University of the Gambia

Compiler Design and Interpreter CPS-358

Mini C language (MCC)

Supervisor:

Prof Boilart
Lecturer

Author:

LAMIN JATTA
Mat 2134160

December 24, 2016

Abstract

The abstract of your report goes here.

Contents

1 Introduction

The `introduction` serves to give a description of the project, for the purpose of introducing readers to the content of the report. The Mini C Compiler(MCC) components that were used and modified and explains the analysis and synthesis design approach of cpmpiler.

We discuss how the design was transform to code in the implementation section of the report. As testing is an integral component of software design, we test the implementation of our MCC.

2 Design

MCC, design is seperated into two part `The analysis` and `The synthesis`.

The analysis part breaks up the source program into constituent pieces and imposes a grammatical structure on them. It then uses this structure to create an intermediate representation of the source program. The analysis part also collects information about the source program and stores it in a data structure called symbol table, which is passed along with the intermediate representation to the synthesis part.

The synthesis part constructs the desired target program from intermediate representation and the information in the symbol table This design comprises of structure, this structure are the components that were modified in each layer and a call between the components as shown in: Figure 1.

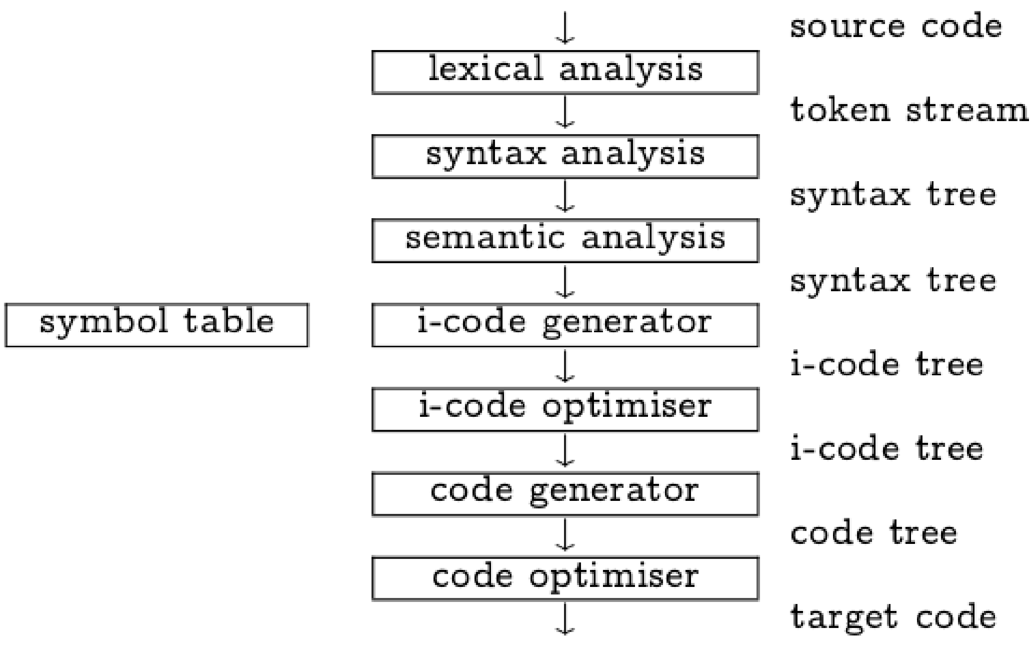

Figure 1: synthesis Components

Knowing the structure of the compiler the modifications i need to implement are:

- The Continue Statement
- The Break Statement
- The Switch Statement

3 Implementation

3.1 The break Statement

A break statement causes the Java interpreter to skip immediately to the end of a containing statement. We have already seen the break statement used to break loop or switch statement. The break statement is most often written as simply the keyword break followed by a semicolon:

break;

When used in this form, it causes the Java interpreter to immediately exit the innermost containing `while`, `do`, `for`, or `switch statement`. The `break` statement may only be used within the body of a loop. For example:

```java
/**
 * An example of using Break Statement in a loop.
*/
  for(int i = 0; i < data.length; i++) {  // Loop through the data array
      .
      if (data[i] == target) {                // When we find what we're
          looking for,
        index = i;                            // remember where we found
            it
        break;                                // and stop looking!
      }
    }   // The Java interpreter goes here after executing break
```

