```
diff --git a/task/lab4/check.ml b/task_after/lab4/check.ml
index 3ffd132..0116bba 100644
--- a/task/lab4/check.ml
+++ b/task_after/lab4/check.ml
@@ -90,7 +90,7 @@ let try_binop w v1 v2 =
 (* |has_value| -- check if object is suitable for use in expressions *)
 let has_value d =
   match d.d_kind with
-    ConstDef _ | VarDef | CParamDef | VParamDef | StringDef -> true
+    ConstDef _ | VarDef | CParamDef | VParamDef | NParamDef | StringDef -> true
   | _ -> false

(* |check_var| -- check that expression denotes a variable *)
@@ -100,7 +100,7 @@ let rec check_var e addressible =
       let d = get_def x in
       begin
        match d.d_kind with
-          VarDef | VParamDef | CParamDef ->
+          VarDef | VParamDef | CParamDef | NParamDef ->
            d.d_mem <- d.d_mem || addressible
         | _ ->
           sem_error "$ is not a variable" [fId x.x_name]

@@ -198,12 +205,16 @@ and check_args formals args env =
 (* |check_arg| -- check one (formal, actual) parameter pair *)
 and check_arg formal arg env =
   match formal.d_kind with
-    CParamDef | VParamDef ->
+    CParamDef | VParamDef | NParamDef ->
       let t1 = check_expr arg env in
       if not (same_type formal.d_type t1) then
        sem_error "argument has wrong type" [];
       if formal.d_kind = VParamDef then
-        check_var arg true
+        check_var arg true;
+      if formal.d_kind = NParamDef then
+        if not (same_type formal.d_type integer) then
+          sem_error "by-name parameters must be integers" [];
+        check_var arg true
   | PParamDef ->
       let pf = get_proc formal.d_type in
       let x = (match arg.e_guts with Variable x -> x
@@ -378,6 +389,7 @@ let rec check_stmt s env alloc =
       check_dupcases vs;
       check_stmt deflt env alloc

+  | ContinueStmt -> ()

 (* TYPES AND DECLARATIONS *)
```

```
@@ -418,7 +430,7 @@ let local_alloc size nreg d =
 let param_alloc pcount d =
   let s = param_rep.r_size in
   match d.d_kind with
-    CParamDef | VParamDef ->
+    CParamDef | VParamDef | NParamDef ->
       d.d_addr <- Local (param_base + s * !pcount);
       incr pcount
   | PParamDef ->
@@ -434,7 +446,7 @@ let global_alloc d =
 let do_alloc alloc ds =
   let h d =
     match d.d_kind with
-      VarDef | CParamDef | VParamDef | FieldDef | PParamDef ->
+      VarDef | CParamDef | VParamDef | NParamDef | FieldDef | PParamDef ->
         alloc d
     | _ -> () in
   List.iter h ds

diff --git a/task/lab4/lexer.mll b/task_after/lab4/lexer.mll
index 71e8e0a..85f1329 100644
--- a/task/lab4/lexer.mll
+++ b/task_after/lab4/lexer.mll
@@ -22,7 +22,7 @@ let symtable =
     ("repeat", REPEAT); ("until", UNTIL); ("for", FOR);
     ("elsif", ELSIF); ("case", CASE);
     ("and", MULOP And); ("div", MULOP Div); ("or", ADDOP Or);
-    ("not", NOT); ("mod", MULOP Mod) ]
+    ("not", NOT); ("mod", MULOP Mod); ("continue", CONTINUE) ]

 let lookup s =
   try Hashtbl.find symtable s with
@@ -82,6 +82,7 @@ rule token =
   | "|"            { VBAR }
   | ":"            { COLON }
   | "^"            { ARROW }
+  | "=>"           { EQUIV }
   | "("            { LPAR }
   | ")"            { RPAR }
   | ","            { COMMA }

diff --git a/task/lab4/parser.mly b/task_after/lab4/parser.mly
index 21797ce..a67c7ce 100644
--- a/task/lab4/parser.mly
+++ b/task_after/lab4/parser.mly
@@ -15,7 +15,7 @@ open Tree

 /* punctuation */
```

```
 %token              SEMI DOT COLON LPAR RPAR COMMA SUB BUS
-%token              EQUAL MINUS ASSIGN VBAR ARROW
+%token              EQUAL MINUS ASSIGN VBAR ARROW EQUIV
