## Wunschlisten-Analysator

### Lösungsidee

Der Wunschlisten-Analysator liest so lange den Namen eines Kindes und ein Geschenk ein, bis der User eine leere Eingabe tätigt. Jede Eingabe wird an eine Funktion übergeben, die den Namen des Kindes und das Geschenk mithilfe der Standardfunktionen Pos() und Delete() in zwei variablen speichert. Diese Variablen werden dann mit einer dritten, die einen Zeiger auf die Wunschliste enthält, an die Funktion AddPresent() übergeben. Diese Funktion ist zur einfachen Wiederverwendung in einer Unit umgesetzt.

AddPresent() ruft zu Beginn eine Funktion auf, die so lange die Liste durchgeht, bis das übergebene Geschenk gefunden wurde oder feststeht, dass es noch nicht in der Liste existiert. Wurde das Geschenk gefunden, wird die Anzahl, wie oft das Geschenk gewünscht wurde, um eins erhöht. Dann sucht eine weitere Funktion nach dem Namen des Kindes, bis er gefunden wurde, oder das ende der Kinder-Liste erreicht wurde. Wird das Kind in der Liste nicht gefunden, wird eine neue Node erstellt und ans Ende der Kinder-Liste angehängt. Wurde das Geschenk nicht gefunden, wird eine neue Node erstellt und ans Ende der Wunschliste angehängt. Nach dem Anhängen wird dasselbe wie bei gefundenem Element ausgeführt, und das Kind, wenn noch nicht vorhanden, an die Liste angehängt.

Die Funktion WriteWishList() setzt den Wert des Ankerelements auf ‘‘. Eine Schleife gibt so lange jedes Element aus, bis das Ankerelement mit dem leeren Wert erreicht wird. Innerhalb der Schleife wird ausgegeben, wie oft jedes Geschenk gewünscht wurde, das Geschenk selbst und eine Liste aller Kinder, die sich dieses Geschenk gewünscht haben.

### 1.2.1. Pascal-Code Hauptprogramm

PROGRAM WishListAnalyzer;

  USES WishListUnit;

  var wl: WishList;

  PROCEDURE AddPresentToList(s: string);

    var childName, present: string;

    BEGIN (\* AddPresentToList \*)

      childName := s;

      Delete(childName, Pos(':', childName), Length(s) - Pos(':', childName) + 1);

      present := s;

      Delete(present, 1, Pos(' ', present));

      AddPresent(wl, childName, present);

    END; (\* AddPresentToList \*)

  var s: string;

BEGIN (\* WishListAnalyzer \*)

  NewWishList(wl);

  Write('Please enter "<Child>: <Present>": ');

  ReadLn(s);

  while s <> '' do begin

    AddPresentToList(s);

    Write('Please enter "<Child>: <Present>": ');

    ReadLn(s);

  end; (\* WHILE \*)

  WriteWishList(wl);

END. (\* WishListAnalyzer \*)

### 1.2.2 Pascal-Code Unit

UNIT WishListUnit;

INTERFACE

  TYPE

    WishNodePtr = ^WishNode;

    PersonNodePtr = ^PersonNode;

    WishNode = record

      prev, next: WishNodePtr;

      wish: string;

      n: integer; (\*number of occurences of the wish\*)

      children: PersonNodePtr;

      end; (\*RECORD\*)

    PersonNode = record

      next: PersonNodePtr;

      childName: string;

      end; (\* RECORD \*)

    WishList = WishNodePtr;

    PersonList = PersonNodePtr;

  PROCEDURE AddPresent(var wl: WishList; childName: string; present: string);

  PROCEDURE WriteWishList(wl: WishList);

  PROCEDURE NewWishList(var wl: WishList);

IMPLEMENTATION

  PROCEDURE WritePersonList(pl: PersonList);

    BEGIN (\* WritePersonList \*)

      while pl <> NIL do begin

        Write(pl^.childName, ' ');

        pl := pl^.next;

      end;

    END; (\* WritePersonList \*)

  PROCEDURE WriteWishList(wl: WishList);

    var cur: WishNodePtr;

