```
////////////////////////////////////////////////////////////////////
//  PROGRAM: Party Music
//  AUTHOR:  Bill McQuain
//  SYSTEM:  Microsoft Visual C++ 6.0 / Microsoft Windows NT
//  DATE:    18 April 1999
//
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
//
//  Global declarations of constants and types:
//

  MinutesPerTape    45
  SecondsPerMinute  60
  MaxSongs          30
  Sentinel           0

//  Enumerated type for song categories:

 songType  Blues, Country, HipHop, Pop, Rock, Other

//  Struct variable type for song data:

typedef struct {           // Structured variable to hold:
   int      songNumber;    //     song number
   int      songMinutes;   //     minute field of length
   int      songSeconds;   //     seconds field of length
   int      songLength;    //     song length in seconds
   songType Type;          //     song type
   bool     Used;          //     usage flag (true if included on
                           //                 tape)
} songRecord;
```

```
////////////////////////////////////////////////////////////////
Begin function main():

   // Major variables housed in main():
   SongList[]         // array of songRecords to hold song data
   numSongs           // number of songs in list

I.    Read song data list into array
      and count songs: numSongs <-- ReadSongs()

II.   Determine which songs to include on tape: DesignTape()

III.  Print list of songs: PrintList()

End function main():
```

```
////////////////////////////////////////////////////////////////////
//
// Function ReadSongs() reads the input file of song data, storing the
// data into the songRecord variables in SongList.
//
// Parameters:
//    SongList[]     array of song records
//
// Return value: number of songs for which data was read
//
// Pre:  SongList[] is of dimension MaxSongs
// Post: SongList[] holds data for up to MaxSongs songs
//
Begin function ReadSongs():

   // Major local variables:
   inSongs     // input stream
   sNumber     // temporary storage for reading input values
   sMinutes
   sSeconds
   SType
   Idx       0 // array index

I.    Open input file:  music.data

II.   Skip over header line in input file.

III.  Read first line of song data (priming read):
      A.  read song number:             sNumber
      B.  read song category:           sType
      C.  read song minutes:            sMinutes
      D.  read song seconds:            sSeconds

IV.   While there is more data and the array isn't full and
            the sentinel hasn't been read Do

      A.  Copy song number and length into song record:
          1.  SongList[Idx].songNumber  <-- sNumber
          2.  SongList[Idx].songMinutes <-- sMinutes
          3.  SongList[Idx].songSeconds <-- sSeconds


      B.  Store song length in seconds in song record:
          1.  SongList[Idx].songLength  <-- song length in seconds

      C.  Store appropriate song type (enum value) in song record:
          1.  SongList[Idx].Type        <-- Classify(sType)
```

```
      D.  Set used flag for this song:
          a.  SongList[Idx].Used        <-- false

      E.  Increment index for next store operation

      F.  Read next line of song data:
          1.  read song number:             sNumber
          2.  read song category:           sType
          3.  read song minutes:            sMinutes
          4.  read song seconds:            sSeconds

      EndWhile

V.    Close input file
End function ReadSongs().
```

```
////////////////////////////////////////////////////////////////////
//
// Function Classify() converts the character specifying the song
// category in the input file to the corresponding enum value.
//
// Parameters:
//    Type     character value indicating song category
//
// Return value: enum value corresponding to Type
//
// Pre:  Type has a value ('B', 'C', 'H', 'P', 'R' are valid)
// Post: If Type has valid value, enum value corresponding to that
//       value has been returned; otherwise Other is returned.
//
Begin function Classify():

   If Type == 'B' Then
      return Blues.
   Endif
   If Type == 'C' Then
      return Country.
   Endif
   If Type == 'H' Then
      return HipHop.
   Endif
   If Type == 'P' Then
      return Pop.
   Endif
   If Type == 'R' Then
      return Rock.
   Endif

   return Other.

End function Classify().
```

```
////////////////////////////////////////////////////////////////
//
// Function DesignTape() determines which songs in SongList[] can be
// included on the tape, processing the songs in the order they occur.
//
// Parameters:
//    SongList[]   list of songs to be considered
//    numSongs     number of songs in SongList[]
//
// Return value: none
//
// Pre:  SongList[] contains data for numSongs songs
// Post: Each song record in SongList[] has been updated to show
//       whether that song is included.
