CS 4013: Compiler Construction

Project 3 and 4

Michael Collett

INTRODUCTION

------------

The grammar decorations were performed on my previous LL(1) grammar from project 2. Decorating my parsed grammar was the theoretical aspect of this project while implementing it became the practical portion. This compiler now performs type checking, scope checking, procedure call checking and computes the local address of all variables. The output files are memoryAddrs.txt,

list.txt and tok.txt.

METHODOLOGY

------------

**Bold: terminals**

*Non-bold: non-terminals*

TYPE CHECKING - (#) denotes position, no (#) means it’s at the end/obvious, id.type means getType(id.lex)

SCOPE CHECKING

MEMORY CHECKING

PROCEDURE CHECKING

RETURN

INHERITED

1.1 *program* --> **program id** (1)(2) **(**idlst**);** VOID VOID

*program’*

(1) {offset := 0}

(2) checkAddGreenNode(id.lex, PGNAME) set GN pointer

1.2.1 *program’* --> *subdeclarations* VOID VOID

*compound\_statement*

**.**

1.2.2 *program’* --> *compound\_statement*

**.**

1.2.3 *program’* --> *declarations*

*program’’*

1.3.1 *program’’ --> subdeclarations* VOID VOID

*compound\_statement*

**.**

1.3.2 *program’’ --> compound\_statement*

**.**

2.1 *idlst* --> **id** (1)*idlst’* VOID VOID

(1) checkAddBlueNode(id.lex, PGPARAM)

2.2.1 *idlst’* --> **, id** (1)*idlst’*

(1) checkAddBlueNode(id.lex, PGPARAM) VOID VOID

2.2.2 idlst’ --> ϵ

3.1 *declarations* --> **var** **id:** *type*(1) (2)**;** *declarations’* VOID VOID

(1) checkAddBlueNode(id.lex, type.type)

(2) offset += type.width

3.2.1 *declarations’* --> **var** **id:** *type* (1) (2)**;** *declarations’* VOID VOID

(1) checkAddBlueNode(id.lex, type.type)

(2) offset += type.width

3.2.2 *declarations’* --> ϵ

4.1 *type* --> *standard\_type* VOID TYPE/WIDTH

{type.type := standard\_type.type}

{width = standard\_type.width}

4.2 *type* --> **array** **[num** **..** **num]** **of** *standard\_type*

{type.type := “a” + standard\_type.type}

Check if num and num are ints and check if the second one is bigger than the first

{width = standard\_type.width \* (num2-num1+1)}

5.1 *standard*\_*type* --> **integer** VOID TYPE/WIDTH

{standard\_type.type := int}

{width := 4}

5.2 *standard\_type*  --> **real**

{standard\_type.type := real}

{width := 8}

6.1 *subdeclarations* --> *subdeclaration* **;** (1)*subdeclarations’* VOID VOID

(1) pop stack

6.2.1 *subdeclarations’* --> *subdeclaration* **;** (1) *subdeclarations’* VOID VOID

(1) pop stack

6.2.2 *subdeclarations’* --> ϵ

7.1 *subdeclaration* --> *subprogram\_head* VOID VOID

*subdeclaration’*

7.2.1 *subdeclaration’* --> *subdeclarations* VOID VOID

*compound\_statement*

7.2.2 *subdeclaration’* --> *compound\_statement*

7.2.3 *subdeclaration’* --> *declarations*

*subdeclaration’’*

7.3.1 *subdeclaration’’ --> subdeclarations* VOID VOID

*compound\_statement*

7.3.2 *subdeclaration’’ --> compound\_statement*

8.1 *subprogram\_head* --> **procedure** **id** (1) *subprogram\_head’* VOID VOID

