

Songtext-Writer

Lösungsidee

Ein Element des Songtextes (Strophe, Zeile, Silbe) setzt sich aus einem kleineren Element zusammen.

z.B.

1. Silbe = Konsonant + Vokal
2. Zeile = Silbe * (2n+1)
3. Strophe = 3* Zeile
4. Songtext = 3* Strophe

Wenn nun eine Funktion die Produktion eines Elementes übernehmen würde und dieses zurückgeben würde, könnten andere Funktionen dieses ihrerseits zur Produktion von Elementen verwenden.

So kann man die Funktionen hierarchisch ineinander verschachteln. Schema:

```
Funktion Songwriter(){
    string Songtext;

    For (So oft wie Strophen im Songtext){
        Songtext += Strophe()
    }

    return Songtext;
}

Funktion Strophe(){
    string Strophentext;

    For (So oft wie Zeilen pro Strophe){
        Strophentext += Zeile();
    }

    return Strophentext;
}

Funktion Zeile(){
    string Zeilentext;
    string Grundsilbe = Silbe();

    For(So oft wie Silben in der Zeile){
        Zeilentext += Grundsilbe;
    }

    return Zeilentext;
}

Funktion Silbe(){
    return Konsonant + Vokal;
}
```

Umsetzung

Die Umsetzung erfolgte in der Programmiersprache C++.

Das Schema entspricht im Eigentlichen dem oben bereits beschriebenen ineinander Verschachteln von Funktionen.

Hinzu kommen noch viele Variablen, Arrays, Bedingungen und Zufallszahlen.

Zufallszahlen spielen im Programm eine besondere Rolle, da so sichergestellt werden kann, dass sich die produzierten Songtexte voneinander unterscheiden.

Ursprünglich habe ich das Programm in PHP - einer C++ ähnlichen serverseitigen Scriptsprache - geschrieben.

Dieses kann unter <http://www.tim-hollmann.de/BwInf/> erreicht werden, ist aber noch nicht ganz ausgereift (die Muster der Silben und Zeilenzahlen sind noch nicht enthalten).

Drei vom Programm erzeugte Songtexte

Anmerkung: Da laut Angabe pro Strophe nur eine sehr begrenzte Anzahl von Konsonanten und Vokalen möglich ist, kann es zu Häufungen von Grundsilben innerhalb einer Strophe kommen(siehe Bild 2 Strophe 1).

1)

```

C:\Users\Tim\Desktop\songwriter\Debug\songwriter.exe
Songtext-Writer zum 32.BwInf
Junioraufgabe 1
Copyright (c) by Tim Hollmann, 2013
=====

BU BU BUp di BU BU
BV BV BVp di BV BV
MV MV MUp di MV MV
BY BV BVp di BV BV
BU BU BUp di BU BU

XU XU XUp di XU XU
XU XU XUp di XU XU
XI XI XIp di XI XI
XU XU XUp di XU XU

CA CA CAp di CA CA
CO CO COp di CO CO
SO SO SOp di SO SO
CO CO COp di CO CO
CO CO COp di CO CO

RA RA RAp di RA RA
BE BE BEp di BE BE
RE RE REp di RE RE
BE BE BEp di BE BE

=====
Eingabe:(1=Neuer Songtext;alles Andere:Ende):

```

2)

```

C:\Users\Tim\Desktop\songwriter\Debug\songwriter.exe
Songtext-Writer zum 32.BwInf
Junioraufgabe 1
Copyright (c) by Tim Hollmann, 2013
=====

RU RU RU RUp di RU RU RU
RU RU RU RUp di RU RU RU
RU RU RU RUp di RU RU RU
RU RU RU RUp di RU RU RU

FI FI FI di FI FI
NO NO NOp di NO NO
FO FO FOp di FO FO
NO NO NOp di NO NO
yeah!

JA JA JA JAp di JA JA JA
RU RU RU RUp di RU RU RU
JU JU JU JUp di JU JU JU
JU JU JU JUp di JU JU JU

NA NA NAp di NA NA
NI NI NIp di NI NI
ZI ZI ZIp di ZI ZI
NI NI NIp di NI NI

=====
Eingabe:(1=Neuer Songtext;alles Andere:Ende):

```

3)

```

C:\Users\Tim\Desktop\songwriter\Debug\songwriter.exe
Songtext-Writer zum 32.BwInf
Junioraufgabe 1
Copyright (c) by Tim Hollmann, 2013
=====

XU XU XUp di XU XU
DA DA DAp di DA DA
XU XU XUp di XU XU
DA DA DAp di DA DA

BU BU BUp di BU BU
BU BU BUp di BU BU
BO BO BOp di BO BO
BU BU BUp di BU BU

RV RV RVp di RV RV
KV KV KVp di KV KV
KV KV KVp di KV KV
RV RV RVp di RV RV
yo man!

=====
Eingabe:(1=Neuer Songtext;alles Andere:Ende):

```

Wichtigste Teile des Quelltextes

Einbinden benötigter Header

```
#include <iostream> //Ein- und Ausgabe über die Konsole
#include <string> //String ist in diesem Programm sehr wichtig
#include <time.h> //Zeit(wird für den Zufallsgenerator benötigt)
#include <stdlib.h> //Zufallsgenerator

using namespace std; //sonst müsste man immer statt 'string' 'std::string' schreiben
```