Listing 1: The Break Statement Class

On the other hand, the break statement in the switch can be optional for halting a case in a switch statement

```java
/**
 * An example of using Break Statement in a Switch Statement.
*/
     switch(expression){
    case value1:
     //code to be executed;
     break;  //optional
    case value2:
     //code to be executed;
     break;  //optional
    ......

    default:
     code to be executed if all cases are not matched;
    }
```

Listing 2: The Switch Statement Class

3.2 The continue Statement

While a break statement exits a loop, a continue statement quits the current iteration of a loop and starts the next one. continue, in both its unlabeled and labeled forms, can be used only within a `while`, `do`, or `for loop`. When used without a label, continue causes the innermost loop to start a new iteration. When used with a label that is the name of a containing loop, it causes the named loop to start a new iteration. For example:

```
/**
 * An example of using Continue Statement in a loop.
*/
  for(int i = 0; i < data.length; i++) {  // Loop through data.
      if (data[i] == -1)                     // If a data value is
          missing,
        continue;                            // skip to the next
            iteration.
      process(data[i]);                      // Process the data value.
    }
```

Listing 3: The Continue Statement Class

while, do, and for loops differ slightly in the way that continue starts a new iteration:

With a while loop, the Java interpreter simply returns to the top of the loop, tests the loop condition again, and, if it evaluates to true, executes the body of the loop again.

With a do loop, the interpreter jumps to the bottom of the loop, where it tests the loop condition to decide whether to perform another iteration of the loop.

With a for loop, the interpreter jumps to the top of the loop, where it first evaluates the increment expression and then evaluates the test expression to decide whether to loop again. As you can see, the behavior of a for loop with a continue statement is different from the behavior of the "basically equivalent" while loop I presented earlier; increment gets evaluated in the for loop, but not in the equivalent while loop.

3.3 The switch Statement

This code makes the repetitive structure of the problem more explicit, making the code easier to read, and hence maintain. Also, if the variable X should change its name, only one line would need changing compared to one line per case when if statements are used. The general form of the switch statement is

```
  switch (expression)
  case label
    command_list
  case label
    command_list


  otherwise
    command_list
endswitch
```

where label can be any expression. However, duplicate label values are not detected, and only the command_list corresponding to the first match will be executed. For the switch statement to be meaningful at least one case label command_list clause must be present, while the otherwise (default) command_list clause is optional.

If label is a cell array the corresponding command_list is executed if any of the elements of the cell array match expression. As an example, the following program will print 'Variable is either 6 or 7'.

```
 A = 7;
switch (A)
  case { 6, 7 }
    printf ("variable is either 6 or 7\n");
  otherwise
    printf ("variable is neither 6 nor 7\n");
endswitch
```

One advantage of using the switch statement compared to using if statements is that the labels can be strings. If an if statement is used it is not possible to write

```
  if (X == "a string") # This is NOT valid
```

since a character-to-character comparison between X and the string will be made instead of evaluating if the strings are equal. This special-case is handled by the switch statement, and it is possible to write programs that look like this

```
 switch (X)
  case "a string"
    do_something

endswitch
```

3.4 The Production and Structure

We first set the Production in the Parser class that define the rules of the Grammer for the Language. Since `Break` and `Continue` statement are `Statement` so i add them in the Production of the `ASTStatement`. `The Switch Statement` touches all the three nodes of the `ASTProgram` class which are the ASTDeclaration, ASTStatement and the ASTExpression and are all Statements.

A Break and Continue Statement does not passed any arguement and so it does not contains the return keyword and do not return anything.

On the otherhand the `Switch Statement` have a different Production rule that has a `Variables, an Expression` and `Case label` with each case containing Block of Statement and each BlockStatement can contain variable assingment.

```
switch (expression){
      case Label: statement;
break;
      case Label: statement;
break;

....

      defaule: statement;

   }
```

To achieved this we create AST class with the implementation of the visitor pattern for the class to be accessible to all related Node class and extending the super class ASTStatement:

1. The ASTBreakStatement
2. The ASTContinueStatement
3. The ASTSwitchStatement

The IdentityTranformation class is the algorithm that reconstructs (up to some indentations) the source code from the AST thus insuring that nothing got forgotten. And the `accept` method give the access algorithm to the visit Node.

