MIKADOS

Pascal Link Program - User´s Guide

Dansk Data Elektronik ApS
23 March 1981

Author: Claus Tøndering

Table of contents

1. Introduction.

In many cases when a programmer designs a large program, the need arises for a means to link seperately compiled modules together. The advantages of this are, first, there is no need to recompile everything after small corrections; second, the modules to be compiled will be smaller, so that a program which normally would be to big for the compiler to handle would be reduced to an acceptable size.

The PEX and PLINK programs are designed to equip the programmer with tools to link seperately compiled Pascal modules. The programs operate directly with the P-code files, so that no modification of the compiler is necessary to use them.

This manual describes PEX and PLINK and how to use them.

It is required that the user be familiar with Pascal´s SEGMENT facility.

2. Basic Concepts.

The PEX and PLINK programs provide the user with a means to link together seperately compiled SEGMENT procedures and functions of a Pascal program. They cannot be used to link together procedures and functions that are not segments.

The principles of operation are:

The P-code file of a compiled Pascal program is a direct access file with a record length of 128 bytes. The first of these records is a segment descriptor. It contains information about the length of each segment of the program and the record number where the P-code segment starts.

We wish to create a Pascal program where one of the segments is taken from a seperate source file. This is done in the following manner:

The source text of the main program must contain a SEGMENT PROCEDURE or a SEGMENT FUNCTION declaration of the external segment. However, the body of that procedure is left empty, that is, the BEGIN and END are present, but no code is included between BEGIN and END.

The source text of the module containing the code for the segment procedure is structured in the same manner as the main program, except that all unnecessary declarations are omitted, the body of the main program is omitted, but the code of the segment procedure is present.

This will be described in greater detail in chapter 4 of this manual.

The two modules are compiled independently. The PEX program is then executed. This program will edit the P-code file of the

main program, so that the segment descriptor will contain information that the relevant segment is not present in the P-code. Execution of the PLINK program will then merge the P-code file of the main program together with the P-code file of the external segment procedure, so that the relevant segment of the main program is replaced by the seperately compiled segment procedure.

3. An Example.

The link between the seperately compiled modules is established not by means of common identifiers, but rather with the location of the declarations of the various identifiers in the modules.

The structure of the compilation units is most easily described with an example.

Let us take the following program, which is written here in the conventional manner with no external segments:

```
PROGRAM ALFA;

CONST VALUE=12.86;

VAR R,S,T: REAL;
    I: INTEGER;

SEGMENT PROCEDURE BETA(J: INTEGER);
VAR K: INTEGER;
BEGIN
  K:=I*J;
  WRITELN(´I,J,K are: ´,I:5,J:5,K:5);
END;

BEGIN
  WRITE(´Write a real number and an integer: ´);
  READ(R,I);
  S:=VALUE*I;
  T:=S*R;
  BETA(3);
  WRITELN(´T=´,T);
END.
```

We wish to remove the segment procedure BETA from the above program, and make it a seperately compiled procedure.

The main program is restructured:

```
PROGRAM ALFA;

CONST VALUE=12.86;

VAR R,S,T: REAL;
    I: INTEGER;

SEGMENT PROCEDURE BETA(J: INTEGER);
BEGIN
  (* External segment *)
END;

BEGIN
  WRITE(´Write a real number and an integer: ´);
  READ(R,I);
  S:=VALUE*I;
  T:=S*R;
  BETA(3);
  WRITELN(´T=´,T);
END.
```

The code and the local declaration of BETA have now been removed. The comment in the body of BETA is, of course, not required, but it is a good programming practice to have it there.

This module is compiled in the usual manner.

