Volldynamische C++ -Datentypen zur linguistischen Datenverarbeitung

G. Hotz, P. Molitor, W. Muth
Universität des Saarlandes

27. November 1990

Wir haben in C++ [1] zwei volldynamische Datentypen entwickelt: String und Wörterbuch. Um einen String anzulegen, muß dessen Größe nicht wie in C explizit angegeben werden; sie wird intern ermittelt und lediglich durch die Kapazität des verfügbaren Hauptspeichers beschränkt. Charakteristisch für diesen Datentyp ist seine dynamische Eigenschaft: der durch eine Variable belegte Speicherplatz wächst und schrumpft mit zu- und abnehmender Länge des Strings. Wird die Länge des Strings verkleinert, so wird weiterhin derselbe physikalische Speicher verwendet, bis die Ausnutzung dieses Bereichs unter 50 % fällt. Dann erst wird der Speicher für diese Variable wieder maßgeschneidert angelegt.

Ein Wörterbuch dient der Verwaltung von großen Datenmengen. Unter einem Wörterbuch kann man sich z.B. ein Englisch-Deutsch-Wörterbuch vorstellen, in dem etwa unter dem Schlüssel "give" als zugehöriger Eintrag die deutsche Übersetzung "geben" und weitere Attribute abgelegt sind. Als Schlüssel eines Wörterbuchs kann jeder String verwendet werden; die Einträge sind Records, dessen Komponenten ebenfalls dynamische Typen wie Strings enthalten können.

Neben internen Wörterbüchern, deren Daten sich am Programmende verflüchtigen, gibt es auch externe Wörterbücher. Sie sind mit Dateien verbunden, weil es zum einen bei großen Datenmengen nicht möglich ist, alle Daten im Hauptspeicher eines Rechners zu halten. Zum anderen überleben die Daten eines Wörterbuchs das Programmende durch die Kopplung mit einer Datei. Ein anderes Programm kann mit diesen Daten arbeiten, indem ein Wörterbuch mit der entsprechenden Datei verbunden wird. Die Wörterbücher sind ebenfalls volldynamisch, d.h. ihre Größe und damit die Anzahl ihrer Einträge wird lediglich durch die Kapazität des verfügbaren Speichers begrenzt. Wie bei den Strings erstreckt sich der Begriff der Dynamik auch auf die Ausnutzung des Speichers. Wörterbücher werden in Teilen von 2 kBytes = 1 Seite abgespeichert. Mit zunehmender Größe des Wörterbuchs nimmt die Zahl der Seiten zu; wird das Wörterbuch verkleinert, so wird es dann reorganisiert, wenn die Speicherplatzausnutzung unter 50% fällt.

Die beiden vorgestellten Datentypen eignen sich in besonderem Maße und in einfacher Weise, Anwendungen aus dem Bereich der linguistischen Datenver-

[1] Die Programmiersprache C++ wurde 1980 von Bjarne Stroustrup bei den Bell Laboratories von AT&T entwickelt.

arbeitung zu schreiben. Dies ermöglicht ein schnelles Prototyping, so daß Änderungswünsche des Benutzers in einem frühen Stadium erkannt und berücksichtigt werden können. Die bereitgestellten Operationen der Datentypen wurden am Bedarf der linguistischen Datenverarbeitung erprobt.

Operationen auf Strings

Die folgende Tabelle zeigt nun einige mögliche Stringoperationen:

Stringoperation	Erläuterung

string s;
string t("schatz");
string u(t);
	Deklaration und Initialisierung von Strings mit dem leeren Wort, einer Zeichenkette und einem String

s = "wort"; t = s;	Zuweisungen
u = s + t; s += "e";	Konkatenation von s und t s wird mit "e" konkateniert
u − = t; s = u − t;	u wird, falls möglich, um den Suffix t gekürzt und an s weitergereicht
s = 3 * t; u * = 2;	s wird die 3-malige Konkatenation von t zugewiesen u wird mit sich selbst konkateniert
s = − s;	s wird gespiegelt, aus "TON' wird "NOT"
s(3,5) = u(2,6) + t(5);	In s wird das 3. bis 5. Zeichen ausgeblendet und dafür der Wert der rechten Seite eingesetzt: die Konkatenation des 2. bis 6. Zeichens von u und des 5. Zeichens von t.

s.replace('f',u,t);
s.replace('l',u,t);
s.replace('a',u,t);
	Das erste ('f'), letzte ('l') oder alle ('a') Vorkommmen eines Musters u im String s werden durch t ersetzt.

s.pos(t)	Position des ersten Auftretens von t in s
cin≫s; cout≪s;	Eingabe von s Ausgabe von s
s == t , s!= t s > t , s <= t	Überprüfung von s und t auf Gleichheit, Ungleichheit sowie der lexikographischen

Ein Beispiel

Der vorgestellte Datentyp **string** eignet sich gut für linguistische Anwendungen, etwa den Test auf Palindrom; ein Palindrom, z.B. "otto", ergibt vorwärts und rückwärts gelesen denselben Sinn.
Diesen Test können wir leicht formulieren: if (s == −s).

