

1 Candles

1.1 Lösungsidee

Wenn die Höhe sich um 1 erhöht, scheinen sich alle Kerzen an jeder Seite der neuen "Hauptkerze" (die größte, mittlere in der Visualisierung) zu gruppieren. Da eine Kerze als Dreieck repräsentiert ist, verdreifacht sich die Anzahl der Kerzen, zusätzlich gibt es eine neue "Hauptkerze". Eine Christbaum der Höhe 1 hat genau eine Kerze. Damit ist die rekursive Lösung gefunden.

Die iterative Lösung funktioniert im Prinzip gleich. Die Variable result wird mit 1 initialisiert. Ist die Höhe gleich 1, so wird die Schleife übersprungen und 1 zurückgegeben. Ansonsten wird in einer Schleife bis h hochgezählt und immer das vorläufige Ergebnis verdreifacht und 1 addiert.

1.2 Quellcode

```
1  PROGRAM Candles;
2
3  FUNCTION CandlesRec(h: INTEGER): INTEGER;
4  BEGIN
5      IF h = 1 THEN BEGIN
6          CandlesRec := 1;
7      END ELSE BEGIN
8          CandlesRec := 1 + (3 * CandlesRec(h-1));
9      END;
10 END;
11
12 FUNCTION CandlesIt(h: INTEGER): INTEGER;
13 VAR
14     i: INTEGER;
15     result: INTEGER;
16 BEGIN
17     result := 1;
18     FOR i := 2 TO h DO BEGIN
19         result := 1 + (3 * result);
20     END;
21     CandlesIt := result;
22 END;
23
24 VAR
25     userInput: INTEGER;
26
27 BEGIN
28     Write('How high is the tree? > ');
29     ReadLn(userInput);
30
31     IF userInput > 0 THEN BEGIN
32         Write('CandlesRec:');
33         WriteLn(CandlesRec(userInput):3);
34
35         Write('CandlesIt:');
36         WriteLn(CandlesIt(userInput):3);
37     END ELSE BEGIN
38         WriteLn('Enter a positive value');
39     END;
40 END.
```

1.3 Tests

Input	Beschreibung	Output
1	-	How high is the tree? > 1 CandlesRec: 1 CandlesIt: 1
2	-	How high is the tree? > 2 CandlesRec: 4 CandlesIt: 4
3	-	How high is the tree? > 3 CandlesRec: 13 CandlesIt: 13
0	Wert kleiner gleich 1	How high is the tree? > 0 Enter a positive value

2 Floor-Funktion

2.1 Lösungsidee

Der Baum soll iterativ so lange durchwandert werden, bis entweder genau der gesuchte Wert gefunden ist, oder einer der Teilbäume NIL ist. In der Schleife wird jeder Wert als potenzieller Treffer gespeichert, der kleiner als der gegebene Wert x ist.

Ist der Zeiger auf den nächsten Knoten (bei mir "st") gleich dem Wert x , wird dieser zurückgegeben, ansonsten der zuletzt gespeicherte Treffer.

Hilfsfunktionen ermöglichen die Ein- und Ausgabe des Baumes.

2.2 Quellcode

```
1  PROGRAM FindFloor;
2  TYPE
3      TreeNodePtr = ^TreeNode;
4      TreeNode = RECORD
5          left, right: TreeNodePtr;
6          data: INTEGER;
7      END;
8      TreePtr = TreeNodePtr;
9
10  PROCEDURE InsertTree(VAR tree: TreePtr; val: INTEGER);
11      VAR
12          n: TreeNodePtr;
13          st, pt: TreeNodePtr;
14  BEGIN
15      New(n);
16      n^.left := NIL;
17      n^.right := NIL;
18      n^.data := val;
19
20      IF tree = NIL THEN BEGIN
21          tree := n;
22          Exit;
23      END;
24
25      st := tree;
26      pt := NIL;
27      WHILE st <> NIL DO BEGIN
28          pt := st;
29          IF val < st^.data THEN BEGIN
30              st := st^.left;
31          END ELSE BEGIN
32              st := st^.right;
33          END;
34      END;
35
36      IF val < pt^.data THEN BEGIN
37          pt^.left := n;
38      END ELSE BEGIN
39          pt^.right := n;
40      END;
41  END;
42
43  PROCEDURE ReadTree(VAR tree: TreePtr);
```

```

44     VAR
45         val: INTEGER;
46 BEGIN
47     val := -1;
48     WHILE val <> 0 DO BEGIN
49         Write(' > ');
50         ReadLn(val);
51         IF val <> 0 THEN BEGIN
52             InsertTree(tree, val);
53         END;
54     END;
55 END;
56
57 PROCEDURE WriteTree(tree: TreePtr);
58 BEGIN
59     IF tree <> NIL THEN BEGIN
60         WriteTree(tree^.left);
61         Write(tree^.data:3);
62         WriteTree(tree^.right);
63     END;
64 END;
65
66 PROCEDURE DisposeTree(VAR tree: TreePtr);
67 BEGIN
68     IF tree <> NIL THEN BEGIN
69         DisposeTree(tree^.left);
70         DisposeTree(tree^.right);
71         Dispose(tree);
72     END;
73     tree := NIL;
74 END;
75
76 FUNCTION Floor(tree: TreePtr; x: INTEGER): TreeNodePtr;
77     VAR
78         st: TreeNodePtr;
79         possResult, result: TreeNodePtr;
80 BEGIN
81     IF tree = NIL THEN BEGIN
82         Floor := NIL;
83         Exit;
84     END;
85     st := tree;
86     possResult := NIL;
87     result := NIL;
88
89     WHILE (st <> NIL) AND (st^.data <> x) DO BEGIN

