MINI Compiler – Final Report

Mark Randles
Department of Computer Science
CS409 - Language Design & Implementation
`randlem@bgnet.bgsu.edu`

22nd November 2004

1 Environment

The environment that was used to produce this compiler was GCC/G++ version 3.3.4 from the GNU organization. The environment that this compiler was written was Slackware 10. The compiler also appears to compile under the BGUNIX environment, but the executable produced by the outdated version of GCC/G++ on BGUNIX appears to enter an infinite loop while compiling a MINI program. The cause of this is unknown.

2 Extra Classes

There were no extra classes defined for this compiler. However, there were extra function calls added, and different functionality added to various support classes. The main classes remain quite virgin.

3 Problems and Issues

This compiler is incomplete. It appears to implement about 85-95language, implementing everything but forward declarations of procedures. For most other syntaxes it appears to produce correct object code. The testing of this compiler was such that each individual code generator was tested extensively, and all apparently passed this authors battery of tests.

There is one reproducible segmentation fault with this compiler currently. It would appear that a NULL pointer can be introduced into either Parser::currentOp or Parser::nextOp. When this pointer is used it causes a complete failure of the program, and it's termination. The most common point of failure is when the getKey() method of the NULL pointer is being called. The author believes the solution to this problem is in the scanner portion of the project. During the call to Scanner::nextToken() the scanner needs to see if the token is actually in the token table, and if not then throw an error.

Also, there seems to be a problem with the control flow of the standard test program sample.mini, provided by the professor. When compiled without error on this compiler, it seems to run erratically in the testing environment. The output of the program seems to be off of what should appear. However, the author was unable to find a quick-fix without intensely studying the code produced, which was impossible at the late hour this issue was discovered.

4 Extensions

There was one extension, or what this author considers one. The functionality of the operator ”break” was extended to act like a C/C++ return statement when called from the scope of a proc structure or the main body of the program. Here are some examples: This code will produce a program flow where the main program calls myproc, which will print a zero and return immediately, which will immediately call the code associated with the second break to terminate the program.

Program 1 Listing of program test_break_1.mini

```
program testbreak;
 proc myproc;
  write 0;
  break;
  write 1;
 endproc;

 call myproc;
 break;
 call myproc;
endprogram;
```

5 Compiler Interface

No changes were made.

6 Issues with SIPS

While using the SIPS runtime environment a few issues presented themselves. First and for most was the apparent lack of good cross-platform coding practices exercised when laying out the display. Many of the data display boxes are not dynamically resized to fit whatever font is used in them, and because of the font differences between Windows and Linux, the displayed text will often run over the sides of the boxes.

Also, one of the more nagging features of the SIPS environment is the file selection dialog will not save it's previous folder so every time a user wishes to use another object file, they must re-navigate the file system to find their file. I'm unsure if this problem extends to Windows, but it does exist on Linux.

The third and last issue that came up was the amount of helpful information that SIPS produces on a loading error, which is exactly none. When it tries to load a invalid opcode it displays a message saying so, but it does not make it clear at what position in the object file it occurs or what the offending opcode is.

These were issues that I found while using the SIPS environment. They are for informative purposes only.

7 Working Code Samples

All of these are available in the directory tests in the mini folder.

Most example should compile normally displaying the line ”Successful compilation!” when finished. Examples with ”err” in their filename should error during compilation, however during the compiler testing process these may have been changed and there is no guarantee of their intended action.

Program 2 Listing of program sample.mini

```
/*
   This program demonstrates many features of
   the language.

   Enter these input values:  5  7  10  -1
*/
program sample;

int largest = 0, i, j,
    listLength, list[50] = (’x’, ’X’, 20, 020, 0x20),
value;

proc GetList;
    int limit = 50, negOne, NegOne,
        value;
    0xf423E - 03641077 -> negOne;  /* 999998 - 999999 */
    -1 -> NegOne;
    read value;
    while (negOne < value)
        if (limit <= listLength)
            break;
        endif;
        value -> list[listLength];
        call AddOne;
        read value;
    endwhile;
    NegOne + 1 -> NegOne;
    write negOne, NegOne;
endproc;

proc AddOne;
    listLength - NegOne -> listLength;
endproc;

/* Beginning of main program */

1234 -> value;
5 -> listLength;
call GetList;
write value;
0 -> i;
while (i < listLength)
    if (largest < list[i])
        list[i] -> largest + list[1] -> value;
        if (i < 4)
            i+1->j;
            value + list[j] -> value;
        endif;
    endif;
    i + 1 -> i;
endwhile;
write largest, value, list[1];

endprogram
```

Program 3 Listing of program test_arith.mini

```
program testarith;
int one=1,two=2,three=3,result=0;

/* test + */
one + two -> result;
write result;

/* test - */
two - one -> result;
write result;

/* test * */
two * three -> result;
write result;

/* test / */
three / two -> result;
write result;

/* test stringing */
two * three -> result;
write result;
result + one + two -> result;
write result;
result / two + one -> result;
write result;


/* test literal */
1 + 2 + 3 + 4 -> result;
write result;
4 / 2 + 4 -> result;
write result;

endprogram;
```

Program 4 Listing of program test_ary.mini

```
program testary;
int ary[101];

endprogram;
```

Program 5 Listing of program test_ary_ld.mini

```
program aryld;
int ary[2] = (2,3),one=1,result=0;

/* test a single ld op on the array */
write ary[0];
write ary[one];

/* test in an expression */
ary[0] * 2 -> result;
write result;

ary[1] * ary[0] -> result;
write result;

/* read reading into an array */
read ary[0];
ary[0] * 2 -> result;
write result;


endprogram;
```

Program 6 Listing of program test_break_1.mini

```
program testbreak;
proc myproc;
write 0;
break;
endproc;

call myproc;
break;
call myproc;
endprogram;
```

Program 7 Listing of program test_con.mini

```
program testcon;
int one=1,two=2,three=3;

if(one < two)
write one;
endif;

if(two < three)
write two;
else
write three;
endif;

if(one = one)
write one;
endif;

if(one <= one)
write one;
endif;

while(one <= three)
write one;
one + 1 -> one;
endwhile;

endprogram;
```

Program 8 Listing of program test_err_prog0.mini

```
program 5;

endprogram;
```

Program 9 Listing of program test_err_prog1.mini

```
program test

endprogram;
```

Program 10 Listing of program test_err_write.mini

```
program testerrwrite;
int one=1,two=2,three=3,ary[10]=(1,2,3,4,5,6,7,8,9,10);

write one,two,three,ary[5];
endprogram;
```

Program 11 Listing of program test_int.mini

```
program testint;
int test,asdf;
endprogram;
```

Program 12 Listing of program test_int_ary.mini

```
program testintary;
int largest = 0, i, j, listLenght,
list[50] = ( ’x’, ’X’, 20, 020, 0x20), value;
endprogram;
```

Program 13 Listing of program test_proc.mini

```
program testproc;
int one=1,two=2;

proc myproc;
int three=3;
write one,three;
endproc;

break;
one + two -> one;
call myproc;

endprogram;
```

Program 14 Listing of program test_prog.mini

```
program test;

endprogram;
```