Inf Progtech 1, Praktikums-Beispiel

Morsecode decodieren

Gesucht ist ein Programm, das mit einem *Morsetext auf der Befehlszeile* aufgerufen wird[1] und *den entsprechenden Klartext ausgibt*.

Das Programm soll intern einen *binären Baum* verwenden, der aber kein Suchbaum ist: Der Baum ist so gebaut, dass man damit Morsezeichen decodieren kann.

Seine Knoten sind wie folgt deklariert:

```
typedef struct _knoten {
  char zeichen;           // Zeichen, das dem Pfad zu diesem Knoten entspricht
                          // '\0', wenn dieser Knoten keinem Zeichen entspricht
  struct _knoten *punkt;  // Subbaum, wenn als nächstes ein Punkt folgt
  struct _knoten *strich; // Subbaum, wenn als nächstes ein Strich folgt
} baumKnoten;
```

Man beginnt beim Decodieren für jedes Morsezeichen bei der Wurzel des Baumes und analysiert das Morsezeichen Zeichen für Zeichen: Ist das nächste Zeichen ein `'.'`, folgt man dem Pointer **`punkt`**, ist es ein `'-'`, folgt man dem Pointer **`strich`**. Ist das Morsezeichen zu Ende (Zwischenraum oder Stringende), so enthält der Baumknoten, bei dem man angekommen ist, das Zeichen **`zeichen`**, das dem Morsezeichen entspricht. Dieses wird ausgegeben.

Weiters stelle ich dir noch eine (unvollständige) Morsetabelle zur Verfügung:

```
typedef struct {
  const char zeichen;
  const char *code;
} codeDef;

codeDef codeTab[] = {
  {'a', ".-"},
  {'b', "-..."},
  {'c', "-.-."},
  {'d', "-.."},
  {'e', "."},
  {'f', "..-."},
  {'g', "--."},
  {'h', "...."},
  {'i', ".."},
  {'j', ".---"},
  {'k', "-.-"},
  {'l', ".-.."},
  {'m', "--"},
  {'n', "-."},
  {'o', "---"},
  {'p', ".--."},
  {'q', "--.-"},
  {'r', ".-."},
  {'s', "..."},
  {'t', "-"},
  {'u', "..-"},
  {'v', "...-"},
  {'w', ".--"},
  {'x', "-..-"},
  {'y', "-.--"},
  {'z', "--.."},
  {'ä', ".-.-"},
  {'ö', "---."},
  {'ü', "..--"},
  {'C', "----"},
  {'1', ".----"},
  {'2', "..---"},
  {'3', "...--"},
  {'4', "....-"},
  {'5', "....."},
  {'6', "-...."},
  {'7', "--..."},
  {'8', "---.."},
  {'9', "----."},
  {'0', "-----"},
  {'.', ".-.-.-"},
  {',', "--..--"},
  {':', "---..."},
  {'?', "..--.."},
  {'(', "-.--.-"},
  {';', "-.-.-."},
  {'-', "-....-"},
  {'+', ".-.-."},
  {'/', "-..-."},
  {'\'', ".----."},
  {'=', "-...-"},
  {'\"', ".-..-."},
  {'!', "..--."},
  {' ', ""}
};
```

1 Du kannst dir aussuchen, ob dein Programm ein *einziges* **argv**-Wort für den gesamten Morsetext erwartet (einzelne Morsezeichen durch Zwischenraum getrennt, dafür muss der gesamte Morsetext auf der Befehlszeile in **"** … **"** eingeschlossen werden) oder ein **argv**-Wort *pro einzelnem Morsezeichen*.

Als **ersten Schritt** soll dein Programm aus dieser Tabelle *den oben beschriebenen Baum aufbauen*:

- Zuerst einmal wird ein *Wurzelknoten* dynamisch angelegt.
- Dann wird die Tabelle *Zeile für Zeile eingetragen*.

  Für jede Zeile folgt man von der Wurzel aus dem Morsecode Zeichen für Zeichen.
  Stößt man dabei bei **`punkt`** oder **`strich`** auf einen **`NULL`**-Pointer,
  wird ein *neuer Knoten* dynamisch angelegt und an dieser Stelle angehängt.

  Ist man am *Ende des Codes*, wird in dem betreffenden Baumknoten
  als **`zeichen`** derjenige Buchstabe eingetragen, der diesem Morsecode entspricht.
- Bei Zwischenknoten des Baumes, die *keinem* gültigen Morsecode entsprechen,
  soll in **`zeichen '\0'`** eingetragen werden (bzw. wird jeder neue Knoten
  zuerst einmal vorbeugend mit **`zeichen`** gleich **`'\0'`** und den beiden Pointern
  gleich **`NULL`** angelegt, erst nachträglich werden dann andere Werte eingetragen).
- Bei der Wurzel wird der *Zwischenraum* in **`zeichen`** eingetragen
  (dadurch sollte bei einem "leeren" Morsezeichen
  bzw. zwei aufeinanderfolgenden Zwischenräumen in der Eingabe
  ein Zwischenraum im decodierten Text ausgegeben werden).

Als **zweiten Schritt** soll dein Programm dann mit Hilfe dieses Baumes
die Eingabe wie oben beschrieben decodieren.

Hinweis: Achte auf sorgfältige *Fehlerprüfung*:

- Schlägt eine *Speicherallokation* fehl,
  soll das Programm mit einer Fehlermeldung enden.
- Enthält die Eingabe *andere Zeichen* als **`'.'`**, **`'-'`** und **`' '`**,
  soll eine Fehlermeldung ausgegeben werden.
- Auch in der *Code-Tabelle* sollen *falsche Zeichen* im Morsecode erkannt werden.
  Ebenso soll erkannt werden, wenn derselbe Morsecode in der Tabelle *zwei Mal*
  vorkommt.
- Steht man beim Decodieren auf einem Baumknoten mit **`zeichen`** gleich **`'\0'`**,
  wenn das eingegebene Morsezeichen zu Ende ist, so war das Morsezeichen
  unvollständig.
- Ist ein **`punkt`**- oder **`strich`**-Pointer, dem man beim Decodieren folgen will, **`NULL`**,
  so ist das Morsezeichen *ungültig*.