ADF 2*x* & PRO 2*x* **Übungen zu Fortgeschrittenen Algorithmen & Datenstrukturen und OOP** **SS 21, Übung 6**

Abgabetermin: Mi, 19. 05. 2021

☐ **Gr. 1,** Dr. S. Wagner **Name** Angelos Angelis **Aufwand in h** 8

☒ **Gr. 2,** Dr. D. Auer

☐ **Gr. 3,** Dr. G. Kronberger **Punkte** _____ **Kurzzeichen Tutor / Übungsleiter*in** _____ / _____

1. MidiPascal (24 Punkte)

Wesentliche Sprachkonstrukte, die MiniPascal fehlen, sind Verzweigungen und Schleifen. Also erweitern wir MiniPascal um die binäre Verzweigung (*IF*-Anweisung), die Abweisschleife (*WHILE*-Schleife) sowie die Verbundanweisung (*BEGIN ... END*) – und taufen die neue Sprache MidiPascal.

Nachdem wir mit dem Datentyp *INTEGER* und ohne Erweiterungen der Ausdrücke um relationale Operatoren auskommen wollen, verwenden wir für Bedingungen in Verzweigungen und Schleifen *INTEGER*-Variablen mit der Semantik, dass jeder Wert ungleich 0 als *TRUE* und (nur) der Wert 0 als *FALSE* interpretiert wird – so wie das z. B. in der Programmiersprache C definiert ist. Folgende Tabelle zeigt zur Verdeutlichung eine Abbildung von MidiPascal auf (vollständiges) Pascal:

MidiPascal	(vollständiges) Pascal
`VAR x: INTEGER;`	`VAR x: INTEGER;`
`IF x THEN ...`	**`IF x <> 0 THEN ...`**
`WHILE x DO ...`	**`WHILE x <> 0 DO ...`**

Mit diesen Spracherweiterungen könnte man dann z. B. ein MidiPascal-Programm schreiben, das für eine eingegebene Zahl *n* die Fakultät *f* = *n*! iterativ berechnet und diese ausgibt. Siehe Quelltextstück rechts.

```
f := n; n := n - 1;
WHILE n DO BEGIN
  f := n * f;
  n := n - 1;
END;
WRITE(f);
```

Damit diese neuen Sprachkonstrukte im Compiler umgesetzt werden können, sind zwei neue Bytecode-Befehle notwendig. Folgende Tabelle erläutert diese beiden Befehle:

Bytecode-Befehl	Semantik
`Jmp addr`	Springe an die Codeadresse *addr*
`JmpZ addr`	Hole oberstes Element vom Stapel und wenn dieses 0 (*zero*) ist, springe nach *addr*

Nun muss man nur noch klären, welche Bytecodestücke für die einzelnen, neuen MidiPascal-Anweisungen zu erzeugen sind. Folgende Tabelle stellt die notwendigen Transformationen anhand von Mustern dar:

MidiPascal	Bytecode (mit fiktiven Adressen)

`IF x THEN BEGIN`
`  then stats`
`END;`
`...`
	`1 LoadVal x`
	`4 JmpZ 99`
	`... code for then stats`
	
	`99 ...`

Lösungsidee:
An sich muss man hier nur den Minipascal Compiler so erweitern dass er auch If Anweisungen und while Schleifen erkennen und interpretieren kann. Wobei dann auch „do“ und „then“ interpretiert werden muss. Die Semantik wird auch dementsprechend erweitert

Test:

```
PROGRAM Test;
  VAR
    n, f : INTEGER;
BEGIN
  READ(n);
  f := 1;
  n := n - 1;
  WHILE n DO BEGIN
    f := 1 + f;
    n := n - 1;
  END;
  WRITE(f);
END.
```

Ausgabe:

```
MidiPascal source file > test.mp
parsing completed: success
code interpretation started ...
var@0 > 100
100
... code interpretation ended
```