Then it gives out the pretty print out of the source code. The Break and Continue are just indented, While the Switch will loop through the Array of Statement (Case Label) that i use a HashMap data structure with there key set and give indentations for clear reading.

From there it takes us to the Samantic part which comprises of three parts

1. Compute Address set the mapping of the assign variable to an address, value as an identifer. This mapping is done in the data structure known as the symbole Table. The Switch Statement is modify as followed:

```
public Object visitSwitchStatement(ASTSwitchStatement swStat,
    Object o) {
  start switchStatemet;

  accept the variable and asign an address;

  // process statements
  for(Entry<Integer, ASTBlockStatement> e : swStat.statements.
      entrySet())
  {
    accept each entryset and assign an address;
  }

  if(default exist)
    accept and assign an address;

  end switchStatement;

  return null;
}
```

For Break and Continue there is not much to do, we just access the break and exit the execution.

2. compute Reachable checks if there is any statement following a return statement that cannot be executed and thus consider it useless. To detect unreachable code, we need to test if a statement contains a return statement then mark that following statements as unreachable.

   The Break and Continue statements never contains a return statement thus check there reachable. In the Switch statement, it contains a return statement if its statement contains a return statement. So we check if it has a return statement by looping in the block statement and stop if it comes across return statement. Otherwise check if the default statement contain return statement then break and end the execution of the switch statement.

   using the visitor pattern i visit the SwitchStatement and check for reachable

```
public Object visitSwitchStatement(ASTSwitchStatement swStat,
    Object o) {
  boolean reachable = (Boolean) o;

  if (not reachable) {
    throw illegal messages;
  }

  this.insideSwitch = true;

  // a switch statement contains a return statement , if the
      statements have a return statement...
  boolean hasReturnS = false;

  for(Entry<Integer, ASTBlockStatement> e : swStat.statements.
      entrySet())
  {
    hasReturnS = (Boolean) get the value of the entryset and
        check if reachable ;
    if(it contain return ){
      break;// end for loop...
    }
  }

  // check default statement..
  boolean hasReturnD= false;

  if(it exist)
    hasReturnD = cast it boolean and check if it is reachable;

  return hasReturnS || hasReturnD;
}
```

3. Type Checking that checks for the declaration and expression, if variable declaration is encountered, the type information is written into the Entry of the variable. Otherwise if it is an expression the type of its operands computed and the results are compared.

   Break and Continue statements do not have an expression but are statements that are use without declaration so we just enter the Break and the Continue and exit. On the other hand the Switch statement contains three Declaration:

   - the variable (expression)
   - the defaule statement
   - the HashMap of statement

```
public Object visitSwitchStatement(ASTSwitchStatement swStat,
    Object o) {
  start the switch

    Type t = accept the type of the variable and cast it to Type;

    if (not the same time)) {
      throw  RuntimeException for the error position
    }

    // get the set of keys...
    // Don't use for loop index. since the values will start at
        index 0
    for(each key set at the variable statement){
      get the statement and accept it.
    }

    if(default is not null)
      check the statement and accept;

    m.d();
    m.verbose("<switch");

  end the switch
}
```

We first check the type of the VARIABLE that is declared accept and set it in the symbol table. then we loop through the HashMap Checking the key sets of the block statements compared the type to avoid type conflict.

The final touch is the `ByteCodeGenerator` is where the JVM is working with evaluation stack. Local variables are stored in stack registers. The stack registers are numbered 0 to number of variables minus 1. In this case you tell the source code where to start the execution and end the execution.

