LANGUAGE REFERENCE MANUAL

TEAM – 11

CS18B004 Akshitha
CS18B009 Deepanvi Penmetcha
CS18B013 G Nikhitha Vedi
CS18B031 S Bhavana

भारतीय प्रौद्योगिकी संस्थान तिरुपति
TIRUPATI

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI

TABLE OF CONTENTS

1 ABOUT THE SOURCE LANGUAGE

MinCode is a custom toy language that has some basic constructs which enable the user to code simple programs. The constructs of this language are elucidated below.

2 PROGRAM CONSTRUCTS

Constructs are basic building components of a program. They include different types of variables, literals, expressions, control statements like conditionals and loops, functions.

2.1 Identifiers

Identifiers are sequences of characters that are used for naming variables and functions. These names should include only English letters and digits. Any special characters should not be used. They should always start with a letter.

2.2 Keywords

Keywords are special identifiers that are inbuilt in the source language and cannot be used to name any variables or for any other purpose.

The list of keywords included in the *MinCode* are:

int init print get if else loop true false

2.3 Constants

Constants are numbers or sequence of characters which take the form of its assigned data type. *MinCode* allows Integer constants, bool constants (true/false only for conditions in control flow statements), String constants.

2.3.1 Integer Constants

An integer constant is a sequence of digits [0-9] that can be signed or unsigned.

2.3.2 Bool Constants

A bool constant is either true or false used only in conditional statements.

2.3.3 String Constants

A string constant is characters enclosed in double quotes. Strings should not contain double quotation marks(“ ”) which are exclusively used to encode the string. String constants are used only in print statements.

2.4 Expressions

MinCode uses standard operator associativity and precedence rules, same as in C language, to solve compound expressions that can have the same or different data types. These expressions can be arithmetic, logical, conditional or relational. Logical, conditional and relational occur only for conditions in the selection and iteration statements.

2.5 Operators

Operators specify the operations that are to be performed on the operands. *MinCode* employs Arithmetic operators (+, - (both binary and unary), /, *), Relational (>, <, ==, !=, <=, >=, =), Assignment (=), Logical (&&, ||, !).

2.5.1 Assignment Operators

The operator '=' is a standard assignment operator, which stores the value of its right operand in the left operand which is a variable.

2.5.2 Arithmetic Operators

MinCode offers operators for standard arithmetic operations: addition (+), subtraction (-), multiplication (*) and division (/).

2.5.3 Relational Operators

Relational operators are binary operators which compare two numeric operands to determine if one operand is greater than (>), less than (<), equal to (==), not equal to (! =), greater than or equal to (>=), less than or equal to (<=) another operand.

2.5.4 Logical Operators

Logical Operators give a boolean result of comparing the truth value of two similar operand types. They act as AND, OR, NOT operations.

2.6 Declarations

MinCode allows declarations without assignments. A declaration statement specifies the data type of the variable along with the variable. The type is restricted to int. A variable can only be declared once even if the block in which they are declared is different.

The syntax of a declaration statement :

type variable ;

2.7 Comments

Comments are anything that start and end with '#'.

2.8 Statements

Statements are used to take action and control the flow of the user's program. All statements in *MinCode* end with a semicolon.

2.8.1 Assignment

MinCode provides assignment operator '=' for assigning values to variables. This operator assigns values or values of variables on the right-hand side of the expression to the variables on the left-hand side.

The syntax of the assignment statement is :

variable = variable/constant/unary/binary expression;

2.8.2 Selection

MinCode provides 'if', 'if-else' statements to control the flow of the program based on the truth value of the expression confided in the if statement. The truth value decides whether the corresponding block should be executed or not.

The syntax of the **if** selection statement is :

```
if(condition)
{
        # this block is executed if the condition's truth value is true #
        statement(s);
}
```

The syntax of the **if-else** selection statement is :

```
if(condition)
{
        # this block is executed if the condition's truth value is true #
        statement(s);
}
```

```
        else
        {
            # this block is executed if the condition's truth value is false i.e. if
            the corresponding if block is not executed. #
            statement(s);
        }
```

2.8.3 Iteration

MinCode provides a 'loop' iteration, which executes the statements in the body of the loop block, repeatedly based on the truth value of the expression. The inner statements of the 'loop' keep executing till the expression's truth value is 1 i.e., expression is true. *MinCode* also supports nested iterations.

