**Compiler HW3 BASIC Interpreter**

21000494 이성경

21200525 이상준

1. How to build and execute
2. Build source file

|  |
| --- |
| >make |

1. Run

|  |
| --- |
| >./basic <BASIC FILE NAME> |

1. Demo

<https://youtu.be/g4MFSp87t9s>

1. Source code

<https://github.com/seongkyeong/CPLHW3>

1. Added Language Feature

We are added language feature for BASIC Interpreter: while-loop syntax. The syntax use like below.

|  |
| --- |
| NUM WHILE Expression EXIT NUM |

And this is the example code & execution.

|  |
| --- |
| 5 LET A = 1  10 WHILE A < 10 EXIT 25  15 PRINT A  20 LET A = A + 1  25 PRINT "while end" |
| RUN  1  2  3  4  5  6  7  8  9  "while end" |

While-loop process block executes until the Expression have a false value. If the Expression value is false than go to line number EXIT.

1. Describe the design of AST node.
2. Define types for AST Node
   1. Define each command enum type

|  |
| --- |
| typedef enum { type\_rem, type\_goto, type\_let, type\_let\_dim, type\_dim, type\_print, type\_input, type\_if, type\_while } type; |

In order to classify the role of each command, set the type as above.

* 1. Define each operation enum type

|  |
| --- |
| Typedef enum { op\_uminus, op\_not, op\_plus, op\_minus, op\_multi, op\_divide, op\_mod, op\_equal, op\_left, op\_right, op\_left\_equal, op\_right\_equal, op\_left\_right } op\_type; |

While travalsing for AST node, we declared the type as above to handle exceptions and calculations for the corresponding operation type.

* 1. Define types for ast\_node

|  |
| --- |
| typedef enum { ast\_int, ast\_var, ast\_op } ast\_type; |

The types of types that AST nodes can have are Int, variable, and operation.

1. Define each structure
   1. AST Node Structure

|  |
| --- |
| typedef struct ast\_node{  ast\_type at;  op\_type ot;  var variable;  int value;  struct ast\_node \*left;  struct ast\_node \*right;  } ast\_node; |

This structure is AST node. AST node can have Int, variable, operation.

* 1. Variable structure

|  |
| --- |
| typedef struct{  char name[MAX\_VAR\_NAME];  int \*value;  int size;  struct ast\_node \*node;  } var; |

Variable can be 1-D array. So value is a pointer. Since run-time execution this value and size is set.

* 1. GOTO node structure

|  |
| --- |
| typedef struct{  int next;  } goto\_node; |

This node just has only next line number value.

* 1. PRINT node structure

|  |
| --- |
| typedef struct{  int value;  char str[MAX\_PRINT\_LENGTH];  ast\_node \*node;  } print\_node; |

Print structure has two syntax. One is just print a String. The other is print Expression.

* 1. LET node structure

|  |
| --- |
| typedef struct{  char name[MAX\_VAR\_NAME];  ast\_node \*node1;  ast\_node \*node2;  } let\_node; |

LET node is setting variable. If the variable is 1-D array, It can be represented by node2.

* 1. DIM node structure

|  |
| --- |
| typedef struct{  char name[MAX\_VAR\_NAME];  ast\_node \*node;  } dim\_node; |

DIM node initializes 1-D array.

* 1. INPUT node structure

|  |
| --- |
| typedef struct{  char name[MAX\_VAR\_NAME];  } input\_node; |

INPUT node is set when the user input the number.

* 1. IF node structure

|  |
| --- |
| typedef struct{  ast\_node \*node;  int next;  } if\_node; |

IF node has one Expression and if Expression is true, then go to next number line.

* 1. WHILE node structure

|  |
| --- |
| typedef struct{  ast\_node \*node;  int end;  } while\_node; |

WHILE node has one Expression and if Expression is false, then go to end number line.

* 1. Triple node structure

|  |
| --- |
| typedef struct{  char str[MAX\_CODE\_LENGTH];  type t;  int line;  goto\_node gn;  print\_node pn;  let\_node ln;  dim\_node dn;  input\_node in;  if\_node ifn;  while\_node wn;  } triple; |

Triple node structure has string of code, type of command, line number, and set the node for each type.

1. Run command

The RUN command executes the execute function for each line of code, executes the next line if the return value is INT\_MAX, and terminates execution if it is less than 0. If the return value is a positive number, the code with the line number is executed.

1. Execute function

The execute function distinguishes the type value from the passed triple node by switch-case statement and executes specific code for each case.

The REM and The GOTO type just return next line number.

The LET type is divided into the case of putting a value in an Array variable and the case of a value in a general variable. If it succeeds in assigning a value, it returns INT\_MAX else returns -1.

The DIM type initializes an Array type variable. If it succeeds in assigning a value, it returns INT\_MAX else returns -1.

The PRINT type prints the value if the string value is set, otherwise it prints the value for the expression. It returns INT\_MAX.

In the INPUT type, the user inputs an integer value and sets it to the specified variable.

The IF type returns the next line number if Expression is TRUE, or an INT\_MAX value if Expression is TRUE.

The WHILE type returns the next line number if Expression is TRUE, or an INT\_MAX value if Expression is TRUE.

1. Traval function

The Traval function takes an ast\_node value as an argument. Depending on the type of ast\_node, it is divided into 3 cases and the operation is done. If the type is ast\_int, return an int value. If it is ast\_var, it returns the updated value to the variable table. In the case of ast\_op, the left and right nodes are searched according to the operation type.