```

```

90         IF st^.data < x THEN BEGIN (*mögl. Treffer, rechts weitersuchen*)
91             possResult := st;
92             st := st^.right;
93         END ELSE BEGIN (* st^.data > x, links weitersuchen*)
94             st := st^.left;
95         END;
96     END;
97
98     IF st = NIL THEN BEGIN
99         result := possResult;
100     END ELSE BEGIN
101         result := st;
102     END;
103
104     Floor := result;
105 END;
106
107 VAR
108     t: TreePtr;
109     x: INTEGER;
110     tmp: TreeNodePtr;
111
112 BEGIN
113     t := NIL;
114     ReadTree(t);
115     WriteTree(t);
116     WriteLn;
117
118     Write(' Floor of > ');
119     ReadLn(x);
120     tmp := Floor(t, x);
121     IF tmp <> NIL THEN BEGIN
122         WriteLn(tmp^.data);
123     END ELSE BEGIN
124         WriteLn('Error: value invalid or tree empty');
125     END;
126
127     DisposeTree(t);
128 END.

```

2.3 Tests

Input	Begründung	Output
2 4 3 5 9 7 6 x = 8	Insert funktioniert und richtiger Floor bei Standard-Input	<pre> > 2 > 4 > 3 > 5 > 9 > 7 > 6 > 0 2 3 4 5 6 7 9 Floor of > 8 7 </pre>
2 4 3 5 9 7 6 x = 5	Wert x kommt im Baum vor	<pre> 2 3 4 5 6 7 9 Floor of > 5 5 </pre>
-	Leerer Baum	<pre> > 0 Floor of > 4 Error: value invalid or tree empty </pre>
4 5 7 9 x = 3	Wert x kleiner als alle Werte im Baum, es gibt keinen Floor	<pre> > 4 > 5 > 7 > 9 > 0 4 5 7 9 Floor of > 3 Error: value invalid or tree empty </pre>
4 5 7 9 x = 10	Wert x größer als alle Werte im Baum	<pre> > 4 > 5 > 7 > 9 > 0 4 5 7 9 Floor of > 10 9 </pre>

3 Morsen mit Binärbaum

3.1 Lösungsidee

Die Prozedur InsertMorseCode soll den angegebenen Code als "Wegweiser" verwenden:

- ist das nächste Zeichen im Code ein '.', wird links fortgeführt
- ist das nächste Zeichen im Code ein '-', wird links fortgeführt

Damit die Reihenfolge der Aufrufe mit den Buchstaben egal ist, werden entlang dem Weg Dummyknoten angelegt, wenn die Knoten entlang eines Pfades noch nicht existieren. Alle diese Werte müssen am Ende mit echten Buchstaben überschrieben sein, was die Funktion TreeValid prüft.

Die Funktion Lookup folgt einem ähnlichen Prinzip, nur dass keine neuen Knoten eingefügt werden müssen. Der Code fungiert als "Wegweiser". Zurückgegeben wird der ch-Wert des Knotens, auf den der Zeiger 'st' nach der Schleife zeigt.

Die Funktion ConvertMessage übernimmt die finale Konvertierung des Codes in einen string. Der Hauptcode wird dabei Zeichen für Zeichen durchlaufen und dabei ein output zusammengebaut. Immer wenn ein Leerzeichen oder Strichpunkt kommt, endet ein Buchstabe, welcher mit Lookup übersetzt wird und zum output hinzugefügt wird. Bei einem Strichpunkt wird zusätzlich ein Leerzeichen hinzugefügt (Wortende).