(1) checkAddGreenNode(id.lex, PPNAME) set GN pointer

{offset := 0}

8.2.1 *subprogram\_head’* --> **;** VOID VOID

8.2.2 *subprogram\_head’* --> *arguments* **;**

9.1 *arguments* --> **(***parameter\_list***)**VOID VOID

10.1 *parameter\_list* --> **id** **:** *type* (1) *parameter\_list’* VOID VOID

(1) checkAddBlueNode(id.lex, PPARAM, type.type)

10.2.1 *parameter\_list’*  --> **;** ***id*** **:** *type* (1) *parameter\_list’* VOID VOID

(1) checkAddBlueNode(id.lex, PPARAM, type.type)

10.2.2 *parameter\_list’*  --> ϵ

11.1 *compound\_statement* --> **begin** *compound\_statement’* VOID VOID

11.2.1 *compound\_statement’* --> *optional\_statements*

**end**

11.2.2 *compound\_statement’* --> **end** VOID VOID

12.1 *optional\_statements* --> *statement\_list* VOID VOID

13.1 *statement\_list --> statement statement\_list’* VOID VOID

13.2.1 *statement\_list’ -->* **;** *statement statement\_list’* VOID VOID

13.2.2 *statement\_list’ -->* ϵ

14.1 *statement* --> *variable* **assignop** *expression* VOID VOID

{if variable.type == expression.type or either variable.type or expression.type are ERR: Do Nothing

Else: ERR\*}

14.2 *statement* --> *procedure\_statement*

14.3 *statement* --> *compound\_statement*

14.4 *statement* --> **while** *expression* **do** *statement*

{if expression != Bool: ERR\*}

14.5 *statement* --> **if** *expression* () **then** *statement* *statement’*

{if expression != Bool: ERR\*}

14.6.1 *statement’* --> **else** *statement* VOID VOID

14.6.2 *statement’* --> ϵ

15.1 *variable* --> **id** (1)*variable’* (2) VOID TYPE

(1){variable’.i := id.type}

(2){variable.type := variable’.type}

15.2.1 *variable’* --> **[***expression***]** TYPE TYPE

|  |  |  |
| --- | --- | --- |
| variable’.type | expression.type | variable’.i |
| INT | INT | AINT |
| REAL | INT | AREAL |
| ERR\* | Anything except ERR | Anything except ERR |
| ERR | Anything | ERR |
| ERR | ERR | Anything |

15.2.2 *variable’* --> ϵ

{variable’.type := variable’.i}

16.1 *procedure\_statement* --> **call** **id** *procedure\_statement’* VOID VOID

procedurePointer := checkProcedure(id.lex) (print SYM ERR if not)

(global or procedure\_statement’.i := procedurePointer)

16.2.1 *procedure\_statement’* --> **(***expression\_list***)** POINTER VOID

16.2.2 *procedure\_statement’* --> ϵ

Check if procedure had originally defined inputs

17.1 *expression\_list*  --> *expression expression\_list’* POINTER VOID

#If pointer is null do nothing

checkParam(expression.type)

advance parameter (expression\_list.i)

17.2.1 *expression\_list* ‘ --> **,** *expression expression\_list’* POINTER VOID

#If pointer is null do nothing

checkParam(expression.type)

advance parameter (expression\_list.i)

17.2.2 *expression\_list* ‘ --> ϵ

CheckNoMoreParams()

18.1 *expression* --> *simple\_expression* (1) *expression’* (2) VOID TYPE

(1) {expression’.i := simple\_expression’.type}

(2) {expression.type := expression’type}

18.2.1 *expression’* --> **relop** *simple\_expression* TYPE TYPE

|  |  |  |  |
| --- | --- | --- | --- |
| expression’.type (out) | expression’.i (in) | relop.op (in) | se.type(in) |
| BOOL | REAL/INT | all relop ops | REAL/INT |
| ERR\* | non equivalent | all relop ops | non equivalent |
| ERR | Anything | all relop ops | ERR |
| ERR | ERR | all relop ops | anything |

