Webový překladač PL/0

Tomáš Linhart
Pavel Třeštík

Navržená gramatika

```
program = block "." ;
block = [ "const" ident [":" data_type] "=" value {"," ident [":" data_type] "=" value} ";"]
      [ "var" ident [":" data_type] {"," ident [":" data_type]} ";"]
      { "procedure" ident [ "(" ident [ : data_type ] {"," ident [ : data_type ]} ")" ] ";" block ";" } statement ;


statement = [ ident ":=" {ident ":="} expression
      | "{" ident {, ident} "} := {" value{, value} "}"
      | "call" ident
      | "?" ident
      | "!" expression
      | "begin" statement {";" statement } "end"
      | "if" condition "then" statement [ "else" statement ]
      | "(" condition ") ? " "return" statement ":" "return" statement
      | "while" condition "do" statement
      | "for" number "to" number "do" statement
      | "foreach" ident "in" array_ident "do" statement ]
      | "return" value;


condition = "odd" expression
      | expression ("="|"#"|"<"|"<="|">"|">=") expression ;


expression = [ "+"|"-"] term { ("+"|"-") term};
term = factor {("*"|"/") factor};
factor = ident | number | value | "(" expression ")";
```

Konečná gramatika (při odevzdání práce)

```
program = block "." ;
block = [ "const" ident [":" data_type] "=" value {"," ident [":" data_type] "=" value} ";"]
        [ "var" ident [":" data_type] {"," ident [":" data_type]} ";"]
        { "procedure" [data_type] ident [ "(" ident [ : data_type ] {"," ident [ : data_type ]} ")" ] ";" block ";" } statement ;

statement = [ ident ":=" expression
            | "{" ident {, ident} "} := {" value{, value} "}"
            | "call" ident ["(" expression {"," expression } ")"]
            | "?" ident
            | "!" expression
            | "begin" statement {";" statement } "end"
            | "if" condition_expression "then" statement [ "else" statement ]
            | "(" condition_expression ") ? " statement ":" statement
            | "while" condition_expression "do" statement
            | "for" expression "to" expression "do" statement
            | "return" expression ];

condition_expression = ["~"] condition { ("&"|"|") ["~"] condition } ;

condition = "odd" expression |
        expression ("="|"#"|"<"|"<="|">"|">=") expression ;

expression = ["+"|"-"] term {("+"|"-") term} | "call" ident;

term = ["~"] factor { ("*"|"/"|"&"|"|") ["~"] factor};

factor = ident | value | "(" expression ")";
```