3.2 Quellcode

```
1  PROGRAM MorseCode;
2
3  TYPE
4      TreeNodePtr = ^TreeNode;
5      TreeNode = RECORD
6          left, right: TreeNodePtr;
7          ch: CHAR;
8      END;
9      MorseTreePtr = TreeNodePtr;
10
11  PROCEDURE InsertMorseCode(tree: MorseTreePtr; ch: CHAR; code: STRING);
12      VAR
13          n, st: MorseTreePtr;
14          i: INTEGER;
15  BEGIN
16
17      IF tree = NIL THEN BEGIN
18          WriteLn('Error: Root Node does not exist');
19          Exit;
20      END;
21      tree^.ch := '/'; (*root*)
22
23      i := 1;
24      st := tree;
25      n := NIL;
26      WHILE i <= length(code) DO BEGIN
27          IF code[i] = '.' THEN BEGIN (*links gehen*)
28              IF st^.left = NIL THEN BEGIN (*Knoten erstellen, mit dummyWert
29                  füllen, in den Baum einhängen, damit man "drüberlaufen kann"*)
30                  New(n);
31                  n^.left := NIL;
32                  n^.right := NIL;
33                  n^.ch := ' ';
34                  st^.left := n;
35                  END;
36                  st := st^.left;
37              END ELSE BEGIN
38                  IF code[i] = '-' THEN BEGIN (*rechts gehen*)
39                      IF st^.right = NIL THEN BEGIN
40                          New(n);
41                          n^.left := NIL;
42                          n^.right := NIL;
```

```

43         n^.ch := ' ';
44         st^.right := n;
45     END;
46     st := st^.right;
47 END ELSE BEGIN (*ungültiger Code, Exit*)
48     WriteLn('Error: invalid Code - Exit');
49     Exit;
50 END;
51 END;
52 inc(i);
53 END;
54
55 IF i = 1 + length(code) THEN BEGIN (*st ist an der richtigen Stelle,
neuer Knoten bereits da, nur mehr Wert ändern*)
56     st^.ch := ch;
57 END ELSE BEGIN (*Fehler (i muss gleich 1+length(code) sein)*)
58     WriteLn('Error');
59     Exit;
60 END;
61 END;
62
63 FUNCTION Lookup(tree: MorseTreePtr; code: STRING): CHAR;
64 VAR
65     st: MorseTreePtr;
66     i: INTEGER;
67 BEGIN
68     i := 1;
69     st := tree;
70     WHILE (i <= length(code)) AND (st <> NIL) DO BEGIN
71         IF code[i] = '.' THEN BEGIN
72             st := st^.left;
73         END ELSE BEGIN
74             IF code[i] = '-' THEN BEGIN
75                 st := st^.right;
76             END ELSE BEGIN
77                 WriteLn('Error: invalid Code - Exit');
78                 Lookup := '?';
79                 Exit;
80             END;
81         END;
82         inc(i);
83     END;
84
85     IF st <> NIL THEN BEGIN
86         Lookup := st^.ch;
87     END ELSE BEGIN (* nicht enthalten*)

```

```

88     Lookup := '?';
89     END;
90 END;
91
92 FUNCTION ConvertMessage(tree: MorseTreePtr; mCode: STRING): STRING;
93     VAR
94         output, tmpCode: STRING;
95         i: INTEGER;
96 BEGIN
97     output := '';
98     IF (tree = NIL) OR (mCode = '') THEN BEGIN
99         ConvertMessage := 'Tree Empty or code missing';
100        Exit;
101    END;
102
103    i := 1;
104    tmpCode := '';
105    output := '';
106    WHILE i <= length(mCode) DO BEGIN
107        IF NOT (mCode[i] IN [' ', ';']) THEN BEGIN (*bei gültigem Code also
108            nur . oder -, ungültige Zeichen zu tmpCode hinzugefügt und nachher an Lookup
109            weitergegeben*)
110            tmpCode := tmpCode + mCode[i];
111            IF i = length(mCode) THEN BEGIN
112                output := output + Lookup(tree, tmpCode);
113            END;
114            END ELSE BEGIN (* bei gültigem code also nur ' ' oder ;*)
115                IF mCode[i] = ' ' THEN BEGIN
116                    IF tmpCode <> '' THEN BEGIN (*Fall, dass zuerst ';' und
117                        darauffolgend ' ' eingegeben wird, wird ausgeschlossen*)
118                        output := output + Lookup(tree, tmpCode);
119                        tmpCode := '';
120                    END;
121                END ELSE BEGIN
122                    IF mCode[i] = ';' THEN BEGIN
123                        IF tmpCode <> '' THEN BEGIN (*Fall, dass zuerst ' ' und
124                            darauffolgend ';' eingegeben wird, wird ausgeschlossen*)
125                            output := output + Lookup(tree, tmpCode) + ' ';
126                            tmpCode := '';
127                        END;
128                    END;
129                END;
130            END;
131            inc(i);
132        END;
133    END;