//
Begin function DesignTape():

   // Major local variables:
   timeRemaining    // time remaining on tape; init to length of tape
   Idx              // array index

I.  For each song:
    A.  If song will fit on the tape Then
        1.  Update time remaining on tape.
        2.  Mark song to be used:  SongList[Idx].Used <-- true

End function DesignTape().
```

```
//////////////////////////////////////////////////////////////////
//
// Function PrintList() handles printing the output file.
//
// Parameters:
//    SongList[]   list of songs to be considered
//    numSongs     number of songs in SongList[]
//
// Return value: none
//
// Pre:  SongList[] contains data for numSongs songs
// Post: specified output has been printed to output file
//
Begin function PrintList():

   // Major local variables:
   outSongs                  // output stream
   totalLength  0            // total length of songs on tape
   Idx                       // array index

I.   Open output file:  tape.list

II.  Print output header:   PrintHeader(outSongs)

III. For each song in list:

     A. Print song data:
        1.  song number     SongList[Idx].songNumber
        2.  song minutes    SongList[Idx].songMinutes
        3.  song seconds    SongList[Idx].songSeconds

     B. // Print data according to whether song is used:
        If song is used Then
           1.  Update total length of tape.
           2.  Print minutes in total length of tape.
           3.  Print seconds in total length of tape.
        Else
           1.  Print "song will not fit".
        Endif

IV.  Print trailer data:  PrintTrailer()

V.   Close output file.

End function PrintList().
```

```
//////////////////////////////////////////////////////////////////
//
// Function PrintHeader() prints the header to the output file.
//
// Parameters:
//     Out     output stream variable
//
// Return value: none
//
// Pre:  output file has been opened
// Post: specified header has been printed to output file
//
Begin function PrintHeader():

I.    Print programmer ID:   "Programmer:  Bill McQuain"

II.   Print output title:    "Party Music"

III.  Print a blank line.

IV.   Print column headers:
          "Song          Song Time           Total Time"
          "Number    Minutes   Seconds    Minutes   Seconds"

V.    Print delimiters for table.

End function PrintHeader().
```

```
//////////////////////////////////////////////////////////////////
//
// Function PrintTrailer() prints the time remaining on the tape and
// the song category counts.
//
// Parameters:
//    Out          output stream
//    SongList[]   list of songs to be considered
//    numSongs     number of songs in SongList[]
//    totalLength  total length of songs on tape in seconds
//
// Return value: none
//
// Pre:  Output file has been opened.
//       SongList[] contains data for numSongs songs.
//       totalLength equals length of tape in seconds.
// Post: Specified information has been printed to output file.
//
Begin function PrintTrailer():

   // Major local variables:
   numBlues   0              // number of blues songs on tape
   numCountry 0              //           country songs on tape
   numHipHop  0              //           hiphop songs on tape
   numPop     0              //           pop songs on tape
   numRock    0              //           rock songs on tape
   Idx                       // array index

I.   Calculate and print time remaining on tape:
     A.  Time remaining is total tape length minus length of songs.
     B.  Print table delimiter line.
     C.  Print labels and time remaining in minutes and seconds.
     D.  Print blank line.

II.  Count how many songs in each category are on tape:
     For each song in list:
         If song is used on tape Then
            If song type is Blues Then
               increment numBlues.
            Else if song type is Country Then
               increment numCountry.
            Else if song type is HipHop Then
               increment numHipHop.
            Else if song type is Pop Then
               increment numPop.
            Else if song type is Rock Then
               increment numRock.
```

```
            Else
               increment numOther.
            Endif
         Endif

III.  Print song category counts:
      A.  Print header:   "Song categories".
      B.  If numBlues is greater than 0 Then
          1.   Print label:  "Blues".
          2.   Print count:  numBlues.
          Endif
      B.  If numCountry is greater than 0 Then
          1.   Print label:  "Country".
          2.   Print count:  numCountry.
          Endif
      B.  If numHipHop is greater than 0 Then
          1.   Print label:  "HipHop".
          2.   Print count:  numHipHop.
          Endif
      B.  If numPop is greater than 0 Then
          1.   Print label:  "Pop".
          2.   Print count:  numPop.
          Endif
      B.  If numRock is greater than 0 Then
          1.   Print label:  "Rock".
          2.   Print count:  numRock.
          Endif

End function PrintTrailer().
```