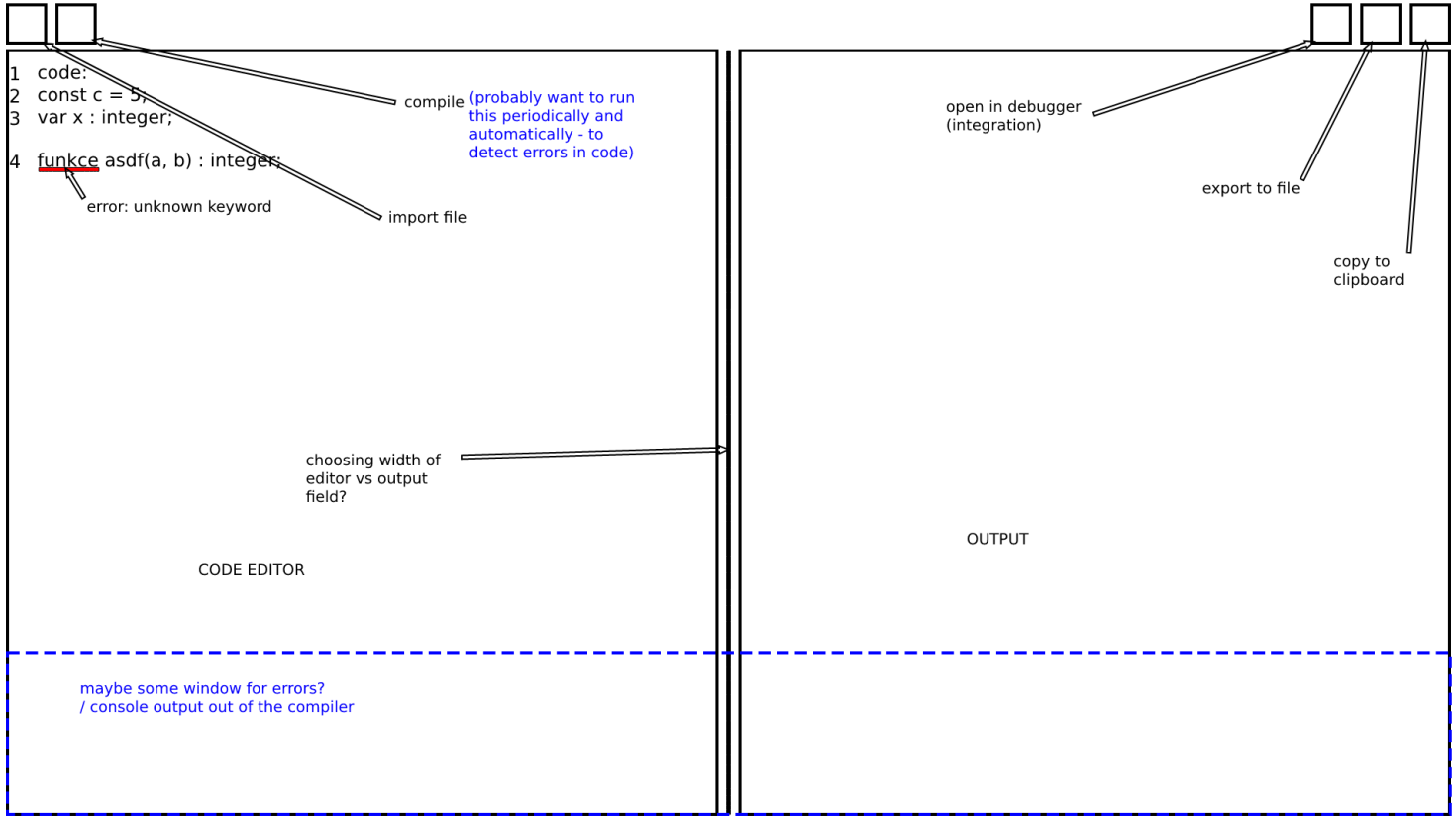
Implementace překladače (backend)

- Využití alternativy lex pro JS: https://github.com/zaach/jison-lex
  - Generování parsovacího skriptu
- Překlad pomocí rekurzivního sestupu
  - Nedostatky navržené gramatiky
  - Řešeno úpravou gramatiky či “odstraněním” vlastnosti

Některé problémy s implementací rekuzivního sestupu

- For cyklus
- Stack pointer v překladači? Nebo kopírování hodnoty na SP? Nebo..
- Ternární operátor (resp. jeho hloupější verze – neumí “return” hodnoty)
- Je to statement = nelze předělat do na expression
- Podobný problém s call, ale ten lze přidat do expression
- Další drobnosti:
- Vkládání JMP instrukcí na začátek listu (podle cvičení) = procedury
- Rozdíly v použité dokumentaci a interpretu z předchozího roku

Grafický návrh



Reálný stav

Boolean | Load example | Run tests

```
const m=7, n=85;
var x, y, z, q, r;

procedure nasobeni;
var a, b;
begin a := x; b:= y; z:=0;
    while b > 0 do
    begin
        if odd b then z := z + a;
            a := 2 * a; b := b / 2;
    end;
end;

procedure deleni;
var w;
begin r:=x; q := 0; w:= y;
    while w <= r do w := 2*w;
    while w > y do
    begin q:=2*q; w:=w/2;
        if w <= r   then
        begin r:=r-w; q:=q+1;
        end;
    end;
end;

procedure gcd;
```

