Inf Programmieren 1 Übung: STL & GUI

Klaus Kusche

Wörter-Index

Wir wollen ein Programm schreiben, das zu einem Text einen *Index aller Wörter* aufbaut:
Für jedes Wort soll die *Liste aller Vorkommen* (Positionen) gespeichert werden.
Wir verpacken das Ganze in ein schönes GUI-Programm (siehe Demo!)
aufbauend auf unserem File-Viewer aus Übung 20 (**`viewer.cpp`**).

Versuche zuerst einmal, die *benötigten Klassen und Methoden in der Online-Doku*
zu finden. Wo das nicht gelingt: Mich fragen, ich helfe!

Logik:

- Zentrale Datenstruktur ist eine STL-**`map`**: Erste Komponente (Suchbegriff)
  jedes Elementes ist das jeweilige *Wort* (vom Typ **`wxString`**), zweite Komponente
  ist ein STL-**`vector`** von **`long`**-Werten (*Liste der Positionen* dieses Wortes im Text).

  Tipp: Mach dir für die **`map`** und den **`vector`** **`typedef`**'s!

  Unsere *Frame-Klasse* bekommt eine solche Map als *Member*.
- Wir erweitern unsere Frame-Klasse um *zwei private Methoden*:
  - Eine *baut den Index* auf. Das ist die komplexeste Methode des Programms,
    ich bin dabei wie folgt vorgegangen:
    - Als erstes wird der Inhalt der Map komplett gelöscht.
    - Dann holt man sich vom Textfenster die Anzahl der Zeilen,
      macht eine *Schleife über alle Zeilen*, und holt sich den Text
      aus dem Textfenster zeilenweise (dafür gibt es Methoden).
    - Jede Zeile geht man von links nach rechts durch und ermittelt
      Anfang und Ende des nächsten *Wortes*. Sobald man ein Wort
      gefunden hat, macht man daraus einen *eigenen* **`wxString`**.

      Achtung: **`wxString`**s sind *Unicode-Strings* in 16-bit-Darstellung,
      jedes Zeichen ist ein **`wchar_t`** statt ein **`char`**. Man braucht daher zur
      Prüfung auf Buchstaben **`iswalpha`** (Header **`cwctype`**) statt **`isalpha`**.
    - Das gefundene Wort wird in der Index-**`map`** *gesucht* und
      gegebenenfalls *eingefügt* (Tipp: Man braucht die beiden Fälle
      gar nicht unterscheiden: Der **`operator[]`** auf eine **`map`**
      mit dem zu suchenden Wert als Index liefert in jedem Fall
      das **`second`** des gesuchten Elementes: Wenn es noch nicht
      vorhanden ist, wird es gleich neu eingefügt).

      In beiden Fällen hängt man dann die *Position an den Positions*-**`vector`**
      dieses Wortes *hinten* an (in der Textfeld-Klasse gibt es Methoden, die
      zu Zeile und Spalte die Position im Text ausrechnen und umgekehrt,
      aber Achtung auf die unintuitive Reihenfolge der Parameter!).

    - Die andere private Methode ist nur eine Hilfsmethode, siehe unten.
- Weiters habe ich in unserem Haupt-Frame noch ein paar *private Member* deklariert (alle schön in der Init-Liste des Konstruktors auf **0** / **NULL** setzen!):
    - Einen *Pointer* **cur_list** auf einen Positions-**vector**:
      Wird *kein Suchergebnis* angezeigt, ist er **NULL**,
      sonst zeigt er auf den Positions-**vector** des *gerade angezeigten Suchwortes*.
    - Einen **int**, der angibt, *welche Position* aus diesem Positions-**vector** gerade angezeigt wird (Index der aktuellen Position im **vector**).
    - Und noch einen **int**, der die *Länge des aktuellen Suchwortes* enthält.

Oberfläche:

- Unser GUI bekommt über den bisherigen Buttons eine *zweite Zeile* (Hilfs-Sizer wie bei den unteren Buttons verwenden!) mit einem *Textfeld* (zur Eingabe des Suchwortes, Style-Option **wxTE_PROCESS_ENTER** beim Anlegen!) und *zwei Buttons* **Prev** und **Next** (voriges und nächstes Vorkommen, beide sollten im Konstruktor *deaktiviert* werden ("disable" = auf ausgegraut setzen), bis es wirklich etwas zu suchen gibt!).