18.2.2 *expression’* --> ϵ

{expression’.type := expression’.i}

19.1 *simple\_expression* --> *term* (1) *simple\_expression’* (2)TYPE TYPE

(1) {simple\_expression’.i := term.type}

(2) {simple\_expression.type := simple\_expression’.type}

19.2 *simple\_expression* --> *sign term* (1) *simple\_expression’* (2)

(1) {simple\_expression’.i := term.type}

(2) {simple\_expression.type := simple\_expression’.type}

19.3.1 *simple\_expression’* --> **addop** *term* (1) *simple\_expression’* (2)TYPE TYPE

(1)

|  |  |  |  |
| --- | --- | --- | --- |
| se2’.i (out) | term.type (in) | addop.op (in) | se1’.i (in) |
| INT | INT | + - | INT |
| ERR\* | INT BOOL | + - | REAL BOOL |
| ERR\* | REAL BOOL | + - | INT BOOL |
| REAL | REAL | + - | REAL |
| ERR | ERR | + - or | Anything |
| ERR | Anything | + - or | ERR |
| BOOL | BOOL | or | BOOL |
| ERR\* | BOOL INT REAL | or | INT REAL |
| ERR\* | INT REAL | or | BOOL INT REAL |

(2) {simple\_expression’1.type := simple\_expression’2.type}

19.3.2 *simple\_expression’* --> ϵ

{simple\_expression’.type := simple\_expression.i}

20.1 *term* --> *factor* (1) *term’* (2) VOID TYPE

(1) {term’.i := factor.type}

(2) {term.type := term’.type}

20.2.1 *term’* --> **mulop** *factor* (1) *term’* (2) TYPE TYPE

(1)

|  |  |  |  |
| --- | --- | --- | --- |
| term2’.i (out) | factor.type (in) | mulop.op (in) | term1’.i (in) |
| INT | INT | \* / mod div | INT |
| ERR\* | REAL BOOL | \* / mod div | INT BOOL |
| ERR\* | INT BOOL | \* / mod div | REAL BOOL |
| REAL | REAL | \* / mod div | REAL |
| ERR | Anything | \* / mod div and | ERR |
| ERR | ERR | \* / mod div and | Anything |
| BOOL | BOOL | and | BOOL |
| ERR\* | BOOL | and | REAL INT |
| ERR\* | REAL INT | and | BOOL |

NOTE: If mulop.op is / must check for divide by zero

(2) {term.type := term’.type}

20.2.2 *term’* --> ϵ

{term’.type := term’.i}

21.1 *factor* --> **id** (1)*factor’* (2) VOID TYPE

(1){factor’.i := id.type}

(2){factor.type := factor’.type}

21.2 *factor* --> **num**

{factor.type := check num for either sreal, lreal or int}

21.3 *factor* --> **(** *expression* **)**

{factor.type := expression.type}

21.4 *factor* --> **not** *factor (1)(2)*

*(1) {if factor.type == BOOL: we good; else; SYM ERR}*

*(2) {factor1.type := factor2.type}*

21.5.1 *factor’* --> ***[****expression****]*** TYPE TYPE

|  |  |  |
| --- | --- | --- |
| factor’.type | expression.type | factor’.i |
| INT | INT | AINT |
| REAL | INT | AREAL |
| ERR\* | Anything except ERR | Anything except ERR |
| ERR | Anything | ERR |
| ERR | ERR | Anything |

21.5.2 *factor’* --> ϵ

{factor’.type := factor’.i}

22.1 *sign* --> **+** VOID VOID

22.2 *sign* --> **-**

IMPLEMENTATION

--------------

I began with the set of productions previously massaged. These productions were first decorated for type checking and then scope and then memory address offsets. The grammar is displayed above in the methodology section. I built an especially peculiar doubly-linked list out of blue and green nodes to handle the type and scope checking. Semantic errors were printed below each line just as syntactic or lexical errors were.