    BEGIN

      wl^.wish := '';

      cur := wl^.next;

      while cur^.wish <> '' do begin

        Write(cur^.n, '\*', cur^.wish, ': ');

        WritePersonList(cur^.children);

        WriteLn();

        cur := cur^.next;

      end; (\* WHILE \*)

    END;

  PROCEDURE NewWishList(var wl: WishList);

    BEGIN (\* NewWishList \*)

      New(wl);

      wl^.wish := '';

      wl^.n := 0;

      wl^.children := NIL;

      wl^.next := wl;

      wl^.prev := wl;

    END; (\* NewWishList \*)

  FUNCTION FindWishNode(l: WishList; present: string): WishNodePtr;

    var node: WishNodePtr;

    BEGIN (\* FindWishNode \*)

      node := l^.next;

      l^.wish := present;

    if node <> NIL then begin

      while node^.wish <> present do

        node := node^.next;

      if node = l then

        FindWishNode := NIL

      else

        FindWishNode := node;

    end else

      FindWishNode := NIL;

    END; (\* FindWishNode \*)

  FUNCTION FindChildNode(l: PersonList; childName: string): PersonNodePtr;

    BEGIN (\* FindChildNode \*)

      while (l <> NIL) AND (l^.childName <> childName) do

        l := l^.next;

      FindChildNode := l;

    END; (\* FindChildNode \*)

  PROCEDURE AppendPersonList(var l: PersonList; n: PersonNodePtr);

    var cur: PersonNodePtr;

    BEGIN (\* AppendPersonList \*)

      if l = NIL then

        l := n

      else begin

        cur := l;

        while cur^.next <> nil do

          cur := cur^.next;

        cur^.next := n;

      end;

    END; (\* AppendPersonList \*)

  PROCEDURE NewPersonNode(var l: PersonList; childName: string);

    var node: PersonNodePtr;

    BEGIN (\* NewPersonNode \*)

      New(node);

      node^.childName := childName;

      node^.next := NIL;

      AppendPersonList(l, node);

    END; (\* NewPersonNode \*)

  PROCEDURE AppendWishList(var l: WishList; n: WishNodePtr);

    BEGIN (\* AppendWishList \*)

      n^.prev := l^.prev;

      n^.next := l;

      l^.prev^.next := n;

      l^.prev := n;

    END; (\* AppendWishList \*)

  PROCEDURE NewWishNode(var l: WishList; present: string);

    var node: WishNodePtr;

    BEGIN (\* NewWishNode \*)

      New(node);

      node^.wish := present;

      node^.n := 1;

      node^.prev := NIL;

      node^.next := NIL;

      AppendWishList(l, node);

    END; (\* NewWishNode \*)

  PROCEDURE AddPresent(var wl: WishList; childName: string; present: string);

    var presentNode: WishNodePtr;

        childNode: PersonNodePtr;

    BEGIN

      presentNode := FindWishNode(wl, present);

      if presentNode <> NIL then begin

        Inc(presentNode^.n);

        childNode := FindChildNode(presentNode^.children, childName);

        if childNode = NIL then begin

          NewPersonNode(presentNode^.children, childName);

        end; (\* IF \*)

      end else begin

        NewWishNode(wl, present);

        presentNode := FindWishNode(wl, present);

        if presentNode <> NIL then begin

          childNode := FindChildNode(presentNode^.children, childName);

          if childNode = NIL then begin

            NewPersonNode(presentNode^.children, childName);

          end; (\* IF \*)

        end; (\* IF \*)

      end; (\* IF \*)

    END;

BEGIN (\* WishListUnit \*)

END. (\* WishListUnit \*)

### 1.3 Testfälle

## Christbaumkerzen

### 2.1.1 Lösungsidee rekursiv

Die Funktion erhält als Parameter die Höhe des Baumes. Ist die Höhe 0, wird 0 zurückgegeben. Ist die Höhe 1, wird 1 zurückgegeben. Ist die Höhe größer als 1, wird 1 + 3 \* die Anzahl der Kerzen bei Höhe – 1 zurückgegeben.

