) in order .

- relational operators : <= , >=, <, >, ==, ! =, namely for less than equal to, greater than equal to, less than, greater than, is equal to and is not equal to respectively .
- assignment operators : =, ++, −−, ++ does an addition of +1 to a variable and −− does a subtraction of -1 to a variable .
- logical operators : ||, &&, for concatenating 2 conditions .
- bitwise operators : |, &,ˆ namely bitwise or, and , xor operators .
- unary operator : ! , only one not operator supported .

4 Control Statements

if

```
if(<cond>)
{
    <stmt>
}
```

First, <cond> is evaluated. If the result is true, <stmt> is executed ,else nothing performed .

if-else

```
if(<cond>)
{
    <stmt1>
}
else
{
    <stmt2>
}
```

First, <cond> is evaluated. If the result is true, <stmt1> is executed ,else <stmt2> is executed, if it exists.

loop

MiniC supports two types of loop:

1. **while**
2. **for**

while

```
while(<cond>)
{
    <stmt>;
}
```

First $< cond >$ is evaluated if it is true then the statement within braces is executed, else nothing is caused and the control moves to next block to while . Once the statement is executed, once again <cond> is evaluated for truth and a repitition of the steps follow .

for

```
for(<assignment>;<cond>;<expr>)
{
    <stmt>;
}
```

First <assignment> is performed where the loop variable may be initialised, now <cond> is evaluated, if the result is true then the statement within braces is executed, else nothing is caused and the control moves to next block to for . Next <expr> is performed , once again <cond> is evaluated for truth and a repeatition of the steps follow .

5 I/O Routines

cin(var) Any input routine is written using the keyword 'cin' followed by the location within braces. At this location the variable being input is stored. cout(var) The output is written using the keyword 'cout' followed by the print statements. The print statement is quite similar to the one we use in python .

6 File Read/Write Routines

ifstream(filename)
fin(var)
close(filename)
Urges to read from the file named "filename" . Next input can be read using above way except fin replacing cin . close(filename) closes the file .

ofstream(filename)
fout(var)
close(filename)
Urges to print the output into the file named "filename" . Next output can be written using above way except fout replacing cout . close(filename) closes the

file .

7 Function Handling

```
function <DataType> function_name(Arguments)
{
    <Statements>
}
int main()
{
    cout(func function_name(Arguments));
}
```

A function is declared with the keyword ”function” at the start . It is invoked by the keyword ”func” . The arguments are passed through call by value mechanism. Recursion is supported . For returning anything the function uses ”return”keyword , incase a function simply has ”return;”, the function terminates at that point and returns nothing .

8 Semantic Checks

- There should be a main function from which the execution begins and the control goes on as the statements within main direct it . The return value of this function doesn’t matter .
- Variables and functions must be declared before they are used.
- The array size should be some positive finite integer not leading to excessive memory usage .
- Number of arguments in function declaration and number of parameters while calling the function must be same .
- If a function call is used as an expression , it must return something, that is it shouldn’t be of void type .
- A function should not be defined twice in the code .
- A function must return a value of the same datatype as it is declared as.

9 Micro-Syntax

<id> → [A-Za-z][A-Za-z0-9_]*
<intConstant> → [0-9]$^+$
<charConstant> → [Ascii{0-255}]

```
<boolConstant>    →   [True,False]
<stringConstant>    →   [Ascii{0-255}]*
```

10 Macro-Syntax

```
<Program>    →   <StartUp> int main() { <Statements> }

<StartUp>    →   ϵ
             |   <variable-def> <StartUp>
             |   <function-def> <StartUp>

<Statements>    →   ϵ | <Statement> <Statements>

<Statement>    →   <variable-def>;
               |   <AssignmentStatement>;
               |   <InputStatement>;
               |   <FileReadStatement>;
               |   <FileInputOpenStatement>;
               |   <FileOutputOpenStatement>;
               |   <FileCloseStatement>;
               |   <OutputStatement>;
               |   <FileWriteStatement>;
               |   <ForFragment>;
               |   <WhileFragment>;
               |   <IfFragment>;
               |   <IfElseFragment>;
               |   <TernaryStatement>;
               |   return;
               |   return <id>;
               |   return <expr>;
               |   return <FunctionCall>;

<expr>    →   ‘(’ <expr> ‘)’
          |   <function call>
          |   <expr> <arith-op> <expr>
          |   <expr> <rel-op> <expr>
          |   ‘-’ <expr>
          |   ‘!’ <expr>
          |   <constant>
          |   <id>

<arith-op>    →   ‘+’ | ‘-’ | ‘*’ | ‘/’

<rel-op>    →   ‘<’ | ‘>’ | ‘<=’ | ‘>=’ | ‘!=’ | ‘==’
```

```
<constant>    →    <intConstant>
              |    <charConstant>
              |    <boolConstant>
              |    <stringConstant>

<variable-def>    →    <DataType> <vars>;

<vars>    →    <id> | <id>,<vars> | <id>[size(uint)] | <id>[size(uint)][size(uint)]

<InputStatement>    →    cin(<id>);

<OutputStatement>    →    cout(<id>); | cout(<expr>);

<FileReadStatement>    →    fin(<id>)

<FileWriteStatement>    →    fout(<id>);

<FileInputOpenStatement>    →    ifstream(<id>);

<FileOutputOpenStatement>    →    ofstream(<id>);

<FileCloseStatement>    →    close(<id>);

<ForFragment>    →    for(<Statement>;<Condition>;<Statement>) {<Statements>};

<WhileFragment>    →    while(<Condition>) {<Statements>};

<IfFragment>    →    if(<Condition>) {<Statements>};

<IfElseFragment>    →    if(<Condition>) {<Statements>}; else {<Statements>};

<TernaryStatement>    →    (<Condition>) ? {<Statements>} : {<Statements>};

<AssignmentStatement>    →    <id> = <expr>;
                         |    <id>[uint val] = <expr>;
                         |    <id>[uint val][uint val] = <expr>;

<Condition>    →    True
               |    False
               |    !(<Condition>)
               |    <Condition> || <Condition>
               |    <Condition> && <Condition>
               |    <expr> <rel-op> <expr>

<function-def>    →    Function <DataType> <id>(<Arguments>) { <Statements>
```

}

<FunctionCall> → <id>(<id>*);

<DataType> → int | bool | char | uint

<Arguments> → ϵ
| <DataType> <id>, <Arguments>
| <DataType> <id>