DISCUSSION AND CONCLUSIONS

--------------------------

The declaration processing took time to understand, but is well worth it. I really enjoyed this project and found it both fun and challenging. I feel very comfortable with the material and my knowledge of it after having implemented the front end of a compiler. It would be quite interesting to automate a significant portion of this process, namely, lexical analysis and syntax analysis.

REFERENCES

----------

The C Programing Language By Brian Kernighan and Dennis Ritchie

Standard Libraries in C

Memory:

Variable Offset

a 0

b 4

c 12

d 0

e 0

e 0

x 0

Listing:

1.

2. program test (input, output);

3. var a : integer;

4. var b : real;

5. var c : array [1..2] of integer;

6.

7. procedure proc1(x:integer; y:real;

8. z:array [1..2] of integer; q: real);

9. var d: integer;

10. begin

11. a:= 2;

12. z[a] := 4;

13. c[3] := 3

14. end;

15.

16. procedure proc2(x: integer; y: integer);

17. var e: real;

18.

19. procedure proc3(n: integer; z: real);

20. var e: integer;

21.

22. procedure proc4(a: integer; z: array [1..3] of real);

23. var x: integer;

24. begin

25. a:= e

26. end;

27.

28. begin

29. a:= e;

30. e:= c[e]

31. end;

32.

33. begin

34. call proc1(x, e, c, b);

35. call proc3(c[1], e);

36. e := e + 4.44;

-SEMANTIC ERROR-

: incorrect types, variable and expression must be of the same type

37. a:= (a mod y) div x;

38. while ((a >= 4) and ((b <= e)

39. or (not (a = c[a])))) do

40. begin

41. a:= c[a] + 1

42. end

-SEMANTIC ERROR-

: incorrect types, variable and expression must be of the same type

43. end;

44.

45. begin

46. call proc2(c[4], c[5]);

47. call proc2(c[4],2);

48. if (a < 2) then a:= 1 else a := a + 2;

-SEMANTIC ERROR-

: incorrect types, variable and expression must be of the same type

49. if (b > 4.2) then a := c[a]

50. end.

Tokens:

Line No. Lexeme Token Type Attribute

2 program 30 0

2 test 1 0x7fa922c02e30

2 ( 2 81

2 input 1 0x7fa922c02eb0

2 , 4 85

2 output 1 0x7fa922c02f30

2 ) 2 82

2 ; 4 86

3 var 31 0

3 a 1 0x7fa922c030d0

3 : 6 0

3 integer 34 0

3 ; 4 86

4 var 31 0

4 b 1 0x7fa922c03220

4 : 6 0

4 real 35 0

4 ; 4 86

5 var 31 0

5 c 1 0x7fa922c03370

5 : 6 0

5 array 32 0

5 [ 2 83

5 1 10 0

5 .. 5 0

5 2 10 0

5 ] 2 84

5 of 33 0

5 integer 34 0

5 ; 4 86

7 procedure 37 0

7 proc1 1 0x7fa922c03610

7 ( 2 81

7 x 1 0x7fa922c03690

7 : 6 0

7 integer 34 0

7 ; 4 86

7 y 1 0x7fa922c03770

7 : 6 0

7 real 35 0

7 ; 4 86

8 z 1 0x7fa922c03910

8 : 6 0

8 array 32 0

8 [ 2 83

8 1 10 0

8 .. 5 0

8 2 10 0

8 ] 2 84

8 of 33 0

8 integer 34 0

8 ; 4 86

8 q 1 0x7fa922c03b40

8 : 6 0

8 real 35 0

8 ) 2 82

8 ; 4 86

9 var 31 0

9 d 1 0x7fa922c03d00

9 : 6 0

9 integer 34 0

9 ; 4 86

10 begin 38 70

11 a 1 0x7fa922c030d0

11 := 3 0

11 2 10 0

11 ; 4 86

12 z 1 0x7fa922c03910

12 [ 2 83

12 a 1 0x7fa922c030d0

12 ] 2 84

12 := 3 0

12 4 10 0

12 ; 4 86

13 c 1 0x7fa922c03370

13 [ 2 83

13 3 10 0

13 ] 2 84

13 := 3 0

13 3 10 0

14 end 38 71

14 ; 4 86

16 procedure 37 0

16 proc2 1 0x7fa922c04210

16 ( 2 81

16 x 1 0x7fa922c03690

16 : 6 0

16 integer 34 0

16 ; 4 86

16 y 1 0x7fa922c03770

16 : 6 0

16 integer 34 0

16 ) 2 82

16 ; 4 86

17 var 31 0

17 e 1 0x7fa922c04570

17 : 6 0

17 real 35 0

17 ; 4 86

19 procedure 37 0

19 proc3 1 0x7fa922c046c0

19 ( 2 81

19 n 1 0x7fa922c04740

19 : 6 0

19 integer 34 0

19 ; 4 86

19 z 1 0x7fa922c03910

19 : 6 0

19 real 35 0

19 ) 2 82

19 ; 4 86

20 var 31 0

20 e 1 0x7fa922c04570

20 : 6 0

20 integer 34 0

20 ; 4 86

22 procedure 37 0

22 proc4 1 0x7fa922c04b30

22 ( 2 81

22 a 1 0x7fa922c030d0

22 : 6 0

22 integer 34 0

22 ; 4 86

22 z 1 0x7fa922c03910

22 : 6 0

22 array 32 0

22 [ 2 83

22 1 10 0

22 .. 5 0

22 3 10 0

22 ] 2 84

22 of 33 0

22 real 35 0

22 ) 2 82

22 ; 4 86

23 var 31 0

23 x 1 0x7fa922c03690

23 : 6 0

23 integer 34 0

23 ; 4 86

24 begin 38 70

25 a 1 0x7fa922c030d0

25 := 3 0

25 e 1 0x7fa922c04570

26 end 38 71

26 ; 4 86

28 begin 38 70

29 a 1 0x7fa922c030d0

29 := 3 0

29 e 1 0x7fa922c04570

29 ; 4 86

30 e 1 0x7fa922c04570

30 := 3 0

30 c 1 0x7fa922c03370

30 [ 2 83

30 e 1 0x7fa922c04570

30 ] 2 84

31 end 38 71

31 ; 4 86

33 begin 38 70

34 call 43 0

34 proc1 1 0x7fa922c03610

34 ( 2 81

34 x 1 0x7fa922c03690

34 , 4 85

34 e 1 0x7fa922c04570

34 , 4 85

34 c 1 0x7fa922c03370

34 , 4 85

34 b 1 0x7fa922c03220

34 ) 2 82

34 ; 4 86

35 call 43 0

35 proc3 1 0x7fa922c046c0

35 ( 2 81

35 c 1 0x7fa922c03370

35 [ 2 83

35 1 10 0

35 ] 2 84

35 , 4 85

35 e 1 0x7fa922c04570

35 ) 2 82

35 ; 4 86

36 e 1 0x7fa922c04570

36 := 3 0

36 e 1 0x7fa922c04570

36 + 9 96

36 4.44 11 0

36 ; 4 86

37 a 1 0x7fa922c030d0

37 := 3 0

37 ( 2 81

37 a 1 0x7fa922c030d0

37 mod 8 79

37 y 1 0x7fa922c03770

37 ) 2 82