 %token              BADTOK IMPOSSIBLE

 /* keywords */
@@ -23,6 +23,7 @@ open Tree
 %token              PROC RECORD RETURN THEN TO TYPE
 %token              VAR WHILE NOT POINTER NIL
 %token              REPEAT UNTIL FOR ELSIF CASE
+%token              CONTINUE

 %type <Tree.program>    program
 %start              program
@@ -87,6 +88,7 @@ formal_decls :
 formal_decl :
    ident_list COLON typexpr          { VarDecl (CParamDef, $1, $3) }
  | VAR ident_list COLON typexpr       { VarDecl (VParamDef, $2, $4) }
+ | EQUIV ident_list COLON typexpr     { VarDecl (NParamDef, $2, $4) }
  | proc_heading                       { PParamDecl $1 } ;

 return_type :
@@ -119,7 +121,8 @@ stmt1 :
  | FOR name ASSIGN expr TO expr DO stmts END
                                { let v = makeExpr (Variable $2) in
                                  ForStmt (v, $4, $6, $8, ref None) }
-| CASE expr OF arms else_part END    { CaseStmt ($2, $4, $5) } ;
+| CASE expr OF arms else_part END    { CaseStmt ($2, $4, $5) }
+| CONTINUE                           { ContinueStmt };

diff --git a/task/lab4/tgen.ml b/task_after/lab4/tgen.ml
index 9c4195a..a6eea85 100644
--- a/task/lab4/tgen.ml
+++ b/task_after/lab4/tgen.ml
(* |addr_size| -- size of address *)
@@ -64,6 +65,9 @@ let gen_closure d =
     ProcDef ->
      (address d,
       if d.d_level = 0 then <CONST 0> else schain (!level - d.d_level))
+  | NParamDef ->
+    (<LOADW, address d>,
+     <LOADW, <OFFSET, address d, <CONST addr_size>>>)
   | PParamDef ->
     (<LOADW, address d>,
      <LOADW, <OFFSET, address d, <CONST addr_size>>>)
@@ -94,12 +98,14 @@ let rec gen_addr v =
           address d
         | VParamDef ->
           <LOADW, address d>
```

```
+         | NParamDef ->
+           <LOADW, address d>
@@ -166,6 +171,15 @@ and gen_arg f a =
         [gen_addr a]
     | VParamDef ->
       [gen_addr a]
+   | NParamDef ->
+     begin
+       match a.e_guts with
+         Variable x ->
+           let(proc,env) = gen_closure f in [proc;env]
+       | _ -> [gen_expr a]
+     end
    | PParamDef ->
       begin
        match a.e_guts with
@@ -250,12 +264,12 @@ let gen_jtable sel table0 deflab =
  end

 (* |gen_stmt| -- generate code for a statement *)
-let rec gen_stmt s =
+let rec gen_stmt j_lab s =
   let code =
    match s.s_guts with
      Skip -> <NOP>

-    | Seq ss -> <SEQ, @(List.map gen_stmt ss)>
+    | Seq ss -> <SEQ, @(List.map (gen_stmt j_lab) ss)>

     | Assign (v, e) ->
       if scalar v.e_type || is_pointer v.e_type then begin
@@ -280,10 +294,10 @@ let rec gen_stmt s =
         <SEQ,
          gen_cond test l1 l2,
          <LABEL l1>,
-         gen_stmt thenpt,
+         gen_stmt j_lab thenpt,
          <JUMP l3>,
          <LABEL l2>,
-         gen_stmt elsept,
+         gen_stmt j_lab elsept,
          <LABEL l3>>

     | WhileStmt (test, body) ->
@@ -294,15 +308,16 @@ let rec gen_stmt s =
         <LABEL l1>,
         gen_cond test l2 l3,
         <LABEL l2>,
-        gen_stmt body,
```

```
+          gen_stmt l1 body,
           <JUMP l1>,
           <LABEL l3>>

      | RepeatStmt (body, test) ->
-        let l1 = label () and l2 = label () in
+        let l1 = label () and l2 = label () and l3 = label () in
         <SEQ,
          <LABEL l1>,
-         gen_stmt body,
+         gen_stmt l3 body,
+         <LABEL l3>,
          gen_cond test l2 l1,
          <LABEL l2>>

@@ -310,16 +325,21 @@ let rec gen_stmt s =
         (* Use previously allocated temp variable to store upper bound.
