DIRECTORY
  AltoDefs: FROM "altodefs",
  IODefs: FROM "iodefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs";

DEFINITIONS FROM StreamDefs, SegmentDefs;

StringCompactor: PROGRAM IMPORTS IODefs, SegmentDefs, StreamDefs, StringDefs, SystemDefs

BEGIN

CompStrDesc: TYPE = RECORD [
  offset, length: CARDINAL];

nStrings: CARDINAL;
nChars: CARDINAL;
nArrays: CARDINAL;

InStream, sOutStream, rOutStream: StreamHandle;

SLptr: TYPE = POINTER TO SL;

SL: TYPE = RECORD [
  link: SLptr,
  startindex: StreamIndex,
  length: CARDINAL];

ALptr: TYPE = POINTER TO AL;

AL: TYPE = RECORD [
  link: ALptr,
  name: NL,
  ARRAYIndex: StreamIndex,
  NeedsIndexDef: BOOLEAN,
  headSL, tailSL: SLptr,
  nStrings: CARDINAL];

NL: TYPE = RECORD [
  startindex: StreamIndex,
  length: CARDINAL];

BackUp: PROCEDURE [s: StreamHandle] =
BEGIN OPEN StreamDefs;
SetIndex[s, ModifyIndex[GetIndex[s], -1]];
RETURN
END;

NextString: PROCEDURE [s: SLptr] RETURNS [BOOLEAN] =
BEGIN
  c: CHARACTER;
  nc: CARDINAL + 0;
  QuoteFound, CollectingChars: BOOLEAN + FALSE;

  DO
    IF InStream.endof[InStream] THEN SIGNAL SyntaxError;
    c + InStream.get[InStream];
    IF c = ':' AND -CollectingChars THEN RETURN[FALSE];
    IF c = '"' THEN
      IF QuoteFound THEN
        IF CollectingChars THEN
          BEGIN QuoteFound + FALSE; nc + nc+1 END
        ELSE ERROR
      ELSE
        IF CollectingChars THEN QuoteFound + TRUE
        ELSE BEGIN s.startindex + GetIndex[InStream]; CollectingChars + TRUE; END
      END
    ELSE
      IF QuoteFound THEN
        BEGIN s.length + nc; BackUp[InStream]; EXIT END
      ELSE IF CollectingChars THEN nc + nc+1;
    ENDLOOP;

nChars + nChars + nc;
nStrings + nStrings+1;
RETURN[TRUE]
END;

lastCR: StreamIndex;

ParseState: TYPE = {start, aRay, arRay, arrAy, arraY, sTring, string,
                   string, string, stringG, Of, of, end};

NextItem: PROCEDURE [a: ALptr] =
BEGIN
  c: CHARACTER;
  nc: CARDINAL ~= 0;
  state: ParseState + start;
  array: BOOLEAN;

  DO
  IF InStream.endof[InStream] THEN SIGNAL AllDone;
  c ~ InStream.get[InStream];
  nc ~ nc+1;
  SELECT c FROM
    'A' =>
    state ~ SELECT state FROM
       start => aRay,
       arRay => arRay,
       string => string,
       ENDCASE => start;
    'R' =>
    state ~ SELECT state FROM
       aRay => arRay,
       arRay => arRay,
       string => string,
       ENDCASE => start;
    'Y' =>
    BEGIN
      IF state = arRay THEN
      BEGIN a.ARRAYindex ~ GetIndex[InStream]; state ~ end END
      ELSE state ~ start;
      END;
    'S' =>
    IF state = start THEN
      BEGIN a.name.length ~ nc-1; state ~ sTring END
      ELSE state ~ start;
    'T' =>
    state ~ IF state = sTring THEN sTring ELSE start;
    'I' =>
    state ~ IF state = string THEN string ELSE start;
    'N' =>
    state ~ IF state = string THEN string ELSE start;
    'G' =>
    IF state = string THEN
      BEGIN array ~ FALSE; state + end END
      ELSE state ~ start;
    ENDLOOP;
    a.name.startindex ~ lastCR;
    a.NeedsIndexDef ~ array;
    IF array THEN
      BEGIN
      state ~ Of;
      DO
      IF InStream.endof[InStream] THEN SIGNAL SyntaxError;
      c ~ InStream.get[InStream];
      nc ~ nc+1;
      SELECT c FROM
        IN [DC..'] =>
        SELECT state FROM
           start => state + Of;
    END:
OF \rightarrow\) NULL;  
\text{end} \rightarrow\) EXIT;  
ENDCASE \rightarrow\) state + start;  
'O \rightarrow\)  
state + IF state = OF THEN OF ELSE start;  
'F \rightarrow\)  
state + IF state = of THEN end ELSE start;  
ENDCASE \rightarrow\) BEGIN a.NeedsIndexDef = FALSE; state + start; END;  
ENDLOOP;  
a.name.length + nc;  
END;  
CollectStrings[a];  
IF array THEN nArrays + nArrays + 1;  
RETURN  
END;  
AllDone: SIGNAL = CODE;  
SyntaxError: SIGNAL = CODE;  
headsAL, tailsAL: ALptr;  
CollectStrings: PROCEDURE [a: ALptr] =  
BEGIN  
s: SLptr;  
oOldnStrings: CARDINAL + nStrings;  
a.headSL + a.tailSL + NIL;  
WHILE NextString[s = AllocateSL[]] DO  
AppendSL[a, s];  
ENDLOOP;  
SystemDefs.FreeHeapNode[a];  
a.nstrings + nStrings = oldnStrings;  
RETURN  
END;  
CollectArrays: PROCEDURE =  
BEGIN  
a: ALptr;  
headAL + tailsAL + NIL;  
nStrings + 0; nChars + 0; nArrays + 0;  
lastCR + StreamIndex[0,0];  
DO  
NextItem[a = AllocateAL[]] !  
AllDone \Rightarrow\) BEGIN SystemDefs.FreeHeapNode[a]; EXIT END;  
AppendAL[a];  
ENDLOOP;  
RETURN  
END;  
AllocateSL: PROCEDURE RETURNS [s: SLptr] =  
BEGIN  
s + SystemDefs.AllocateHeapNode[SIZE[SL]];  
s.link + NIL;  
RETURN  
END;  
AppendSL: PROCEDURE [a: ALptr, s: SLptr] =  
BEGIN  
IF a.tailSL = NIL THEN a.headSL + s  
ELSE a.tailSL.link + s;  
a.tailSL + s;  
RETURN  
END;  
AllocateAL: PROCEDURE RETURNS [a: ALptr] =  
BEGIN  
a + SystemDefs.AllocateHeapNode[SIZE[AL]];  
a.link + NIL;  
RETURN  
END;  
AppendAL: PROCEDURE [a: ALptr] =
BEGIN
IF tailAL = NIL THEN headAL ← a
ELSE tailAL.link ← a;
tailAL ← a;
RETURN
END;