The source text of the segment procedure BETA will look like this:

```
PROGRAM ALFAX;

VAR DUMMY: PACKED ARRAY (1..24) OF CHAR;
    I: INTEGER;

(* EXTERNAL *) SEGMENT PROCEDURE BETA(J: INTEGER);
VAR K: INTEGER;
BEGIN
  K:=I*J;
  WRITELN(´I,J,K are: ´,I:5,J:5,K:5);
END;

BEGIN
END.
```

Here the body of the main program has been removed. The unnecessary declarations of R, S, and T, which are not used by BETA, are removed. The declaration of the global variable I is retained, so that BETA may refer to it. However, since the link between the I used in this module and the I used in the main module is established through the position of the declaration of I, it is required that the declaration of I be offset by the same number of bytes as in the main module. Therefore, the declaration of the variable DUMMY is included. The type of this variable is such that it takes up the same number of bytes as the variables R, S, and T, that is, 3 times 8 bytes.

Of course, the declarations of R, S, and T could have been retained, but no compiler memory would have been saved. Reducing the number of declared variables reduces the amount of compiler memory required.

Here again the (* EXTERNAL *) comment preceding the segment procedure declaration is not required, but is good programming practice.

This module is compiled in the usual manner.

Let us now link these two modules together.

We assume that the P-code of the main module is in file ALFA:P2 and the P-code of the module containing BETA is in file ALFAX:P2.

We use the PEX (P-Pascal EX-External) program to edit the P-code file ALFA. We wish to indicate in this file that segment number 1 is external, that is, a seperately compiled module. The numbering of segments is discussed in chapter 4 of this manual.

PEX is executed in the following manner:

```
>PEX
Write name of file: ALFA:P2
Write number of segment to be made external: 1
Write number of segment to be made external: <escape>
PEX terminated.
```

The PLINK program is executed:

```
>PLINK
Write name of source file: ALFA:P2
Write name of destination file: ALFAR:P2
Write name of file for segment #1: ALFAX:P2
PLINK terminated.
```

These opeations will create a P-code file, ALFAR, which contains the linked program. It may then be executed in the normal manner:

```
>INTRE,ALFAR
```

PEX will alter the contents of the ALFA P-code file, but PLINK will not alter any file except, of course, the resulting P-code file, ALFAR. If, therefore, a change is made in ALFA, it is recompiled and PEX and PLINK should be run again; there is no need to recompile ALFAX. If a change is made in BETA (file ALFAX), BETA should be recompiled and PLINK should be run. There is no need to execute PEX again.

4. The Structure of the Compilation Modules.

Chapter 3 gave a short description of the structure of the compilation modules. This chapter gives a more detailed description.

The word procedure will be used to mean both procedures and functions.

4.1. The Main Module.

This module should contain an appropriate declaration of the external segment procedures required. The declaration should describe all the parameters passed to the segment and a possible value returned.

Declarations of local identifiers within the segment need not be included, nor does the segment body need to contain any statements.

If, however, the segment procedure itself contains another segment procedure, the declaration of this procedure should be included, even though the main module will never reference it directly.

Note that the number of segment procedures may not exceed 15, and that they must be the first procedures declared in a program. They may, however, be preceded by procedure declarations that use the key word EXTERNAL or FORWARD.

4.2. The Modules Containing the Seperate Segments.

The structure of these modules is a bit more complicated, and the user should be careful when writing them. Remember that PLINK performs no check that the various modules are compatible.

A module containing a segment procedure to be used in another program should contain basically all the declarations in the main program that precede the declaration of the procedure itself.

However, the following declarations may be omitted from the module:

Any global CONST or TYPE declaration not required by the segment procedure.

Any global VAR declaration following the declaration of the last global variable used by the segment procedure.

Any global VAR declaration not used by the segment procedure, provided that the declarations are replaced by another declaration that fills the same amount of bytes.

The body of any segment procedure preceding the procedure in question may, of course be omitted; however, the declaration of any possible nested segment procedure within another segment procedure must be included.

The programmer must include the declarations of variables and procedures in the same order as in the main module.

4.3. On Segment Numbering.

The PEX and PLINK programs refer to the segments with segment numbers.