0	JMP	0	13
1	JMP	0	2
2	INT	0	3
3	LOD	1	3
4	LIT	0	1
5	OPR	0	2
6	STO	1	3
7	LOD	1	3
8	LIT	0	2
9	OPR	0	13
10	JMC	0	12
11	CAL	1	2
12	RET	0	0
13	INT	0	4
14	LIT	0	0
15	STO	0	3
16	CAL	0	2
17	RET	0	0

ParserTerm>Welcome to PL0 parser :)
ParserTerm>Debugger successfuly connected

Zvýraznění chyb

Boolean | Load example | Run tests

Assignment statement failed. Identifier: proceduure not found

View Problem (Alt+F8) No quick fixes available

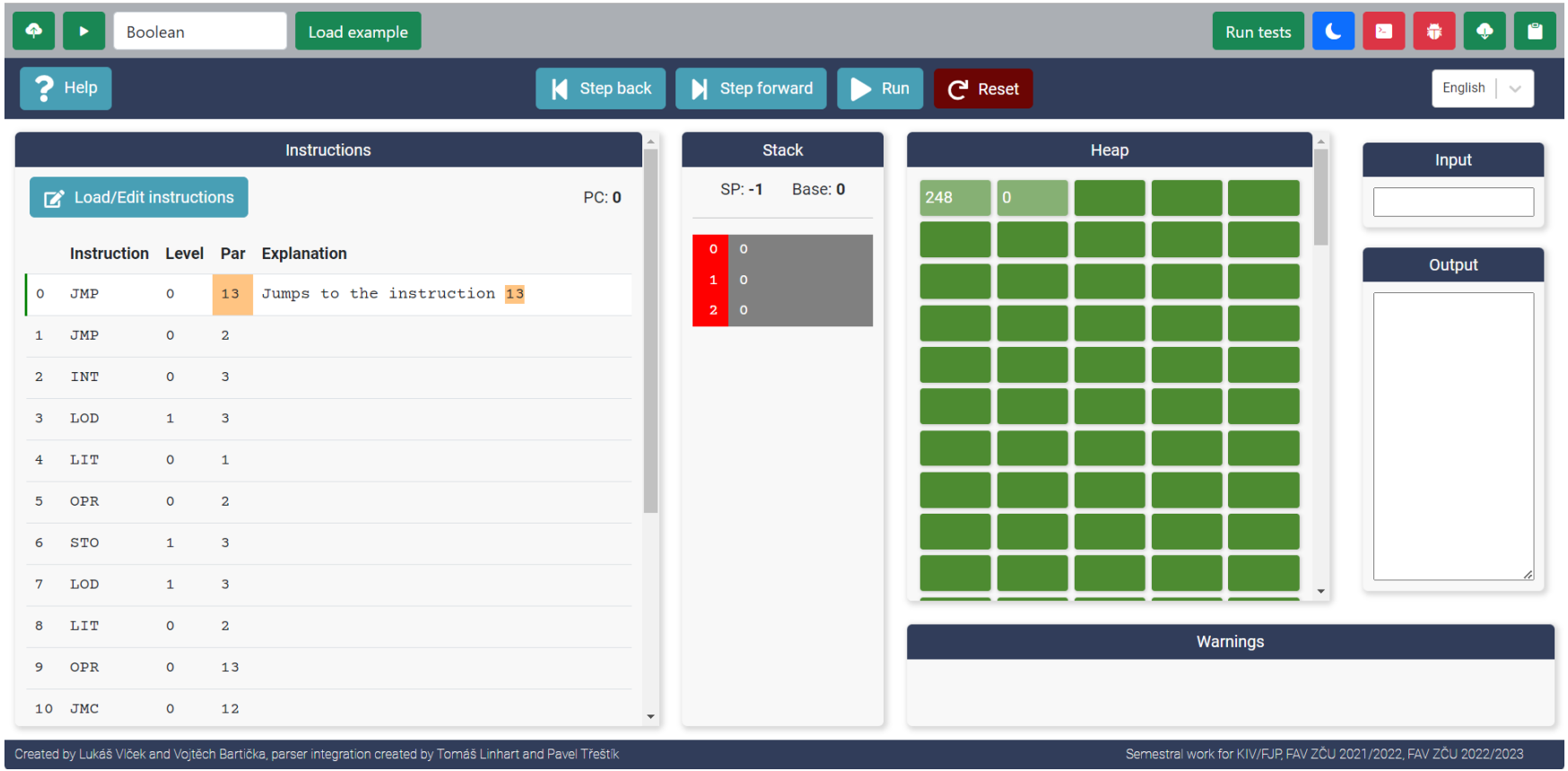
```
proceduure nasobeni;
var a, b;
begin a := x; b:= y; z:=0;
    while b > 0 do
    begin
        if odd b then z := z + a;
            a := 2 * a; b := b / 2;
    end;
end;

procedure deleni;
var w;
begin r:=x; q := 0; w:= y;
    while w <= r do w := 2*w;
    while w > y do
    begin q:=2*q; w:=w/2;
        if w <= r  then
        begin r:=r-w; q:=q+1;
        end;
    end;
end;

procedure gcd;
```