37 div 8 78

37 x 1 0x7fa922c03690

37 ; 4 86

38 while 40 75

38 ( 2 81

38 ( 2 81

38 a 1 0x7fa922c030d0

38 >= 7 92

38 4 10 0

38 ) 2 82

38 and 8 80

38 ( 2 81

38 ( 2 81

38 b 1 0x7fa922c03220

38 <= 7 88

38 e 1 0x7fa922c04570

38 ) 2 82

39 or 9 77

39 ( 2 81

39 not 42 0

39 ( 2 81

39 a 1 0x7fa922c030d0

39 = 7 91

39 c 1 0x7fa922c03370

39 [ 2 83

39 a 1 0x7fa922c030d0

39 ] 2 84

39 ) 2 82

39 ) 2 82

39 ) 2 82

39 ) 2 82

39 do 40 76

40 begin 38 70

41 a 1 0x7fa922c030d0

41 := 3 0

41 c 1 0x7fa922c03370

41 [ 2 83

41 a 1 0x7fa922c030d0

41 ] 2 84

41 + 9 96

41 1 10 0

42 end 38 71

43 end 38 71

43 ; 4 86

45 begin 38 70

46 call 43 0

46 proc2 1 0x7fa922c04210

46 ( 2 81

46 c 1 0x7fa922c03370

46 [ 2 83

46 4 10 0

46 ] 2 84

46 , 4 85

46 c 1 0x7fa922c03370

46 [ 2 83

46 5 10 0

46 ] 2 84

46 ) 2 82

46 ; 4 86

47 call 43 0

47 proc2 1 0x7fa922c04210

47 ( 2 81

47 c 1 0x7fa922c03370

47 [ 2 83

47 4 10 0

47 ] 2 84

47 , 4 85

47 2 10 0

47 ) 2 82

47 ; 4 86

48 if 39 72

48 ( 2 81

48 a 1 0x7fa922c030d0

48 < 7 90

48 2 10 0

48 ) 2 82

48 then 39 73

48 a 1 0x7fa922c030d0

48 := 3 0

48 1 10 0

48 else 39 74

48 a 1 0x7fa922c030d0

48 := 3 0

48 a 1 0x7fa922c030d0

48 + 9 96

48 2 10 0

48 ; 4 86

49 if 39 72

49 ( 2 81

49 b 1 0x7fa922c03220

49 > 7 93

49 4.2 11 0

49 ) 2 82

49 then 39 73

49 a 1 0x7fa922c030d0

49 := 3 0

49 c 1 0x7fa922c03370

49 [ 2 83

49 a 1 0x7fa922c030d0

49 ] 2 84

50 end 38 71

50 . 4 87

-1 EOF 20 0

FIB ERROR LIST:

1. program fib(input, output);

2. var n: integer; var p: integer;

3. var q: real;

4. var numsArray : array [13..12] of integer;?

LEXERR: UnrecognizedSymbol ?

-SYNTAX ERROR-

Expecting one of: begin or procedure

Received: ?

5.

6. procedure fib(a : integer; b : real; c : real);

7. begin

8. if a <= 1 then fib := c

9. else call fib(a - 1, c, b + c)

10. end;

11.

12. procedure fib2(a : integer);

13. var b : integer; var c : integer; var sum : integer;

14. procedure rawr3(b : real);

15. var q : integer;

16. q := b + 2.0.1;

-SYNTAX ERROR-

Expecting one of: begin or procedure

Received: q

17. call fib2(q)

18. end;

19. begin

20. a := aasdlfjlwkjerjkwle - 1;

LEXERR: IDTOOLONG aasdlfjlwkjerjkwle

-SYNTAX ERROR-

Expecting one of: ID, num, (, not +, -

Received: aasdlfjlwkjerjkwle

21. b := 0;

22. sum := 1;

23. claksdjfasdflkaj := b;

LEXERR: IDTOOLONG claksdjfasdflkaj

-SYNTAX ERROR-

Expecting one of: ID, call, begin, while, if

Received: claksdjfasdflkaj

24. while (a\_ > 0) do

LEXERR: UnrecognizedSymbol \_

-SYNTAX ERROR-

Expecting one of: [ mulop addop relop do then ] , ) ; end else

Received: \_

25. begin

26. a := a - 1;

27. b := sum;

28. sum := c + sum;

29. c := b

30. end;

31. fib2 := sum

32. end;

33.

34. procedure init;

35. begin

36. n := 12;

37. if (123.4.5 and 2) or 3 then p := 12

-SYNTAX ERROR-

Expecting one of: mulop addop relop do then ] , ) ; end else

Received: .