OutCompactStrings: PROCEDURE =
BEGIN
  tSH: StreamHandle;
  a: ALptr ← headAL;
  s: SLptr;
  charpos: CARDINAL ← 0;
  i: CARDINAL;
  prevs: SLptr;
  c: CHARACTER;

  sOutStream.reset[sOutStream];
  sOutStream.put[sOutStream, nStrings*SIZE[CompStrDesc]+1];
  WHILE a # NIL DO
    s ← a.headSL;
    WHILE s # NIL DO
      sOutStream.put[sOutStream, charpos];
      sOutStream.put[sOutStream, s.length];
      charpos ← charpos+s.length;
      s ← s.link;
      ENDLOOP;
      a ← a.link;
      ENDLOOP;
  sOutStream.put[sOutStream, nChars];
  sOutStream.put[sOutStream, nChars];
  CleanupDiskStream[sOutStream];
  tSH ← CreateByteStream[outFH, Write+Append];
  SetIndex[tSH, GetIndex[sOutStream]];}
  sOutStream.reset[sOutStream];
  sOutStream.destroy[sOutStream];
  sOutStream ← tSH;
  a ← headAL;
  WHILE a # NIL DO
    s ← a.headSL;
    WHILE s # NIL DO
      SetIndex[InStream, s.startindex];
      FOR i IN [0..s.length) DO
        c ← InStream.get[InStream];
        IF c = "$" THEN c ← InStream.get[InStream];
        sOutStream.put[sOutStream, c]
        ENDLOOP;
        prevs ← s;
        s ← s.link;
        SystemDefs.FreeHeapNode[prevs];
        ENDLOOP;
        a ← a.link;
        ENDLOOP;
        sOutStream.destroy[sOutStream];
        RETURN
        END;

OutRealStrings: PROCEDURE =
BEGIN
  a: ALptr ← headAL;
  s: SLptr;
  wordpos: CARDINAL ← nStrings+1;
  i: CARDINAL;
  prevs: SLptr;
  c: CHARACTER;
  buffer: RECORD[even,odd: CHARACTER];
  parity: {even,odd} ← even;
  FlushBuffer: PROCEDURE =
  BEGIN
    IF parity = odd THEN PutChar[IODefs.NUL];
    END;
  PutChar: PROCEDURE [c: CHARACTER] =
  BEGIN
    IF parity = even THEN BEGIN buffer.even ← c; parity ← odd END
    ELSE
BEGIN
  buffer.odd + c;
sOutStream.put[sOutStream, buffer];
  parity + even
END;

sOutStream.reset[sOutStream];
sOutStream.put[sOutStream, nStrings];
WHILE a # NIL DO
  s + a.headSL;
  WHILE s # NIL DO
    sOutStream.put[sOutStream, wordpos];
    wordpos + wordpos + StringDefs.WordsForString[s.length];
    s + s.link;
  ENDLOOP;
  a + a.link;
ENDLOOP;