Quellcode:

```
PROGRAM MPC;
  USES
    MPScanner, MPI, CodeDef, CodeGen, CodeInt;
```

```
  VAR
    inputFilePath: STRING;
    ca: CodeArray;
    ok: BOOLEAN;

BEGIN (* MPC *)
  IF (ParamCount = 1) THEN BEGIN
    inputFilePath := ParamStr(1);
  END ELSE BEGIN
    Write('MidiPascal source file > ');
    ReadLn(inputFilePath);
  END; (* IF *)

  InitLex(inputFilePath);

  S;

  IF (success) THEN BEGIN
    WriteLn('parsing completed: success');
    GetCode(ca);
    StoreCode(inputFilePath + 'c', ca);

    LoadCode(inputFilePath + 'c', ca, ok);
    IF (NOT ok) THEN BEGIN
      WriteLn('ERROR: cannot open mpc file');
      HALT;
    END; (* IF *)
    InterpretCode(ca);
  END ELSE BEGIN
    WriteLn('parsing failed. ERROR at position (', syLineNr, ',', syColNr, ', ',
sy, ')');
  END; (* IF *)
END. (* MPC *)


UNIT MPI;

INTERFACE
  VAR
    success: BOOLEAN;

  PROCEDURE S;

IMPLEMENTATION
  USES
```

```
    MPScanner, SymTab, CodeDef, CodeGen;

  FUNCTION SyIsNot(expected: Symbol): BOOLEAN;
  BEGIN (* SyIsNot *)
    IF (sy <> expected) THEN BEGIN
      success := FALSE;
    END; (* IF *)
    SyIsNot := NOT success;
  END; (* SyIsNot *)

  PROCEDURE SemErr(msg: STRING);
  BEGIN (* SemErr *)
    WriteLn(' ### ERROR: ', msg);
    success := FALSE;
  END; (* SemErr *)

  PROCEDURE MP; FORWARD;
  PROCEDURE VarDecl; FORWARD;
  PROCEDURE StatSeq; FORWARD;
  PROCEDURE Stat; FORWARD;
  PROCEDURE Expr; FORWARD;
  PROCEDURE Term; FORWARD;
  PROCEDURE Fact; FORWARD;

  PROCEDURE S;
  BEGIN (* S *)
    success := TRUE;
    MP; IF (NOT success) THEN Exit;
    IF (SyIsNot(eofSy)) THEN Exit;
  END; (* S *)

  PROCEDURE MP;
  BEGIN (* MP *)
    (*SEM*)
      InitSymbolTable;
      InitCodeGenerator;
    (*ENDSEM*)
    IF (SyIsNot(programSy)) THEN Exit;
    NewSy;
    IF (SyIsNot(ident)) THEN Exit;
    NewSy;
    IF (SyIsNot(semiColonSy)) THEN Exit;
    NewSy;

    IF (sy = varSy) THEN BEGIN
```

```
      VarDecl; IF (NOT success) THEN Exit;
    END; (* IF *)

    IF (SyIsNot(beginSy)) THEN Exit;
    NewSy;

    StatSeq; IF (NOT success) THEN Exit;
    (*SEM*) Emit1(EndOpc); (*ENDSEM*)

    IF (SyIsNot(endSy)) THEN Exit;
    NewSy;
    IF (SyIsNot(dotSy)) THEN Exit;
    NewSy;
  END; (* MP *)

  PROCEDURE VarDecl;
    VAR
      ok: BOOLEAN;
    BEGIN (* VarDecl *)
      IF (SyIsNot(varSy)) THEN Exit;
      NewSy;
      IF (SyIsNot(ident)) THEN Exit;
      (*SEM*) DeclVar(identStr, ok); (*ENDSEM*)
      NewSy;

      WHILE (sy = commaSy) DO BEGIN
        NewSy;
        IF (SyIsNot(ident)) THEN Exit;
        (*SEM*)
          DeclVar(identStr, ok);
          IF (NOT ok) THEN SemErr('variable declared multiple times');
        (*ENDSEM*)
        NewSy;
      END; (* WHILE *)

      IF (SyIsNot(colonSy)) THEN Exit;
      NewSy;
      IF (SyIsNot(integerSy)) THEN Exit;
      NewSy;
      IF (SyIsNot(semiColonSy)) THEN Exit;
      NewSy;
  END; (* VarDecl *)

  PROCEDURE StatSeq;
  BEGIN (* StatSeq *)
```

```
    Stat; IF (NOT success) THEN Exit;
    WHILE (sy = semiColonSy) DO BEGIN
      NewSy;
      Stat; IF (NOT success) THEN Exit;
    END; (* WHILE *)