38. else p := 14;

39. numsArray[3.1] := 15.56;

40. q := 12

41. end;

42.

43. begin

44. call init;

45. call rawr3(34);

46. call writeln(+6\*q/p + 4);

47. call writeln(fib2\*n);

48. call writeln(numsArray[3] mod 15)

49. end.

-SYNTAX ERROR-

Expecting one of: ;

Received: .

-SYNTAX ERROR-

Expecting one of: .

Received: EOF

FIB ERROR TOK

Line No. Lexeme Token Type Attribute

1 program 30 0

1 fib 1 0x7fca8dc02e10

1 ( 2 81

1 input 1 0x7fca8dc02e90

1 , 4 85

1 output 1 0x7fca8dc02f10

1 ) 2 82

1 ; 4 86

2 var 31 0

2 n 1 0x7fca8dc02ff0

2 : 6 0

2 integer 34 0

2 ; 4 86

2 var 31 0

2 p 1 0x7fca8dc03100

2 : 6 0

2 integer 34 0

2 ; 4 86

3 var 31 0

3 q 1 0x7fca8dc03210

3 : 6 0

3 real 35 0

3 ; 4 86

4 var 31 0

4 numsArray 1 0x7fca8dc03320

4 : 6 0

4 array 32 0

4 [ 2 83

4 13 10 0

4 .. 5 0

4 12 10 0

4 ] 2 84

4 of 33 0

4 integer 34 0

4 ; 4 86

4 ? 99 101

6 procedure 37 0

6 fib 1 0x7fca8dc02e10

6 ( 2 81

6 a 1 0x7fca8dc03610

6 : 6 0

6 integer 34 0

6 ; 4 86

6 b 1 0x7fca8dc036f0

6 : 6 0

6 real 35 0

6 ; 4 86

6 c 1 0x7fca8dc037d0

6 : 6 0

6 real 35 0

6 ) 2 82

6 ; 4 86

7 begin 38 70

8 if 39 72

8 a 1 0x7fca8dc03610

8 <= 7 88

8 1 10 0

8 then 39 73

8 fib 1 0x7fca8dc02e10

8 := 3 0

8 c 1 0x7fca8dc037d0

9 else 39 74

9 call 43 0

9 fib 1 0x7fca8dc02e10

9 ( 2 81

9 a 1 0x7fca8dc03610

9 - 9 97

9 1 10 0

9 , 4 85

9 c 1 0x7fca8dc037d0

9 , 4 85

9 b 1 0x7fca8dc036f0

9 + 9 96

9 c 1 0x7fca8dc037d0

9 ) 2 82

10 end 38 71

10 ; 4 86

12 procedure 37 0

12 fib2 1 0x7fca8dc03dc0

12 ( 2 81

12 a 1 0x7fca8dc03610

12 : 6 0

12 integer 34 0

12 ) 2 82

12 ; 4 86

13 var 31 0

13 b 1 0x7fca8dc036f0

13 : 6 0

13 integer 34 0

13 ; 4 86

13 var 31 0

13 c 1 0x7fca8dc037d0

13 : 6 0

13 integer 34 0

13 ; 4 86

13 var 31 0

13 sum 1 0x7fca8dc04140

13 : 6 0

13 integer 34 0

13 ; 4 86

14 procedure 37 0

14 rawr3 1 0x7fca8dc04250

14 ( 2 81

14 b 1 0x7fca8dc036f0

14 : 6 0

14 real 35 0

14 ) 2 82

14 ; 4 86

15 var 31 0

15 q 1 0x7fca8dc03210

15 : 6 0

15 integer 34 0

15 ; 4 86

16 q 1 0x7fca8dc03210

16 := 3 0

16 b 1 0x7fca8dc036f0

16 + 9 96

16 2.0 11 0

16 . 4 87

16 1 10 0

16 ; 4 86

17 call 43 0

