Einführung in die Programmiersprachen C und C++

Markus Kirschmer, Fakultät für Mathematik

Übungsblatt 9

Hier ein Template, das die Bitkodierung der elementaren Datentypen ausgibt.

Die Funktion `bits()` kann jeden Typ `T` als zweites Argument `x` aufnehmen, dessen im Programm tatsächliche Größe wird zur Compilezeit durch den `sizeof(x)`-Aufruf festgestellt.

```
// bits.cc
// Darstellung der Bit-Codierung der verschiedenen elementaren Datentypen
// ist stark abhaengig von der Rechnerarchitektur!!

#include <iostream>
#include <typeinfo>    // fuer typeid
using namespace std;

template <class T>
void bits(ostream &os, const T& x) {
  char *p = (char *) &x;        // Wandle Adresse von x in char-Adresse
  int   k = sizeof(x);
  if (typeid(x) == typeid((long double)1.0)) k = 10;  // nur 10 Byte verwendet!
  // 32bit-Linux oder -m96bit-long-double:  12 Byte Speicher, 10 Byte Praezision
  // 64bit-Linux oder -m128bit-long-double: 16 Byte Speicher, 10 Byte Praezision
  for (int i = k-1; i >= 0; i--) {      // Durchlaufen der Bytes von x
    for (int c = 128; c > 0; c >>= 1)   // ein Byte bitweise ausgeben
      os << ( p[i] & c ? 1 : 0 );
    os << " ";                          // Trennung der Bytes
  }
  os << "\n";
}

int main() {
  char c;int i; long long ll;
  float f; double d; long double ld;
  // Ausgabe der Groessen
  cout << "sizeof(char)        =  " << sizeof(char)        << "\n";
  cout << "sizeof(int)         =  " << sizeof(int)         << "\n";
  cout << "sizeof(long long)   =  " << sizeof(long long)   << "\n";
  cout << "sizeof(float)       =  " << sizeof(float)       << "\n";
  cout << "sizeof(double)      =  " << sizeof(double)      << "\n";
  cout << "sizeof(long double) =  " << sizeof(long double) << "\n";
  while (1) {
    cout << " : "; cin >> ld;
    // Umwandlung per cast
    c  = (char) ld; i = (int) ld; ll = (long long) ld;
    f = (float) ld; d = (double) ld;
    // Ausgabe:
    bits(cout,c); bits(cout,i); bits(cout,ll);
    bits(cout,f); bits(cout,d); bits(cout,ld);
  }
}
```


Die *Floating Point Unit* hat eine 80-Bit-Architektur, daher die hohe Präzision bei `long double`.

Ganze Zahlen werden binär dargestellt:

Für jede natürliche Zahl n gibt es $e_i \in \{0,1\}$ mit $0 \le i < k$, so daß

$$n = e_{k-1}2^{k-1} + \cdots + e_1 2 + e_0 = e_k e_{k-1} \dots e_1 e_0$$

gilt. Diese Darstellung wird für die Kodierung der ganzen Zahltypen verwendet. Auf welcher Seite der Kodierung das höchst-signifikante Bit e_{k-1} erscheint, hängt von der Rechnerarchitektur ab.

Ist k die Anzahl der verwendeten Bits für einen Ganzzahltyp, so wird in der Regel der Bereich $0, \dots, 2^k-1$ für nicht-negative `unsigned`-Typen verwendet, der Bereich $-2^{k-1}, \dots, 2^{k-1}-1$ für vorzeichenbehaftete `signed`-Typen.

Dabei wird die gleiche additive Arithmetik verwendet: Zahlen mit höchstem Bit $e_{k-1} = 0$ (resp. $= 1$) werden als positiv (resp. negativ) verwendet. Dabei wird für $-n$ die Zahl n' mit der Eigenschaft $n + n' = 2^k$ genommen und das hier auftretende Bit bei der Position k unterdrückt, also gewissermaßen modulo 2^k gerechnet.

Reelle nicht negative Zahlen können geschrieben werden als

$$r = 2^a(e_1 2^{-1} + e_2 2^{-2} + \cdots + e_i 2^{-i} + \cdots)$$

mit $e_1 = 1$ – dies wird durch Wahl des Exponenten a erreicht – und $e_i \in \{0,1\}$.