  END; (* StatSeq *)

  PROCEDURE Stat;
    VAR
      destId: STRING;
      address1, address2: INTEGER;
  BEGIN (* Stat *)
    CASE sy OF
      ident:   BEGIN
                (*SEM*)
                  destId := identStr;
                  IF (NOT IsDecl(destId)) THEN SemErr('variable not declared')
                  ELSE Emit2(LoadAddrOpc, AddrOf(destId));
                (*ENDSEM*)
                 NewSy;
                 IF (SyIsNot(assignSy)) THEN Exit;
                 NewSy;
                 Expr; IF (NOT success) THEN Exit;
                 (*SEM*) IF (IsDecl(destId)) THEN Emit1(StoreOpc); (*ENDSEM*)
               END;
      readSy:  BEGIN
                 NewSy;
                 IF (SyIsNot(leftParSy)) THEN Exit;
                 NewSy;
                 IF (SyIsNot(ident)) THEN Exit;
                 (*SEM*)
                   IF (NOT IsDecl(identStr)) THEN SemErr('variable not declared')
                   ELSE Emit2(ReadOpc, AddrOf(identStr));
                 (*ENDSEM*)
                 NewSy;
                 IF (SyIsNot(rightParSy)) THEN Exit;
                 NewSy;
               END;
      writeSy: BEGIN
                 NewSy;
                 IF (SyIsNot(leftParSy)) THEN Exit;
                 NewSy;
                 Expr; IF (NOT success) THEN Exit;
                 (*SEM*) Emit1(WriteOpc); (*ENDSEM*)
                 IF (SyIsNot(rightParSy)) THEN Exit;
```

```
            NewSy;
          END;
beginSy:  BEGIN
            NewSy;
            StatSeq; IF (NOT success) THEN Exit;
            IF (SyIsNot(endSy)) THEN Exit;
          END;
ifSy:     BEGIN
            NewSy; IF (SyIsNot(ident)) THEN Exit;
            (*SEM*)
             IF (NOT IsDecl(identStr)) THEN SemErr('variable not declared');
             Emit2(LoadValOpc, AddrOf(identStr));
             Emit2(JmpZOpc, 0);
             address1 := CurAddr - 2;
            (*ENDSEM*)
            NewSy; IF (SyIsNot(thenSy)) THEN Exit;
            NewSy;
            Stat; IF (NOT success) THEN Exit;
            NewSy;
            IF (SyIsNot(elseSy)) THEN Exit
            ELSE BEGIN
            (*SEM*)
             Emit2(JmpOpc, 0);
             FixUp(address1, CurAddr);
             address1 := CurAddr - 2;
            (*ENDSEM*)
             NewSy;
             Stat; IF (NOT success) THEN Exit;
            END; (* ELSE *)
            NewSy; IF (SyIsNot(semiColonSy)) THEN Exit;
            (*SEM*)
             FixUp(address1, CurAddr);
            (*ENDSEM*)
          END;
whileSy:  BEGIN
            NewSy; IF (SyIsNot(ident)) THEN Exit;
            (*SEM*)
             IF (NOT IsDecl(identStr)) THEN SemErr('variable not declared');
             address1 := CurAddr;
             Emit2(LoadValOpc, AddrOf(identStr));
             Emit2(JmpZOpc, 0);
             address2 := CurAddr - 2;
            (*ENDSEM*)
            NewSy; IF (SyIsNot(doSy)) THEN Exit;
            NewSy;
```

```
                Stat; IF (NOT success) THEN Exit;
                (*SEM*)
                 Emit2(JmpOpc, address1);
                 FixUp(address2, CurAddr);
                (*ENDSEM*)