### 2.1.2 Lösungsidee iterativ

Die Funktion erhält als Parameter die Höhe des Baumes. Ist die Höhe 0, wird 0 zurückgegeben. Ist die Höhe größer als 0, wird eine Schleife von 1 bis Höhe durchgegangen. Bei jedem durchlauf wird eine zuvor auf 0 instanziierte Summenvariable auf summe \* 3 + 1 gesetzt. Am Ende wird summe zurückgegeben.

### 2.2 Pascal-Code

PROGRAM Candles;

  FUNCTION CalculateCandles(height: integer): integer;

    BEGIN (\* CalculateCandles \*)

      if height > 1 then

        CalculateCandles := 3 \* CalculateCandles(height - 1) + 1

      else if height = 1 then

        CalculateCandles := 1

      else

        CalculateCandles := 0;

    END; (\* CalculateCandles \*)

  FUNCTION CalculateCandlesIterative(height: integer): integer;

    var sum, i: integer;

    BEGIN (\* CalculateCandlesIterative \*)

      if height > 0 then begin

        sum := 0;

        for i := 1 to height do begin

          sum := 1 + sum \* 3;

        end; (\* FOR \*)

      end else

        sum := 0;

      CalculateCandlesIterative := sum;

    END; (\* CalculateCandlesIterative \*)

BEGIN (\* Candles \*)

  WriteLn(CalculateCandles(0));

  WriteLn(CalculateCandles(1));

  WriteLn(CalculateCandles(2));

  WriteLn(CalculateCandles(3));

  WriteLn(CalculateCandles(4));

  WriteLn(CalculateCandles(5));

  WriteLn(CalculateCandles(6));

  WriteLn();

  WriteLn(CalculateCandlesIterative(0));

  WriteLn(CalculateCandlesIterative(1));

  WriteLn(CalculateCandlesIterative(2));

  WriteLn(CalculateCandlesIterative(3));

  WriteLn(CalculateCandlesIterative(4));

  WriteLn(CalculateCandlesIterative(5));

  WriteLn(CalculateCandlesIterative(6));

END. (\* Candles \*)

### 2.3 Testfälle

## 3. Zeichenfolge Spiegeln

### 3.1.1 Lösungsidee Spiegeln

Ist die Länge des an die Funktion Mirror() übergeben Zeichen 1, wird nur das Zeichen zurück gegeben. Ist die Länge größer 1, wird das letzte Zeichen der Zeichenfolge + die Zeichenfolge minus dem letzten Zeichen zurückgegeben. Die Zeichenfolge wird mit einer Funktion ShortenString() verkürzt. Hier wird das Zeichen an der Stelle Length(str) gelöscht und der verkürzte String zurückgegeben.

### 3.1.2 Lösungsidee Einlesen ohne zwischenspeichern

Die Prozedur ReadString liest zuerst ein Zeichen ein. Ist dieses Zeichen kein Zeilenumbruch, wird die Funktion wieder aufgerufen. Sobald der erste Umbruch eingelesen wird, wird jeder Aufruf beginnend mit dem neuesten abgearbeitet und das dort eingelesene Zeichen ausgegeben.

### 3.2 Pascal-Code

PROGRAM MirrorText;

  FUNCTION ShortenString(s: string): string;

    BEGIN (\* ShortenString \*)

      Delete(s, Length(s), 1);

      ShortenString := s;

    END; (\* ShortenString \*)

  FUNCTION Mirror(s: string): string;

    BEGIN (\* Mirror \*)

      if Length(s) > 1 then

        Mirror := s[Length(s)] + Mirror(ShortenString(s))

      else

        Mirror := s;

    END; (\* Mirror \*)

  PROCEDURE ReadString;

    var ch: char;

    BEGIN (\* ReadString \*)

      Read(ch);

      if ch <> Chr(13) then

        ReadString();

      Write(ch);

    END; (\* ReadString \*)

BEGIN (\* Mirror \*)

  WriteLn(Mirror('test'));

  WriteLn(Mirror('this is a test sentence'));

  WriteLn(Mirror('nailimixaM'));

  ReadString();

END. (\* Mirror \*)

### 3.3 Testfälle