The segments of a program are numbered consecutively in order of appearance of any PROGRAM, SEGMENT PROCEDURE, or SEGMENT FUNCTION declaration. The numbering begins with zero, but the user should never have an external segment number zero, as zero refers to the main program itself.

The follwing example illustrates the segment numbering:

```
PROGRAM XYZ;                                --Segment no. 0

SEGMENT PROCEDURE A;                        --Segment no. 1

  SEGMENT FUNCTION B: INTEGER;              --Segment no. 2
  BEGIN
    body of segment no. 2
  END;

BEGIN
  body of segment no. 1
END;

SEGMENT PROCEDURE C;                        --Segment no. 3
BEGIN
  body of segment no. 3
END;

BEGIN
  body of segment no. 0
END.
```

5. The Link Programs.

PEX and PLINK are themselves written in Pascal. It is possible in spite of this to execute the programs under MONITOR control.

5.1. The PEX Program.

PEX requests the user to enter the name of the P-code file containing the main program module. If this file name specification does not contain a disk identification, PEX requests the user to enter the disk-ID. Pressing the escape key will abort the program at this point.

The user is then asked to enter the number of a segment which should be indicated as being external. The number is entered, followed by a depression of the RETURN key, whereupon the user is asked to enter the name of another segment.

This sequence of questions is terminated by pressing the escape key in response to the segment question.

5.2. The PLINK Program.

PLINK requests the user to enter the name of the P-code file containing the main program module. (PLINK uses the word "source" to designate this file. This, of course, does not mean the source text file, but the P-code file used as input for PLINK.) If this file name specification does not contain a disk identification, PLINK will search all disks in the system.

The user is next requested to enter the name of the file which will receive the resulting linked P-code. If this file name specification does not contain a disk identification, the PLINK program requests the user to enter the disk-ID.

The user is requested to enter the name of the file containing the P-code for each segment that was marked by the PEX program in the main module as being external. If this file name specification does not contain a disk identification, PLINK will search all disks in the system.

The escape key will abort the program at any time.

The various segment procedures are not required to reside in different files. A file may contain two segment procedures that are to be linked to the main module.

5.3. Special Considerations.

If the seperately compiled modules contain declarations of EXTERNAL procedures, they must, as usual, be compiled using the compiler E option.

The relocatable file of the main program module is then linked to create a special interpreter. However, this interpreter requires that the P-code be in a file with the same name as the main program module. Because the PLINK program creates a new P-code file with a different name, renaming of this file will be required.

The programmer may take advantage of the fact that it is possible to perform the PEXing and PLINKing in successive steps. It is thus possible, for example, to retain the dummy

version of a segment while testing other segments. If, for example, the main program module contains 2 external segments, the user may first use PEX to mark segment number 1 as external, then create a P-code file using PLINK. When this P-code has been tested, PEX may be used to mark segment 2 of this code as external, and link anew.

6. A Larger Example.

This chapter illustrates some of the principles discussed in the preceding chapters.

Consider the following main program module:

```
PROGRAM ZETA;

CONST TEN=10;

VAR A,B,C: REAL;
    I,J,K: INTEGER;
    STR:   STRING;

PROCEDURE PR1(VAR ALFA: INTEGER; BETA: INTEGER);
FORWARD;

FUNCTION F(INT: INTEGER): INTEGER;
EXTERNAL;

SEGMENT PROCEDURE SEG1(VAR S: STRING);

  SEGMENT FUNCTION SEG2(S: STRING): INTEGER;
  BEGIN
    (* External segment *)
  END;

BEGIN
  (* External segment *)
END;

SEGMENT PROCEDURE SEG3;
BEGIN
  (* External segment *)
END;

PROCEDURE PR1;
BEGIN
  ALFA:=2*BETA;
END;

BEGIN (* MAIN *)
  READ(A,B,C);
  WRITELN(A*TEN,B*TEN,C*TEN);

  STR:='';
  EDIT(STR:20);
  SEG1(STR);
  WRITELN(STR);

  READ(I,J,K);
  SEG3;
  WRITELN(I:5,J:5,K:5);
END.
```