                NewSy; IF (SyIsNot(semiColonSy)) THEN Exit;
              END;
    END;
  END; (* Stat *)

  PROCEDURE Expr;
  BEGIN (* Expr *)
    Term; IF (NOT success) THEN Exit;
    WHILE (sy = plusSy) OR (sy = minusSy) DO BEGIN
      CASE sy OF
        plusSy: BEGIN
          NewSy;
          Term; IF (NOT success) THEN Exit;
          (*SEM*) Emit1(AddOpc); (*ENDSEM*)
        END; (* plusSy *)
        minusSy: BEGIN
          NewSy;
          Term; IF (NOT success) THEN Exit;
          (*SEM*) Emit1(SubOpc); (*ENDSEM*)
        END; (* minusSy *)
      END; (* CASE *)
    END; (* WHILE *)
  END; (* Expr *)

  PROCEDURE Term;
  BEGIN (* Term *)
    Fact; IF (NOT success) THEN Exit;
    WHILE (sy = timesSy) OR (sy = divSy) DO BEGIN
      CASE sy OF
        timesSy: BEGIN
          NewSy;
          Fact; IF (NOT success) THEN Exit;
          (*SEM*) Emit1(MulOpc); (*ENDSEM*)
        END; (* timesSy *)
        divSy: BEGIN
          NewSy;
          Fact; IF (NOT success) THEN Exit;
          (*SEM*) Emit1(DivOpc); (*ENDSEM*)
        END; (* divSy *)
      END; (* CASE *)
```

```
    END; (* WHILE *)
  END; (* Term *)

  PROCEDURE Fact;
  BEGIN (* Fact *)
    CASE sy OF
      ident: BEGIN
        (*SEM*)
          IF (NOT IsDecl(identStr)) THEN SemErr('variable not declared')
          ELSE Emit2(LoadValOpc, AddrOf(identStr));
        (*ENDSEM*)
        NewSy;
      END; (* ident *)
      number: BEGIN
        (*SEM*) Emit2(LoadConstOpc, numberVal); (*ENDSEM*)
        NewSy;
      END; (* number *)
      leftParSy: BEGIN
        NewSy;
        Expr; IF (NOT success) THEN Exit;
        IF (SyIsNot(rightParSy)) THEN Exit;
        NewSy;
      END; (* leftParSy *)
      ELSE BEGIN
        success := FALSE;
        Exit;
      END; (* ELSE *)
    END; (* CASE *)
  END; (* Fact *)

END. (* MPI *)


UNIT MPScanner;

INTERFACE
  TYPE
    Symbol = (errSy, eofSy,
              programSy, varSy, beginSy, endSy,
              readSy, writeSy, integerSy,
              semicolonSy, colonSy, commaSy, dotSy,
              assignSy,
              plusSy, minusSy, timesSy, divSy,
              leftParSy, rightParSy,
              ifSy, thenSy, elseSy, whileSy, doSy,
```

```
              ident, number);

  VAR
    sy: Symbol;                   (* current symbol *)
    syLineNr, syColNr: INTEGER;   (* line/column number of current symbol *)
    numberVal: INTEGER;           (* value if sy = number *)
    identStr: STRING;             (* name if sy = ident *)

  PROCEDURE InitLex(inputFilePath: STRING);
  PROCEDURE NewSy;

IMPLEMENTATION
  CONST
    EOF_CH = Chr(26);
    TAB_CH = Chr(9);

  VAR
    inputFile: TEXT;
    line: STRING;
    lineNr, linePos: INTEGER;
    ch: CHAR;

  PROCEDURE NewCh; FORWARD;

  PROCEDURE InitLex(inputFilePath: STRING);
  BEGIN (* InitLex *)
    Assign(inputFile, inputFilePath);
    Reset(inputFile);
    line := '';
    lineNr := 0;
    linePos := 0;
    NewCh;
    NewSy;
  END; (* InitLex *)

  PROCEDURE NewSy;
  BEGIN (* NewSy *)
    WHILE ((ch = ' ') OR (ch = TAB_CH)) DO BEGIN  (* skip whitespaces *)