0	JMP	0	13
1	JMP	0	2
2	INT	0	3
3	LOD	1	3
4	LIT	0	1
5	OPR	0	2
6	STO	1	3
7	LOD	1	3
8	LIT	0	2
9	OPR	0	13
10	JMC	0	12
11	CAL	1	2
12	RET	0	0
13	INT	0	4
14	LIT	0	0
15	STO	0	3
16	CAL	0	2
17	RET	0	0

ParserTerm>Welcome to PL0 parser :)

ParserTerm>Debugger successfuly connected

Napojení na debugger



Testování

Boolean | Load example | Run tests

```
const m=7, n=85;
var x, y, z, q, r;

procedure nasobeni;
var a, b;
begin a := x; b:= y; z:=0;
    while b > 0 do
    begin
        if odd b then z := z + a;
            a := 2 * a; b := b / 2;
    end;
end;

procedure deleni;
var w;
begin r:=x; q := 0; w:= y;
    while w <= r do w := 2*w;
    while w > y do
    begin q:=2*q; w:=w/2;
        if w <= r   then
        begin r:=r-w; q:=q+1;
        end;
    end;
end;

procedure gcd;
```

```
3
2
3
3
1
2
3
3
2
3
12
 2
 0
 4
 0
 3
 2
 0
```

ParserTerm>Welcome to PL0 parser :)
ParserTerm>Debugger successfuly connected

Test results

- intArithmetics.js
- intConditions.js
- loop.js
- parallelAssignment.js
- proceduresParse.js
- return.js
- ternary.js
- negative/invalidKeyName.js
- negative/notDeclaredVariable.js
- negative/missingEndDot.js
- negative/unknownOperator.js
- negative/missingSemicolon.js

Tests run: 19
Successful tests: 18
Failed tests: 1

Close

Dark mode :)

Boolean | Load example | Run tests

Assignment statement failed. Identifier: proceduure not found

View Problem (Alt+F8) No quick fixes available

```
procedure nasobeni;
var a, b;
begin a := x; b:= y; z:=0;
    while b > 0 do
    begin
        if odd b then z := z + a;
            a := 2 * a; b := b / 2;
    end;
end;

procedure deleni;
var w;
begin r:=x; q := 0; w:= y;
    while w <= r do w := 2*w;
    while w > y do
    begin q:=2*q; w:=w/2;
        if w <= r   then
        begin r:=r-w; q:=q+1;
        end;
    end;
end;

procedure gcd;
```

0	JMP	0	13
1	JMP	0	2
2	INT	0	3
3	LOD	1	3
4	LIT	0	1
5	OPR	0	2
6	STO	1	3
7	LOD	1	3
8	LIT	0	2
9	OPR	0	13
10	JMC	0	12
11	CAL	1	2
12	RET	0	0
13	INT	0	4
14	LIT	0	0
15	STO	0	3
16	CAL	0	2
17	RET	0	0

```
ParserTerm>Welcome to PL0 parser :)
ParserTerm>Debugger successfuly connected
```

Děkujeme za pozornost