CSC 460 Language Translation
Group program 4
Code Generator

Modify the Micro Compiler to generate C code as an intermediate code by adding the action routines: #start, #assign, #read_id, #write_expr, #gen_infix, #process_literal, #process_op, #process_id, #finish, etc. The output file, .C extension, will contain the C code generated by your compiler. The listing file, .lis extension, will remain the same as a properly working program 3. The .C file should compile and run using DevC++.

1. <program> –> **#start** BEGIN <statement list> END
2. <statement list> –> <statement> {<statement list>}
3. <statement> –> <ident> := <expression> **#assign**;
4. <statement> –> READ (<id list>);
5. <statement> –> WRITE (<expr list>);
6. <statement> –> IF (<condition>) THEN <StatementList> <IFTail>
7. <IFTail> –> ELSE <StatementList> ENDIF
8. <IFTail> –> ENDIF
9. <statement> –> WHILE (<condition>) {<StatementList>} ENDWHILE
10. <id list> –> <ident> **#read_id** {, <ident> **#read_id** }
11. <expr list> –> <expression> **#write_expr** {, <expr list> **#write_expr** }
12. <expression> –> <term> {<add op> <term> **#gen_infix**}
13. <term> –> <factor> {<mult op> <factor> **#gen_infix**}
14. <factor> –> (<expression>)
15. <factor> –> - <factor>
16. <factor> –> <ident>
17. <factor> –> INTLITERAL **#process_literal**
18. <add op> –> + **#process_op**
19. <add op> –> - **#process_op**
20. <mult op> –> * **#process_op**
21. <mult op> –> / **#process_op**
22. <condition> –> <addition> {<rel op> <addition> **#gen_infix**}
23. <addition> –> <multiplication> {<add op> <multiplication> **#gen_infix**}
24. <multiplication> –> <unary> { <mult op> <unary> **#gen_infix**}
25. <unary> –> ! <unary>
26. <unary> –> - <unary>
27. <unary> –> <lprimary>
28. <lprimary> –> INTLITERAL **#process_literal**
29. <lprimary> –> <ident>
30. <lprimary –> (<condition>)
31. <lprimary> –> FALSEOP **#process_op**
32. <lprimary> –> TRUEOP **#process_op**
33. <lprimary> –> NULLOP **#process_op**
34. <RelOp> –> < **#process_op**
35. <RelOp> –> <= **#process_op**
36. <RelOp> –> > **#process_op**
37. <RelOp> –> >= **#process_op**
38. <RelOP> –> = **#process_op**
39. <RelOp> –> <> **#process_op**
40. <system goal> –> <program> SCANEOF **#finish**
41. <ident> –> ID **#process_id**

For example:
Input file:

```
begin
a:= BB - 314 + A;
end
```

Output.C file:

```
/*
C program of MICRO program E:\MICRO\MICRO1.IN
Mon Mar 05 08:11:14 2012
*/

#include<stdio.h>
main()
{
int A;
int BB;
int Temp1;
int Temp2;
```

Output.TMP file:

```
        Temp1  =  BB - 314;
        Temp2  =  Temp1 + A;
        A  =  Temp2;
        return 0;
        }
/* PROGRAMED COMPILED WITH NO ERRORS. */
```

Output.LIS file:

```
Listing of MICRO program E:\MICRO\MICRO1.IN  Mon Mar 05 08:11:14 2012

1       begin
2       a:= BB - 314 + A;
3       end

        0       LEXICAL ERRORS
        0       SYNTAX ERRORS
        PROGRAMED COMPILED WITH NO ERRORS.
```

Final Output.C file:

```
        /*
        C program of MICRO program E:\MICRO\MICRO1.IN
        Mon Mar 05 08:11:14 2012
        */

        #include<stdio.h>
        main()
        {
        int A;
        int BB;
        int Temp1;
        int Temp2;
        Temp1  =  BB - 314;
        Temp2  =  Temp1 + A;
        A  =  Temp2;
        return 0;
        }
/* PROGRAMED COMPILED WITH NO ERRORS. */
```