The syntax of the **loop** statement is :

```
    loop(condition)
    {
        # this block of statements is executed till the condition's truth value is
        true #
        statement(s);
    }
```

'if', 'if-else' and 'loop' can be nested and any number of levels are allowed.

2.8.4 I/O Statements

MinCode provides two keywords namely, print and get to write and read the data respectively to the monitor and from the keyboard.

The syntax of **print** keyword :

```
            print("Content to be displayed");  or  print(variable);
```

In the first format of the print statement, it takes a string literal(i.e., a string constant). This is the only place a string literal is allowed.

Using 'print', the user can display the content they want to see on the screen by entering the message within double quotes inside the small brackets that proceed after print. Users can also print the value of a single variable by mentioning the variable name inside the small brackets.

The syntax of **get** keyword :

get(variable);

Users can input the data from the keyboard using the ‘get’ keyword. This keyword reads the data from the keyboard and stores it in the variable mentioned in the small brackets. get can be used to store one variable at a time.

2.9 Block Structured/ Nested Blocks

MinCode uses blocks to differ the scopes of variables and thus delimits the regions where these variables and definitions apply. The beginning of the block is denoted by an open curly brace '{' and the end is denoted by a closing curly brace '}'.

init block

MinCode compiler executes the code which is primarily contained within the init block. Anything outside the reach of the init block will not be compiled. This is the block that lets the compiler know what code is to be executed.

Nested blocks

MinCode allows nested blocks i.e. it allows blocks inside blocks but init remains to be the primary block(the parent block).The structure looks like this:

```
init{
        statement(s);
        loop{
                if{
                        statement(s);
                                loop{
                                        statement(s);
                                }
                        if{
                                statement(s);
                        }
                      else{
                                statement(s);        } } } }
```

3 SAMPLE PROGRAMS

3.1 Program-1

```
init {                                        #beginning of ‘init’ block#

        # this code demonstrates the functioning of expressions, if-else and if statements and nested blocks#

        int a;                                #declaring a variable ‘a’ of type ‘int’#
        print(“Enter a”);                     #displaying a message using ‘print’#
        get(a);                               #inputting the value of variable ‘a’#

        int b; int c; int d;                  #declaring variables ‘b’,’c’,’d’ of type ‘int’#
        b = 2; c = 10; d = 4;                 #assigning values to b,c,d#
        a = b * (c + d) + a;                  #a complex mathematical expression#

        print("\n");                          #printing a new line#

        if(!((a==-30)||(a==30))){             #’if’ statement with conditions using logical operators
                                              checking if the value of a is -30 or 30 using relational
                                              operator(==)#
        print("abs(a) is not 1\n");           #printing the result if condition in if statement is true#
        }                                     #closing the ‘if’ block#
        else{                                 #if the condition in ‘if’ statement is not true then implement
                                              the statements in ‘else’ block#
                print("abs(a) is 1\n");       #printing the result if condition in ‘if’ statement is false#
        }                                     #closing the ‘else’ block#

        if(a!=0){                             #’if’ statement verifying if the value of a is not zero
                                              using relational operator(!=)#
                print("a is non zero\n");     #displaying the message#
        }                                     #closing the ‘if’ block#

                                              #closing init#

 }
```

3.2 Program-2

```
init {

        int i;
        int j;

        i = 0;

        # this program shows the function of loops and also nesting of various control flow statements #

        print("\n");

        loop(i<10){                                  #loop is run repeatedly until the condition given
                                                     is no longer true#
                j = 0;
                loop(j<10){                          #a nested loop which runs until the value of j is
                                                     lesser than 10#
                        if(i>j){
                                print("1 ");
                        }
                        else{
                                print("0 ");
                        }
                        j = j+1;
                }
                print("\n");
                i = i+1;
        }
        print("\n");


 }
```