```

```

130     ConvertMessage := output;
131 END;
132
133 FUNCTION TreeValid(tree: MorseTreePtr): BOOLEAN; (*prüft, dass keine
Dummywerte mehr vorhanden sind*)
134 BEGIN
135     IF tree = NIL THEN BEGIN
136         TreeValid := TRUE;
137     END ELSE BEGIN
138         IF tree^.ch = ' ' THEN BEGIN
139             TreeValid := FALSE;
140         END ELSE BEGIN
141             TreeValid := TreeValid(tree^.left) AND TreeValid(tree^.right);
142         END;
143     END;
144 END;
145
146 PROCEDURE DisposeTree(VAR tree: MorseTreePtr);
147 BEGIN
148     IF tree <> NIL THEN BEGIN
149         DisposeTree(tree^.left);
150         DisposeTree(tree^.right);
151         Dispose(tree);
152         tree := NIL;
153     END;
154 END;
155
156 VAR
157     tree: MorseTreePtr;
158     mCode: STRING;
159
160 BEGIN
161     New(tree); (*root erstellen*)
162     tree^.right := NIL;
163     tree^.left := NIL;
164     tree^.ch := '/';
165
166     InsertMorseCode(tree, 'A', '.-');
167     InsertMorseCode(tree, 'B', '-...');
168     InsertMorseCode(tree, 'C', '-.-.');
169     InsertMorseCode(tree, 'D', '-..');
170     InsertMorseCode(tree, 'E', '.');
171     InsertMorseCode(tree, 'F', '..-.');
172     InsertMorseCode(tree, 'G', '--.');
173     InsertMorseCode(tree, 'H', '....');
174     InsertMorseCode(tree, 'I', '..');

```

```
175     InsertMorseCode(tree, 'J', '.---');
176     InsertMorseCode(tree, 'K', '-.-');
177     InsertMorseCode(tree, 'L', '.-..');
178     InsertMorseCode(tree, 'M', '--');
179     InsertMorseCode(tree, 'N', '-. ');
180     InsertMorseCode(tree, 'O', '---');
181     InsertMorseCode(tree, 'P', '.-- ');
182     InsertMorseCode(tree, 'Q', '--.-');
183     InsertMorseCode(tree, 'R', '-. ');
184     InsertMorseCode(tree, 'S', '...');
185     InsertMorseCode(tree, 'T', '- ');
186     InsertMorseCode(tree, 'U', '..-');
187     InsertMorseCode(tree, 'V', '...-');
188     InsertMorseCode(tree, 'W', '.--');
189     InsertMorseCode(tree, 'X', '-.-.-');
190     InsertMorseCode(tree, 'Y', '-.--');
191     InsertMorseCode(tree, 'Z', '--..');
192
193     IF TreeValid(tree) THEN BEGIN
194         Write('Enter a Code > ');
195         ReadLn(mCode);
196         WriteLn(ConvertMessage(tree, mCode));
197     END ELSE BEGIN
198         WriteLn('Tree invalid');
199     END;
200
201     DisposeTree(tree);
202
203 END.
```

3.3 Tests

Input	Beschreibung	Output
Standard	Allgemeiner Test	Enter a Code >;.... .- --- . - . -... --- SE HAGENBERG
-	leer lassen	Enter a Code > Tree Empty or code missing
leerer Baum (tree = nil)	Fehlermeldung muss kommen	Error: Root Node does not exist Error: Root Node does not exist Error: Root Node does not exist Enter a Code > ... Tree Empty or code missing
' ' folgt auf ;	Lookup wird nicht mit einem leeren Code aufgerufen	Enter a Code >;- --- . - . -... --- SE HAGENBERG
siehe Screenshot	gesuchter Knoten existiert im Baum nicht - liefert ein '?' für dasjenige Zeichen	Enter a Code >;.... .- --- . - . -... - - - SE HAGENBE?G
andere Zeichen als Code	Erwartet: Fehlermeldung + Lookup gibt ein '?' zurück, wenn ein Wert nicht gefunden wird	Enter a Code >;.... .- --- . -a -... --- Error: invalid Code - Exit SE HAGE?BERG
InsertMorseCode(tree, 'Z', '--...');	Hier ist beim Hinzufügen eines Buchstabens im Code ein Punkt zu viel. Es gibt also min einen Dummyknoten,	Tree invalid

Input	Beschreibung	Output
	der Baum ist invalid	