Start bei der main()-Funktion

```
int main(void){
    setlocale(LC_ALL, "german"); //Codepage der Konsole ändern für Deutsche Umlaute

    while (true) {

        cout << songwriter(); //Aufruf der eigentlichen Songwriter-Variablen

        int eingabe;
        cin >> eingabe; // Entscheidungsmöglichkeit am Programmende (1= Neuer Songtext; != 1 = Ende)

        if (eingabe != 1){
            return 0; //Programmende bei Eingabe != 1; sonst (Eingabe = 1) neuer Songtext
        }
    }
}
```

Funktion songwriter()

```
string songwriter(void){
    srand( (unsigned) time(NULL) ); //pseudo-Zufallszahlen-Generator starten
    string songtext= ""; //Sammeln des Songtextes in dieser Variablen

    int strophenzahl = rand()%max_strophenzahl+2; //Zufällige Erzeugung der Strophenzahl

    //Mustererzeugung: da sie strophenübergreifend sind, müssen die Muster schon in songwriter() erzeugt werden
    int muster_zeilen = rand()%2 + 1; // das Muster der Zeilen wird zufällig erzeugt
    int muster_silben = rand()%2 +1; // das Muster der Silben wird ebenfalls zufällig erzeugt

    for(int x = 1; x<= strophenzahl; x++){//für jede Strophe
        int zeilenzahl = 0;
        int silbenzahl = 0;

        //Muster der Zeilen
        if (muster_zeilen == 1){
            zeilenzahl = 4;
        }else{
            if (x % 2 == 0){
                zeilenzahl = 4;
            }else{
                zeilenzahl = 5;
            }
        }

        //Muster der Silben
        if (muster_silben == 1){
            silbenzahl = 5;
        }else{
            if (x % 2 == 0){
                silbenzahl = 5;
            }else{
                silbenzahl = 7;
            }
        }

        //Neue Strophe an den Songtext anhängen (mit Aufruf der Funktion strophe())
        songtext = songtext + "\n\n\n" +strophe(zeilenzahl,silbenzahl);
    }

    return songtext; //Rückgabe des Songtextes
}
```

Funktion strophe()

```
string strophe( int zeilenanzahl, int silbenzahl){

    //[...] Konsonanten- und Vokalmenge definieren und ohne Doppelbefüllung füllen

    for (int x = 1; x <= zeilenanzahl; x++){ //Für jede Zeile
        strophentext = strophentext + "\n" + zeile(silbenzahl,menge_konsonanten,menge_vokale); //Erweitern des Strophentextes
    }

    if (rand()%100 +1 <= 50){ //Mit der Wahrscheinlichkeit von 50% wird ein zufälliger Call angehängt
        // Einen Call anhängen
        string Calls[3]; // Menge der Calls definieren

        //Menge füllen
        Calls[0] = "yeah!";
        Calls[1] = "yo man!";
        Calls[2] = "fake that!";

        strophentext += "\n" + Calls[rand()%2]; // Call anhängen
    }
```

```

    }else{
        //oder eben nicht
    }

    return strophentext; //Rückgabe des Strophentextes
}

```

Funktion zeile()

```

string zeile(int silbenzahl, string menge_konsonanten[2], string menge_vokale[2]){
    string zeilentext= "";

    string grundsilbe = silbe(menge_konsonanten, menge_vokale); // grundsilbe für die jeweilige Zeile erstellen

    string* temparray = new string[silbenzahl]; //temporäres Array erstellen, um darin die (gleichen) Grundsilben zu sammeln

    for (int x = 1; x <= silbenzahl; x++){ // Füllen des temporären Arrays
        temparray[x-1] = grundsilbe;
    }

    //Durch Verwendung eines Arrays kann auf die einzelnen Werte unabhängig voneinander zugegriffen werden
    //Die Ermittlung des Medians ist durch Verwendung eines Arrays ebenfalls einfach:
    temparray[int( silbenzahl /2)] += "p di"; //Anhängen von "p di" an den Median

    for (int x = 1; x <= silbenzahl; x++){ // Umwandlung des Arrays in einen String
        zeilentext += " " + temparray[x-1];
    }

    return zeilentext; // Rückgabe der Zeile
}

```

Funktion silbe()

```

string silbe(string menge_konsonanten[2], string menge_vokale[2]){
    return menge_konsonanten[int(rand()%3)] + menge_vokale[int(rand()%3)]; // Zusammensetzung und Rückgabe der Silbe
}

```

Dies sind nur die größten Bestandteile des Quelltextes; der original-Quelltext (im Ordner zusammen mit der ausführbaren .EXE-Datei) ist detaillierter

.EXE-Datei

Die .exe-Anwendung wurde unter Windows 8 64-Bit kompiliert und sollte auf jedem 64-Bit Windows lauffähig sein.

Für 32-Bit muss die Anwendung aus dem beiliegenden Quellcode selbst auf 32 Bit kompiliert werden, da mir kein 32-Bit Windows zur Verfügung steht.