            We could avoid this if the upper bound is constant. *)
         let tmp = match !upb with Some d -> d | _ -> failwith "for" in
-        let l1 = label () and l2 = label () in
+        let l1 = label () and l2 = label () and l3 = label () in
         <SEQ,
          <STOREW, gen_expr lo, gen_addr var>,
          <STOREW, gen_expr hi, address tmp>,
          <LABEL l1>,
          <JUMPC (Gt, l2), gen_expr var, <LOADW, address tmp>>,
-         gen_stmt body,
+         gen_stmt l3 body,
+         <LABEL l3>,
          <STOREW, <BINOP Plus, gen_expr var, <CONST 1>>, gen_addr var>,
          <JUMP l1>,
          <LABEL l2>>
+
+    | ContinueStmt ->
+        if j_lab = !retlab then failwith "continue statement outside of loop"
+        else <JUMP j_lab>

      | CaseStmt (sel, arms, deflt) ->
        (* Use one jump table, and hope it is reasonably compact *)
@@ -330,13 +350,13 @@ let rec gen_stmt s =
         let gen_case lab (v, body) =
          <SEQ,
           <LABEL lab>,
-          gen_stmt body,
+          gen_stmt j_lab body,
           <JUMP donelab>> in
         <SEQ,
          gen_jtable (gen_expr sel) table deflab,
          <SEQ, @(List.map2 gen_case labs arms)>,
```

```
          <LABEL deflab>,
-         gen_stmt deflt,
+         gen_stmt j_lab deflt,
          <LABEL donelab>> in

   (* Label the code with a line number *)
@@ -370,7 +390,7 @@ let do_proc lab lev nargs (Block (_, body, fsize, nregv)) =
  level := lev+1;
  retlab := label ();
  let code0 =
-   show "Initial code" (Optree.canon <SEQ, gen_stmt body, <LABEL !retlab>>) in
+   show "Initial code" (Optree.canon <SEQ, gen_stmt !retlab body, <LABEL !retlab>>) in
  Regs.init ();
  let code1 = if !optlevel < 1 then code0 else
     show "After simplification" (Jumpopt.optimise (Simp.optimise code0)) in

diff --git a/task/lab4/tree.ml b/task_after/lab4/tree.ml
index ad91ade..0eea580 100644
--- a/task/lab4/tree.ml
+++ b/task_after/lab4/tree.ml
@@ -39,6 +39,7 @@ and stmt_guts =
  | RepeatStmt of stmt * expr
  | ForStmt of expr * expr * expr * stmt * def option ref
  | CaseStmt of expr * (expr * stmt) list * stmt
+ | ContinueStmt

@@ -124,6 +126,7 @@ and fKind =
     VarDef -> fStr "VAR"
   | CParamDef -> fStr "PARAM"
   | VParamDef -> fStr "VPARAM"
+   | NParamDef -> fStr "NPARAM"
   | FieldDef -> fStr "FIELD"
   | _ -> fStr "???"

@@ -150,6 +153,7 @@ and fStmt s =
   | CaseStmt (sel, arms, deflt) ->
     let fArm (lab, body) = fMeta "($ $)" [fExpr lab; fStmt body] in
     fMeta "(CASE $ $ $)" [fExpr sel; fList(fArm) arms; fStmt deflt]
+   | ContinueStmt -> fStr "(CONTINUE)"

diff --git a/task/lab4/tree.mli b/task_after/lab4/tree.mli
index ca47363..71580eb 100644
--- a/task/lab4/tree.mli
+++ b/task_after/lab4/tree.mli
@@ -54,6 +54,7 @@ and stmt_guts =
  | RepeatStmt of stmt * expr
  | ForStmt of expr * expr * expr * stmt * def option ref
  | CaseStmt of expr * (expr * stmt) list * stmt
+ | ContinueStmt
```