Let SEG1 be defined in the following module:

```
PROGRAM ZETA1;

PROCEDURE PR1(VAR ALFA: INTEGER; BETA: INTEGER);
FORWARD;

FUNCTION F(INT: INTEGER): INTEGER;
EXTERNAL;

(* EXTERNAL *) SEGMENT PROCEDURE SEG1(VAR S: STRING);
VAR COUNT: INTEGER;

  SEGMENT FUNCTION SEG2(S: STRING): INTEGER;
  BEGIN
    (* External segment *)
  END;

BEGIN
  COUNT:=SEG2(S) MOD 20 + 1;
  S(COUNT):='*';
END;

PROCEDURE PR1;     (* Required because of the FORWARD declaration *)
BEGIN
END;

BEGIN
END.
```

Let SEG2 be defined in the following module:

```
PROGRAM ZETA2;

PROCEDURE PR1(VAR ALFA: INTEGER; BETA: INTEGER);
FORWARD;

FUNCTION F(INT: INTEGER): INTEGER;
EXTERNAL;

SEGMENT PROCEDURE SEG1(VAR S: STRING);

  (* EXTERNAL *) SEGMENT FUNCTION SEG2(S: STRING): INTEGER;
  VAR C: INTEGER;
  BEGIN
    C:=ORD(S(4));
    SEG2:=F(C);
  END;

BEGIN
END;

PROCEDURE PR1;     (* Required because of the FORWARD declaration *)
BEGIN
END;

BEGIN
END.
```

Finally, let SEG3 be defined by the following module:

```
PROGRAM ZETA3

VAR DUMMY: PACKED ARRAY (1..24) OF CHAR;
    I,J,K: INTEGER;

PROCEDURE PR1(VAR ALFA: INTEGER; BETA: INTEGER);
FORWARD;

FUNCTION F(INT: INTEGER): INTEGER;
EXTERNAL;

SEGMENT PROCEDURE SEG1(VAR S: STRING);

  SEGMENT FUNCTION SEG2(S: STRING): INTEGER;
  BEGIN
    (* External segment *)
  END;

BEGIN
  (* External segment *)
END;

(* EXTERNAL *) SEGMENT PROCEDURE SEG3;
BEGIN
  I:=I+1;
  J:=J+2;
  PR1(K,J);
END;

PROCEDURE PR1;     (* Required because of the FORWARD declaration *)
BEGIN
END;

BEGIN
END.
```

The following MIKADOS commands will compile and link these modules. It is assumed that the modules reside in the files ZETA, ZETA1, ZETA2, and ZETA3 respectively.

```
>PASCALE,ZETA,,E
>PASCALE,ZETA1,,E
>PASCALE,ZETA2,,E
>PASCALE,ZETA3,,E
>PEX
Write name of file: ZETA:P2
Write number of segment to be made external: 1
Write number of segment to be made external: 2
Write number of segment to be made external: 3
Write number of segment to be made external: <escape>
PEX terminated.
>PLINK
Write name of source file: ZETA:P2
Write name of destination file: ZETAR:P2
Write name of file for segment #1: ZETA1
Write name of file for segment #2: ZETA2
Write name of file for segment #3: ZETA3
PLINK terminated.
>LINK,ZETA,R1
>RENAME
Enter old file name:disc identification (e.g. AB:P1) ZETA:P2
Enter new file name ZETAS
Enter file type P
Enter old file name:disc identification (e.g. AB:P1) ZETAR:P2
Enter new file name ZETA
Enter file type P
Enter old file name:disc identification (e.g. AB:P1) <escape>
RENAME terminated.
>ZETA
  -- Now the program will run.
```