sOutStream.reset[sOutStream];
sOutStream.put[sOutStream, nStrings];
WHILE a # NIL DO
  s + a.headSL;
  WHILE s # NIL DO
    SetIndex[InStream, s.startindex];
    FlushBuffer[];
    sOutStream.put[sOutStream, s.length];
    sOutStream.put[sOutStream, s.length];
    FOR i IN [0..s.length) DO
      c + InStream.get[InStream];
      IF c = '"' THEN c + InStream.get[InStream];
      PutChar[c]
    ENDLOOP;
    prevs + s;
    s + s.link;
    SystemDefs.FreeHeapNode[prevs];
  ENDLOOP;
  a + a.link;
ENDLOOP;
FlushBuffer[];
sOutStream.destroy[sOutStream];
RETURN
END;

OutStrings: PROCEDURE [compact: BOOLEAN] =
BEGIN
  IF compact THEN OutCompactStrings[] ELSE OutRealStrings[];
  RETURN
END;

OutRecordDecl: PROCEDURE [compact: BOOLEAN] =
BEGIN
  a: ALptr + headAL;
  preva: ALptr;
  i: CARDINAL;
  rOutStream.reset[rOutStream];
  FOR i IN [0..routfile.length) DO
    IF routfile[i] = '.' THEN EXIT;
    rOutStream.put[rOutStream, routfile[i]];
  ENDLOOP;
  OutString["": DEFINITIONS =
BEGIN
  IF compact THEN OutString[" CSRptr: TYPE = POINTER TO CompStrRecord;
  CompStrDesc: TYPE = RECORD [offset, length: CARDINAL];
  CompStrRecord: TYPE = RECORD [
    relativebase: CARDINAL,",
  ]
  ELSE OutString[" StringRecord: TYPE = RECORD [nStrings: CARDINAL,
  "]
  DO
    SetIndex[InStream, a.name.startindex];
    OutString[" "];
  END;
  RETURN
END;
FOR i IN [0..a.name.length) DO
  IF a.NeedsIndexDef THEN
    IF GetIndex[InStream] = a.ARRAYindex THEN
      BEGIN
        OPEN IODefs;
        OutString["[0..]";
        OutNumber[rOutStream, a.nstrings, NumberFormat[10,FALSE,FALSE,0]];
        rOutStream.put[rOutStream,'\n'];
      END;
      rOutStream.put[rOutStream, InStream.get[InStream]]; ENDLOOP;
    OutString[IF compact THEN "CompStrDesc" ELSE "STRING"];
    prev a + a;
    a + a.link;
    SystemDefs.FreeHeapNode[prev a];
    IF a = NIL THEN EXIT;
    rOutStream.put[rOutStream, ','];
    rOutStream.put[rOutStream, IODefs.CR];
  ENDLOOP;
  OutString["\n"];
  rOutStream.destroy[rOutStream];
RETURN END;

OutString: PROCEDURE [s: STRING] =
BEGIN
  i: CARDINAL;
  FOR i IN [0..s.length) DO rOutStream.put[rOutStream, s[i]]; ENDLOOP;
RETURN End;

YesNo: PROCEDURE [question: STRING] RETURNS [BOOLEAN] =
BEGIN
  OPEN IODefs;
  c: CHARACTER;
  WriteString[question];
  c + ReadChar[];
  DO
    SELECT c FROM 'Y','y' => BEGIN WriteLine["Yes"] ; RETURN[TRUE] END;
    'N','n' => BEGIN WriteLine["No"] ; RETURN[FALSE] END;
    ENDCASE => WriteString["? Type Y or N "];
  ENDLOOP;
  END;

infile: STRING = [40];
soutfile: STRING = [40];
routfile: STRING = [40];
outFH: FileHandle;
compact: BOOLEAN;
BEGIN
  OPEN IODefs;
  WriteLine["Mesa String Compactor"]; DO
    WriteChar[CR];
    WriteString["Input file: ";]
    ReadID[infile];
    IF infile.length = 0 THEN EXIT;
    WriteString[" string output file: "];]
    ReadID[soutfile];
    WriteString[" record output file: "];]
    ReadID[routfile];
    WriteChar[CR];
    compact + YesNo["Do you want the compact representation? "];
    InStream + CreateByteStream[NewFile[infile, Read, OldFileOnly], Read];
    sOutStream + CreateWordStream[NewFile[soutfile, Write+Append, DefaultAccess], Write+Append];
    rOutStream + CreateWordStream[NewFile[routfile, Write+Append, DefaultAccess], Write+Append];
    CollectArrays[]; OutStrings[compact]; OutRecordDec[compact];
    WriteDecimal[nArrays]; WriteString(" arrays, ");
    WriteDecimal[nStrings]; WriteString(" strings, ");
WriteDecimal[nChars]; WriteLine[" characters."];
InStream.destroy[InStream];
ENDLOOP;

END
END...