Neben dem Typ **string**, der linguistischen Anwendungen dient, steht der Typ **Wörterbuch** bereit, mit dem Datenbankanwendungen ermöglicht werden. Die einfache Syntax des Datentyps **Wörterbuch** wollen wir im folgenden an einem Beispiel darlegen:
Übersetzer müssen für ihre Prüfungen viele Vokabeln lernen; sie wollen dabei wissen, welche Vokabeln sie gut und welche sie schlecht beherrschen.

```
struct eintrag {
  string vokabel;
  int nicht_gewußt;
};
main()
{ dictionary wb("SN");
  string s,t;
  eintrag* p = new eintrag;
  wb.connect("Deutsch_Englisch");        // verbindet wb mit Datei
  cin≫s;                                 //deutsches Wort einlesen
  while ( s != "0") {                    // "0" beendet Programm
    if ( wb.finde(s,p) ) {               // wb.finde(s,p) = 1 ⇔ s gefunden
      cin≫t;                             // englische Übersetzung einlesen
      if ( t == p → vokabel )
        p→nicht_gewußt =
         max( 0 , p→nicht_gewußt−1 );    //Antwort korrekt
      else
        p→nicht_gewußt++;                // Antwort falsch
      }
    else {
      p→nicht_gewußt = 0;                // neue Vokabel
      cin≫ p→vokabel;                    // Übersetzung einlesen
    }
    wb.trage_ein(s,p)                    // entweder nicht_gewußt aktuali-
                                         // sieren oder neue Vokabel eintragen
    cin≫s;                               //nächste Vokabel
  };
}
```

Erläuterung:
Zuerst deklarieren wir uns den Typ *eintrag*, der ins Wörterbuch eingetragen werden soll. Das Wörterbuch *wb* mit Typname *dictionary* bekommt als Parameter eine Zeichenkette mit, die die Typen der einzelnen Recordkomponenten angibt; dabei steht :
'S' für Strings
'N' für Zahlen, Pointer und char-Zeichen

Einen *eintrag* p* kann man nun ins Wörterbuch *wb* eintragen und daraus lesen. *wb* wird mit der Datei Deutsch_Englisch verbunden, in der die Daten des Wörterbuchs abgelegt werden. Ist die Eingabe "0", wird das Programm abgebrochen, ansonsten soll eine deutsche Vokabel eingegeben werden. Ist diese schon Schlüssel des Wörterbuchs, wird vom Benutzer die englische Übersetzung abgefragt; sie wird mit der tatsächlichen Übersetzung verglichen, die im Wörterbuch gespeichert ist. Daraufhin aktualisieren wir den Zähler *nicht_gewußt*. War die oben eingegebene deutsche Vokabel nicht im Wörterbuch enthalten, muß es um diesen Eintrag erweitert werden. In beiden Fällen erfolgt eine Eintragung ins Wörterbuch, entweder um den Zähler zu aktualisieren oder eine neue Vokabel mit ihrer Übersetzung hinzuzufügen.

Das zweite Beispiel zeigt die Möglichkeit eines Crash-Kurses auf, der vor einer Prüfung sinnvoll ist. Man will nur noch Vokabeln abfragen, die man in letzter Zeit oft falsch gemacht wurden.

```
struct eintrag {
  string vokabel;
  int nicht_gewußt;
};
main()
{ dictionary wb("SN");
  string s,t,u;
  int grenze;
  eintrag* p = new eintrag;
  wb.connect("Deutsch_Englisch");        // verbindet wb mit Datei
  s = wb.first(); u = wb.last();
  cin≫grenze;
  while ( s != u ) {                     // noch nicht am Ende
    wb.finde(s,p)
    if ( p→nicht_gewußt > grenze )       // Selektion
    { cout≪"Vokabel "+s+" übersetzen";
      cin≫t;
                                         // *** Programmtext wie vorhin, der
                                         // die eingegebene Übersetzung t mit
                                         // p→vokabel vergleicht und
                                         // p→nicht_gewußt aktualisiert
    };
      s = wb.succ(s);
  };
}
```

Erläuterung:
Wie im 1. Beispiel wird der Eintragstyp *eintrag* des Wörterbuchs definiert, das Wörterbuch *wb* angelegt und mit der Datei Deutsch_Englisch verbunden. In *s* halten wir den lexikographisch kleinsten und in *u* den größten Schlüssel des Wörterbuchs fest. Durch die Eingabe von *grenze* werden nur die Vokabeln abgefragt, die öfter als *grenze* mal falsch übersetzt wurden. Der Variablen *s* werden in der Schleife nacheinander alle Schlüssel des Wörterbuchs zugewiesen werden, angefangen vom lexikographisch kleinsten bis zum größten. Nun werden die oft (mehr als *grenze* mal) falsch übersetzten Vokabeln herausgefiltert; diese werden wie im 1. Beispiel abgefragt, die Antwort wird jeweils überprüft und der Eintrag *nicht_gewußt* aktualisiert.

Schlußbemerkung

Bei der Realisierung der Wörterbücher wurden in C geschriebene Teile aus dem Laufzeitsystems der Programmiersprache Comskee verwendet.
Comskee steht für COMputing and String KEEping language und stellt die dynamischen Datentypen **string, set** - eine Menge von Strings -, **sentence** - ein Satz - und **Wörterbuch** zur Verfügung. Comskee wurde im 1972 gegründeten SFB 100 "Elektronische Sprachforschung" an der Universität des Saarlandes entwickelt. Diese Programmiersprache ist mittlerweile für die Betriebssysteme BS2000, SINIX, und Berkley-UNIX und ATARI-TOS verfügbar.
Sie wurde erfolgreich zur Verwaltung von Bibliotheken (multilinguales Recherche-System), Thesauri, Archälogie-Datenbank, usw. eingesetzt.
Ihre besondere Stärke besteht einerseits in dem durch praktische linguistische Arbeiten motivierten Satz von Typen und Operationen und andererseits in der Beschränkung auf universell effizient realisierbare Strukturen. Comskee wird von der in Saarbrücken ansässigen Firma DIALOGIKA vertrieben.