17 fib2 1 0x7fca8dc03dc0

17 ( 2 81

17 q 1 0x7fca8dc03210

17 ) 2 82

18 end 38 71

18 ; 4 86

19 begin 38 70

20 a 1 0x7fca8dc03610

20 := 3 0

20 aasdlfjlwkjerjkwle 99 100

20 - 9 97

20 1 10 0

20 ; 4 86

21 b 1 0x7fca8dc036f0

21 := 3 0

21 0 10 0

21 ; 4 86

22 sum 1 0x7fca8dc04140

22 := 3 0

22 1 10 0

22 ; 4 86

23 claksdjfasdflkaj 99 100

23 := 3 0

23 b 1 0x7fca8dc036f0

23 ; 4 86

24 while 40 75

24 ( 2 81

24 a 1 0x7fca8dc03610

24 \_ 99 101

24 > 7 93

24 0 10 0

24 ) 2 82

24 do 40 76

25 begin 38 70

26 a 1 0x7fca8dc03610

26 := 3 0

26 a 1 0x7fca8dc03610

26 - 9 97

26 1 10 0

26 ; 4 86

27 b 1 0x7fca8dc036f0

27 := 3 0

27 sum 1 0x7fca8dc04140

27 ; 4 86

28 sum 1 0x7fca8dc04140

28 := 3 0

28 c 1 0x7fca8dc037d0

28 + 9 96

28 sum 1 0x7fca8dc04140

28 ; 4 86

29 c 1 0x7fca8dc037d0

29 := 3 0

29 b 1 0x7fca8dc036f0

30 end 38 71

30 ; 4 86

31 fib2 1 0x7fca8dc03dc0

31 := 3 0

31 sum 1 0x7fca8dc04140

32 end 38 71

32 ; 4 86

34 procedure 37 0

34 init 1 0x7fca8dc051f0

34 ; 4 86

35 begin 38 70

36 n 1 0x7fca8dc02ff0

36 := 3 0

36 12 10 0

36 ; 4 86

37 if 39 72

37 ( 2 81

37 123.4 11 0

37 . 4 87

37 5 10 0

37 and 8 80

37 2 10 0

37 ) 2 82

37 or 9 77

37 3 10 0

37 then 39 73

37 p 1 0x7fca8dc03100

37 := 3 0

37 12 10 0

38 else 39 74

38 p 1 0x7fca8dc03100

38 := 3 0

38 14 10 0

38 ; 4 86

39 numsArray 1 0x7fca8dc03320

39 [ 2 83

39 3.1 11 0

39 ] 2 84

39 := 3 0

39 15.56 11 0

39 ; 4 86

40 q 1 0x7fca8dc03210

40 := 3 0

40 12 10 0

41 end 38 71

41 ; 4 86

43 begin 38 70

44 call 43 0

44 init 1 0x7fca8dc051f0

44 ; 4 86

45 call 43 0

45 rawr3 1 0x7fca8dc04250

45 ( 2 81

45 34 10 0

45 ) 2 82

45 ; 4 86

46 call 43 0

46 writeln 1 0x7fca8dc05b40

46 ( 2 81

46 + 9 96

46 6 10 0

46 \* 8 94

46 q 1 0x7fca8dc03210

46 / 8 95

46 p 1 0x7fca8dc03100

46 + 9 96

46 4 10 0

46 ) 2 82

46 ; 4 86

47 call 43 0

47 writeln 1 0x7fca8dc05b40

47 ( 2 81

47 fib2 1 0x7fca8dc03dc0

47 \* 8 94

47 n 1 0x7fca8dc02ff0

47 ) 2 82

47 ; 4 86

48 call 43 0

48 writeln 1 0x7fca8dc05b40

48 ( 2 81

48 numsArray 1 0x7fca8dc03320

48 [ 2 83

48 3 10 0

48 ] 2 84

48 mod 8 79

48 15 10 0

48 ) 2 82

49 end 38 71

49 . 4 87

-1 EOF 20 0