  Im Unterschied zu den anderen Buttons brauchen wir in mehreren Methoden Pointer auf diese drei GUI-Elemente, also können das nicht lokale Variablen im Konstruktor sein, sondern müssen *Member-Variablen* sein, die in der Init-Liste des Konstruktors mittels **new** initialisiert werden.
- Weiters bekommt unser Hauptfenster eine *Statuszeile*.

Event-Handler:

Die Buttons bekommen wie gehabt **ID**'s, Event-Handler und entsprechende **Connect**'s. Da wir jetzt 2 Textfelder haben, bekommen auch diese **ID**'s. Das große mit dem File-Text bleibt ohne Events, das kleine zur Suchtext-Eingabe hat gleich zwei (beide sind wie bei den Buttons **wxCommandEvent**): **wxEVT_COMMAND_TEXT_UPDATED** kommt jedesmal, wenn sich der Text im Textfeld irgendwie ändert, und **wxEVT_COMMAND_TEXT_ENTER** kommt, wenn man die Texteingabe mit der Enter-Taste abschließt.

- Ein bestehender Event-Handler wird *erweitert*: Wenn links im Baum ein *neuer File selektiert* wird, wird zuerst die *Deselect-Hilfsmethode* aufgerufen und nach dem Laden des neuen Files der *Index* mit der entsprechenden Methode *neu erstellt*.
- Der einfachste Event-Handler ist der für *Text-Änderungen* im Such-Eingabefeld: Er ruft nur die *Deselect-Hilfsmethode* auf.
- Der komplizierteste Event-Handler ist der für das *Enter im Such-Eingabefeld*: Er holt sich den Text aus dem Eingabefeld und *sucht* ihn im Wort-Index.
    - Wird er *nicht gefunden*, wird eine entsprechende *Meldung* in der Statuszeile angezeigt und eine eventuell vorhandene *Selektion in Textfenster entfernt* (Selektion mit Position 0 und Länge 0 setzen).
    - Wird das Wort im Index gefunden, so
        - merkt man sich einen *Pointer* auf dessen Positions-**vector** in **cur_list**,

- setzt den *Index* der aktuell angezeigten Position auf **0**,
- merkt sich die *Länge* des Wortes,
- holt sich das **0**-te Element aus diesem Positions-**vector**,
  setzt an dieser Stelle im Textfenster eine *Selektion*,
  die so lange wie das Wort ist, und *scrollt* diese Position
  in den sichtbaren Bereich des Textfensters,
- *aktiviert* den **Next**-Button, falls es mehr als eine Position
  dieses Wortes gibt (Länge des Vektors),
- und zeigt in der *Statuszeile* die *Anzahl der Fundstellen* an
  (auch Länge des Vektors).

  Tipp dazu: In einen **wxString** kann man mit **<<** wie auf das
  Terminal schreiben, also z.B. auch *Zahlen in Text* verwandeln.

- Bleiben noch die beiden Event-Handler für den **Prev**- und **Next**-Button:
  Sie zählen den aktuellen *Index* in der Positionsliste um 1 rauf oder runter,
  markieren die Fundstelle im Textfenster mittels Selektion,
  und *scrollen* das Textfenster zur neuen Fundstelle.

  Weiters *aktivieren* sie den Button in Gegenrichtung und -- falls die neue Position die erste oder letzte ist -- *deaktivieren* den eigenen Button.

- Die *Deselect-Hilfsmethode* wird immer dann aufgerufen,
  wenn die aktuelle Suche nicht mehr zum Textfenster oder zum Suchtext passt,
  weil *eines der beiden geändert* wurde.

  Sie *deaktiviert* den **Prev**- und **Next**-Button,
  setzt das **cur_list**-*Pointermember* auf **NULL**,
  entfernt die Selektion im Textfenster,
  und setzt die Statuszeile auf den *Leerstring*.

  Das alles ist nur notwendig, wenn noch eine Suche aktiv ist,
  d.h. wenn **cur_list** ungleich **NULL** ist.

Zusatzaufgabe:

Die ganz Mutigen können noch versuchen, das Textfeld zur Eingabe des Suchwortes durch eine *Combo-Box* mit allen Worten im Index zu ersetzten
(d.h. die Index-**map** wird gleich nach dem Aufbau einmal durchlaufen,
alle ihre Worte werden als Auswahlmöglichkeit zur Combo-Box hinzugefügt).