COMS-W4115 Spring 2009

Programming Languages and Translators

Professor Stephen Edwards

Minimalistic BASIC Compiler (MBC)

Document Version 1.1

Project Proposal

Joel Christner (jec2160)

via CVN

February 10th, 2009

MBC Overview

Minimalistic BASIC Compiler (MBC) provides a simple means of compiling BASIC programs, and supports many of the commonly-used commands and features of the BASIC language. MBC will output C-compatible code which the user can then compile with a C compiler and execute.

Language Architecture

MBC will compile BASIC programs from source code into executable form. MBC will check each program that a user is attempting to compile and make sure the source code adheres to well-known BASIC language structure. This includes the following, which will be formally documented in a more structured manner in the language reference manual):

- Each line will be processed discretely; a new line character represents the end of a line, and the contents of the line past the sequence identifier are considered the instructions for that line
- Each line will begin with a number called a 'sequence identifier', and these numbers must be ascending from the beginning of the program. Sequence identifiers must be between 1 and 65535
- Whitespace is generally irrelevant, including whitespace that exists before or after statements that follow sequence identifiers. However, a space must exist between the first statement token and the line's sequence identifier
- Programs must be terminated through the use of the word 'END'. Each use of 'END' must be done on a line with a sequence identifier, and multiple instances of END may exist
- Statements can be concatenated on the same line through the use of a colon (:) separating the statements, for example:
    - 10 PRINT "Hello World" : PRINT "Another print command"
- Comments are supported through the use of the token REM, which indicates that all text from REM to the end of the line is not to be compiled. Some examples, each of which is valid:
    - 10 PRINT "Hello World": REM Hello World
      `Correct`
    - 10 REM Welcome to my program
      `Correct`
    - 20 PRINT A$ REM Hello
      `Correct`
- Basic mathematical functions are supported, including addition (+), subtraction (-), multiplication (*), and division (/) between discrete values and variables of the appropriate type
    - Order of precedence:

        - Multiplication and division take precedence over addition or subtraction
        - Functions are performed left to right according to precedence
        - Explicit ordering can be forced through the use of parenthesis
        - Example:
            - 3 * 3 + 3 – 4 * 4 + 5
              = (3 * 3) + 3 – (4 * 4) + 5
              = 9 + 3 – 16 + 5
              = 1
            - 3 * (3 + 3) – 4 * (4 + 5)
              = 3 * 6 – 4 * 9
              = 18 – 36
              = -18

- Comparisons can be used as part of expressions for both string variables and integer variables
    - Greater than (>)
    - Less than (<)
    - Equal to (=)
    - Greater than or equal to (>=)
    - Less than or equal to (<=)
    - Not equal (<>)
    - Integer variables support all of the above
    - String variables support only equal to or not equal
    - All comparisons are treated as BOOLEAN (true/false), therefore, are not confused as assignments
- MBC will support two variable types: strings and integers. String variable names must be followed immediately by a ($) character. Integer variable names must not be followed by the ($) character
    - Variables do not need to be declared, and variable names must not be the same as protected terms. Variable names are case sensitive
    - The item to the left of the equals sign will have its value transposed with the value of the item on the right of the equals sign
    - String variables, when assigned, must have data encapsulated in quotes, i.e.
        - A$ = “Hello World!”
          Correct
        - A$ = Hello World
          Fail!

    - Integer variables, when assigned, must not have data encapsulated in quotes, i.e.
        - A = “100”
          ```
          Fail!
          ```
        - A = 100
          ```
          Correct
          ```

- Protected terms are not case sensitive, and the following terms are considered protected terms:
    - FOR, TO, STEP, IF, THEN
    - GOTO, GOSUB, RETURN
    - PRINT, INPUT
    - REM
    - END
- Loops, algorithmic behavior, and program control are supported through the use of the following commands:
    - GOTO <seq_identifier> - causes the program to jump to the specific point in the program
    - GOSUB <seq_identifier> - causes the program to run the subroutine found at line noted by <seq_identifier>. The program will return to the same point once a RETURN statement is reached
    - FOR <variable> = <val_a> TO <val_b> STEP <amount> - causes the program to execute the statements between the FOR statement and the NEXT <variable> statement
        - Executes once for each time <val> is between or equal to <val_a> and <val_b>
        - STEP <amount> is an optional parameter, incrementing <val> by <amount> for each iteration. If STEP <amount> is not specified, a default increment of 1 is used, implying STEP 1
        - The <variable> in the NEXT statement must be the same as the <variable> in the FOR statement
        - Example:
            - 10 FOR I = 1 TO 20 STEP 5
            - 20 PRINT “Value is “,I
            - 30 NEXT I