      NewCh;
    END; (* WHILE *)
    syLineNr := lineNr;
    syColNr := linePos;

    CASE ch OF
      EOF_CH:    BEGIN sy := eofSy; END;
```

```
'+':       BEGIN sy := plusSy; NewCh; END;
'*':       BEGIN sy := timesSy; NewCh; END;
'-':       BEGIN sy := minusSy; NewCh; END;
'/':       BEGIN sy := divSy; NewCh; END;
'(':       BEGIN sy := leftParSy; NewCh; END;
')':       BEGIN sy := rightParSy; NewCh; END;
';':       BEGIN sy := semicolonSy; NewCh; END;
',':       BEGIN sy := commaSy; NewCh; END;
'.':       BEGIN sy := dotSy; NewCh; END;
':':       BEGIN
             NewCh;
             IF (ch = '=') THEN BEGIN
               sy := assignSy;
               NewCh;
             END ELSE BEGIN
               sy := colonSy;
            END; (* IF *)
           END;
'0'..'9':  BEGIN
             sy := number;
             numberVal := 0;
             WHILE (ch >= '0') AND (ch <= '9') DO BEGIN
               numberVal := numberVal * 10 + Ord(ch) - Ord('0');
               NewCh;
             END; (* WHILE *)
           END;
'a'..'z',
'A'..'Z',
'_'      : BEGIN
             identStr := '';
             WHILE (ch IN ['a'..'z', 'A'..'Z', '_', '0'..'9']) DO BEGIN
               identStr := identStr + ch;
               NewCh;
             END; (* WHILE *)
             identStr := UpCase(identStr);

             IF (identStr = 'PROGRAM') THEN BEGIN
               sy := programSy;
             END ELSE IF (identStr = 'BEGIN') THEN BEGIN
               sy := beginSy;
             END ELSE IF (identStr = 'END') THEN BEGIN
               sy := endSy;
             END ELSE IF (identStr = 'VAR') THEN BEGIN
               sy := varSy;
             END ELSE IF (identStr = 'INTEGER') THEN BEGIN
```

```
                    sy := integerSy;
                  END ELSE IF (identStr = 'READ') THEN BEGIN
                    sy := readSy;
                  END ELSE IF (identStr = 'WRITE') THEN BEGIN
                    sy := writeSy;
                  END ELSE IF (identStr = 'IF') THEN BEGIN
                    sy := ifSy;
                  END ELSE IF (identStr = 'THEN') THEN BEGIN
                    sy := thenSy;
                  END ELSE IF (identStr = 'ELSE') THEN BEGIN
                    sy := elseSy;
                  END ELSE IF (identStr = 'WHILE') THEN BEGIN
                    sy := whileSy;
                  END ELSE IF (identStr = 'DO') THEN BEGIN
                    sy := doSy;
                  END ELSE BEGIN
                    sy := ident;
                  END; (* IF *)
                END;
      ELSE      BEGIN sy := errSy; END;
    END; (* CASE *)
  END; (* NewSy *)

  PROCEDURE NewCh;
  BEGIN (* NewCh *)
    Inc(linePos);
    IF (linePos > Length(line)) THEN BEGIN
      IF (NOT Eof(inputFile)) THEN BEGIN
        ReadLn(inputFile, line);
        Inc(lineNr);
        linePos := 0;
        ch := ' ';
      END ELSE BEGIN
        ch := EOF_CH;
        line := '';
        linePos := 0;
        Close(inputFile);
      END; (* IF *)
    END ELSE BEGIN
      ch := line[linePos];
    END; (* IF *)
  END; (* NewCh *)
END. (* MPScanner *)
```