Die Kodierung geschieht so, daß ein gewisser Bereich von Bits für den Exponenten a vorgesehen wird und ein anderer für die *Mantisse* $e_1 e_2 \dots e_k$, wobei die Entwicklung in der Position k gerundet wird.

Wieder wird ein Bit für das Vorzeichen von r verwendet 0 für positive, 1 für negative Zahlen, und oft, aber nicht immer, wird das Bit e_1 gar nicht kodiert, da es ohnehin 1 ist.

Die Kodierung des Exponenten von $r = 1$ wird als *Bias* bezeichnet, sie liegt in der Mitte des Darstellungsbereiches für die Exponenten.

Ausgabebeispiele:

```
sizeof(char)        =  1
sizeof(int)         =  4
sizeof(long long)   =  8
sizeof(float)       =  4
sizeof(double)      =  8
sizeof(long double) = 16
 : 1
00000001
00000000 00000000 00000000 00000001
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
00111111 10000000 00000000 00000000
00111111 11110000 00000000 00000000 00000000 00000000 00000000 00000000
00111111 11111111 10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Nur noch float:
 : .3333333333333333333333333333333333333333333333333333333333333333333333333333333333
00111110 10101010 10101010 10101011
00111111 11010101 01010101 01010101 01010101 01010101 01010101 01010101
00111111 11111101 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101011
```

Selbstreferentielle Strukturen und Klassen

Zum Ordnen einer Folge von Daten eignet sich die Struktur des Binärbaums, wie in Blatt 4 durch ein C-Programm demonstriert. Hier eine C++-Version.

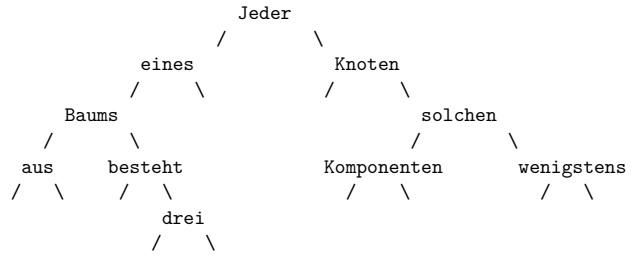
Zur Erinnerung (vgl. Blatt 4):

Jeder Knoten eines solchen Binärbaums besteht aus wenigstens drei Komponenten: den Daten, die zu diesem Knoten gehören sollen, und zwei Zeigern auf jeweils den linken und rechten Teilbaum zu diesem Knoten, in denen die Daten vorhanden sein sollen, die den Daten des aktuellen Knotens in der Ordnung (z. B. alphabetisch) vorangehen bz. nachfolgen. (Im vorliegenden Fall gibt es eine weitere Komponente vom Typ `int`, die zählt, wie häufig ein Wort gesehen wurde.)

Nehmen wir an, wir wollen die Wörter eines Textes auf diese Weise organisieren, z.B. die Wörter des Satzes:

"Jeder Knoten eines solchen Baums besteht aus wenigstens drei Komponenten"

Der Reihenfolge nach verarbeitet, erhalten wir folgende Struktur:



Das Vorgehen ist wie folgt: Zu einem gelesenen Wort wird ein Knoten konstruiert mit dem Wort als Dateneintrag und einem linken und rechten Zeiger auf einen Knoten gleichen Typs, die zunächst auf nichts (= `NULL`) zeigen.

Das nächste gelesene Wort wird mit dem ersten verglichen, geht es diesem in der (alphabetischen) Ordnung voran, wird es als linker Unter-Knoten eingetragen, folgt es in der Ordnung nach, wird es als rechter Unter-Knoten verwendet.

Jedes weitere Wort wird ähnlich behandelt: Es wird mit dem Wurzelknoten verglichen, geht es in der Ordnung diesem voran bzw. folgt es ihm nach, wird in den linken bzw. rechten Teilbaum gegangen und wieder mit dem dort stehenden Wort verglichen, und so wird fortgefahren, bis man zu einem Knoten kommt, an dem noch kein Wort steht. Dort wird es als Knoten eingetragen.

Mit einer Zählerkomponente bei jedem Knoten kann man die Häufigkeit der Wörter zählen.