The Continue Statement stop and restart the execu whail jump back to the lStart.

```
  public Object visitContinueStatement(ASTContinueStatement conStat,
     Object o)
 {
 m.verbose("> continue");
 m.i();
 emt("\t\t ; continue");
 // jump to the begining of the while loop.
 if(this.continueLabe != null)
 {
   emt("goto "+ this.continueLabe);
 }

 emt("\t\t ; continue");

 m.d();
 m.verbose("< continue");

 return null;

 }
```

The Break Statement jumps to the end of the program where it is implemented and stop the execution. the keyword is goto that takes the action for it to lEnd.

```
 public Object visitBreakStatement(ASTBreakStatement breakStatement,
     Object o)
 {
   m.verbose("> break");
   m.i();
   emt("\t\t; break");

    if (this.breakLabel != null)
    {
   emt("goto " + this.breakLabel);
    }
   emt("\t\t; < break");
   m.d();
   m.verbose("< break");

    return null;
 }
```

In the Switch Statement if it stated execution it jumps to end of the label statement if it encountered a Break statement. So what i do is I compare the value assign to the variable and the Key set in the HashMap lsub them is the output is 0 then execute the statement and break then move to lEnd.

if non of the statement matches then the defaule statement should be executed.

```
  @Override
public Object visitSwitchStatement(ASTSwitchStatement swStat, Object o
   ) {
  start switch statement;



  String lEnd = label();// jump to end label , if switch contains a
     break..

  this.breakLabel = lEnd;

  emt("\t\t; switch");
  emt("\t\t; switch variable.");

  emt("\t\t; start switch statments");
  for(Integer key: swStat.statements.keySet()){
    String caseLabel = label();

    emt("ldc "+ key);// problem if this is block.
    swStat.var.accept(this, o);
    emt("isub");
    emt("ifne "+ caseLabel);
    //emt("goto "+ lEnd);
    swStat.statements.get(key).accept(this, o);
    emt("goto "+ lEnd);
    eml(caseLabel);
  }

  emt("\t\t ; end switch statements");

  if(swStat.defaultStatement != null){
    emt("\t\t; deafult statement");
    swStat.defaultStatement.accept(this, o);
  }

  eml(lEnd);
  emt("\t\t; end switch");

  m.d();
  end switch statement;

  // reset label.
  this.breakLabel = null;

  return null;
}
```

4 Conclusion

MCC is a very simple yet powerful Compiler that uses the same analysis and synthesis of any compiler. It however doesn't implement all GRAMMER standard which consequently makes it a rather restricted environment to work with. The aim of this project was to realize the implementation of how Compiler are working and know the stages it takes to execute program source code. To this end, various modifications were made to the different MCC layers, from the analysis down to the synthesis layer. But the advantages of such designs is improved scalability thus new features can be added without making much modifications.

From the design of MCC project implementation and to finally carry out some sample test cases, we manage to present in this report the very basic essential of how MCCompiler execute java source code.

A Future Work

This MCC has many feature that can be done for improvement and in such as a student we can learn lots from it and know how the the compiler works and interpret source codes. This MCC does not support some features such as:

- for loop.
- Array data structure
- Strings
- Float data type
- and so on

These are some features to name a few that can be added in the MCC projects.

B Contribution

Thanks to Ebrima Jallow who sometimes guide me in correcting my errors such as null pointer exception, stackmap errors etc. that were so confusing a bit stresfull sometimes.

Thanks to Prof Boilart who shows me the Production rules and seperated the in and out of the implementation. Continue to help if i have any problem that i come across in my work through.

C Classes

This section will list the classes that i use for implementation and thus the file i modify

Class Name	Description
Parser.jj	where i set the Production rules for the Grammer or Language
ASTVisitor.java	Help to give access for other classes to reach accessible without moch work to do
ASTSwitchStatement.java	where the visitor class will access and to give other classes access the class
ASTBreakStatement.java	Briefly state the purpose of this class
ASTContinueStatement.java	And this class as well similar to Break statement
IdentityTranformation.java	Is where the pretty printing of the source code is modify
ComputeAddress.java	where the mapping of identifer to types, value and variable to the symbol table
ComputeReachableCode.java	checks if the statement contain a return statement then stop execution
TypeChecker.java	check the data type defination and compare to all statement
TFByteCodeGenerator.java	Where local variable are stores stack rigister

Table 1: List of Added classes

D Reffrences

- http://www.javatpoint.com/java-tutorial
- https://en.wikipedia.org/wiki/Control_flow#Goto
- https://www.tutorialspoint.com/java/switch_statement_in_java.htm
- http://docstore.mik.ua/orelly/java-ent/jnut/ch02_06.htm