              ```
              Value is 1
              Value is 6
              Value is 11
              Value is 16
              ```

- IF <condition> THEN <command1> ELSE <command2>
    - The <condition> is evaluated BOOLEAN for TRUE or FALSE. If the result is TRUE, then <command1> is executed. If the result if FALSE, then <command1> is not executed, but rather <command2> is executed.
    - ELSE <command2> is an optional parameter for IF…THEN statement blocks. If ELSE <command2> is not present and <condition> is FALSE then <command1> is simply ignored and program execution continues
    - Example:
        - 10 A=10
        - 20 IF (A=20) THEN GOTO 200
          `Program will not jump to 200 and will continue normally`
        - 10 A=10
        - 20 IF (A=10) THEN GOTO 100 ELSE GOTO 200
          `Program will jump to line 100`
        - 10 A=15
        - 20 IF (A=10) THEN GOTO 100 ELSE GOTO 200
          `Program will jump to line 200`
        - 10 A=10
        - 20 IF (A=10) THEN PRINT "Value of A: ", A
          `Value of A: 10`

- Text can be displayed on the screen using PRINT <data> as shown below, and can support literal string or integers as well as variables or a combination thereof. Any insertion of a variable after a string encapsulated in quotes <""> requires the use of a comma <,> after a closing quote <">. Any insertion of a variable after another variable requires a comma separate the two variables. Appending a string to another string, or a string to a variable, requires a comma before the opening quote.
    - PRINT – display some text on the screen, i.e.
        - 10 PRINT "Hello World!"
          `Hello World!`
        - 20 PRINT A$
          `<contents of A$>`
        - 10 PRINT "Console message: ", A$
          `Console message: <contents of A$>`
        - 10 PRINT A$, B$, "Hello World!", C$, " "
          `<contents of A$><contents of B$>Hello World!<contents of C$><space>`
        - 10 PRINT
          `<blank line>`

- INPUT - receive some input from the console
  - 20 INPUT “What is your name”, A$
    ```
    What is your name? <user could type response>
    ```
  - 20 INPUT “What is the value”, B
    ```
    What is the value? <user could type value here>
    ```

Examples

The following examples show programs that MBC can compile. There are far many other

Example 1 – basic use of variable assignment, PRINT, INPUT, IF, THEN, ELSE, GOTO, and END

```
10 REM Example
20 INPUT “Enter some text”, A$
30 INPUT “Enter some number greater than 0”, A
40 B=0
50 IF A<=0 GOTO 30
60 IF B<A THEN PRINT A$ ELSE GOTO 100
70 B = B + 1
80 GOTO 60
100 END
```

Example 2 – example 1 with GOSUB and RETURN

```
10 REM Example
20 INPUT “Enter some text”, A$
25 PRINT “You entered: “, A$
30 INPUT “Enter some number greater than 0”, A
35 PRINT “You entered: “,A
40 B=0
50 IF A<=0 GOTO 30
60 GOSUB 200
70 B = B + 1
80 IF B<A THEN GOTO 60
100 END
200 REM Subroutine
210 IF B<A THEN PRINT A$
220 RETURN
```

Example 3 – example 1 re-written with FOR statements

```
10 REM Example
20 INPUT “Enter some text”, A$
30 INPUT “Enter some number greater than 0”, A
```

```
35 IF A<=0 GOTO 30
40 FOR I = 0 TO A STEP 1
50 PRINT A$
60 NEXT I
100 END
```

Example 4 – example 1 re-written using IF, THEN, and ELSE

```
10 REM Example
20 INPUT "Enter some text", A$
30 INPUT "Enter some number greater than 0", A
35 IF A<=0 GOTO 30
40 B=0
50 IF B<A THEN PRINT A$ ELSE GOTO 100
60 B = B + 1
70 GOTO 50
100 END
```

Limitations Known at Design Time

The following limitations have been recognized at design time:

- This compiler only supports a minimalistic version of BASIC and does not by any stretch account for all of the capabilities provided by BASIC

Future Features

The following features are being considered for future releases of MBC, and should time permit, may be added to the v1.0 release for the final project deliverable for COMS-W4115.

- Support for DO…WHILE operations

Summary

MBC will provide a compiler for some of the core capabilities of the BASIC language. While executing MBC against BASIC source code, the user will be given feedback on the syntactical and lexical correctness of their BASIC program as well as areas that require correction. Should MBC complete without error, the BASIC program will be converted to C source code, which the user can then compile with their C compiler.