Die so entstehende Struktur ist selbstreferentiell, weil jeder Knoten auf zwei Objekte *gleichen* Typs zeigt.

Eine geordnete Ausgabe erhält man durch die folgende rekursive Vorschrift, bezogen auf die einzelnen Knoten, mit Start an der Wurzel des Baums:

1. Ausgabe des linken Teilbaums, falls vorhanden.
2. Ausgabe des Wortes am aktuellen Knoten, falls vorhanden.
3. Ausgabe des rechten Teilbaums, falls vorhanden.

Eine C-Implementation findet man in K&R, 2. Auflage, Chapter 6, Sect. 6.5.

Wir geben hier eine C++-Klasse an, die das gleiche leistet wie das C-Programm von Blatt 4. Um deutsche Wörter inklusive Umlauten korrekt sortieren zu können, verwenden wir hier wieder die `locale`-Funktionen.


```
// btree.cc : Eine Binaerbaum-Klasse

#include <stdlib.h>
#include <stdio.h>
#include <wctype.h>
#include <locale>
#include <iostream>
#include <string>
#include <string.h>
using namespace std;

class tree {
  struct node{
    string e; int c; node *l, *r;   // Element, Zähler, linker/rechter Teilbaum
    node() { c = 1; l = NULL; r = NULL; }  // Konstruktor fuer struct in C++
  } *n;
public:
  tree() { n = NULL; }        // Konstruktor : fuer Deklarationen
  tree(node *nn) { n = nn;}   // Konstruktor : fuer Initialisierungen

  node* insert(const string &e) {  // Member-Fumktion :
    if (n == NULL) {
      n = new node;           // liefert und initialisiert struct node n
      n->e = e;               // Datenelement
    }
    else if (e == n->e) { n->c++; } // e vorhanden, erhoehe Zaehler
    // lexikographisch (deutsch, siehe `man locale`, unten: "setlocale"
    else if ( strcoll ( e.c_str(), n->e.c_str() ) < 0 )
    // Bemerkung: .c_str() ist eine member-Funktion des std-Datentyps string
    // die "den C-String" zurueckgibt, also einen 0-terminierten Array von char
      n->l = tree(n->l).insert(e);
    else
      n->r = tree(n->r).insert(e);
    return this->n;
  }
  friend ostream& inorder(ostream& os, tree t) {
    if (t.n) {
      inorder(os, tree(t.n->l) );
      os << t.n->e << '(' << t.n->c << ")\n" ;
      inorder(os, tree(t.n->r) );
    }
    return os;
  }
};

int main() {
  tree tr;                          // Deklariere Baum aus Strings
  setlocale(LC_ALL, "de_DE.UTF-8");  // cf. "man setlocale", "man locale"

  string line; char s[200];
  wchar_t w[200], wline[2000], *wlp, *wp;
  while (getline(cin,line)) {               // lies eine Zeile Text
    mbstowcs(wline,line.c_str(),2000);      // konvertiere nach wide char
    wlp = wline;
    while (*wlp) {                          // bis zum Zeilenende ...
```

```
      while (*wlp && !iswalpha(*wlp)) wlp++; // finde Wortanfang
      if (!*wlp) continue;                   // Zeile zu Ende
      wp = w;
      while (iswalpha(*wlp)) *wp++=*wlp++;   // kopiere Wort
      *wp = L'\0';                           // Wortende markieren
      wcstombs(s,w,200);                     // Wort nach UTF-8 konvertieren
      tr.insert(s);                          // in Baum einfuegen
    }
  }
  inorder(cout, tr);   // Ausgabe der sortierten Woerter
}
```

Die Klasse `tree` hat nur eine Komponente, einen Zeiger auf eine Struktur `node`. Diese Struktur nun trägt hier die oben genannten Komponenten und besitzt selbst eine Konstruktorfunktion.
Es gibt zwei Konstruktoren für `tree`, eine Member-Funktion `insert()` für das Einsetzen eines neuen Wortes, und eine Friend-Funktion `inorder()` zur (geordneten) Ausgabe.
In der Funktion `insert()` wird per `new node` Speicherplatz für ein Objekt vom Typ `node` allokiert. `new` ist C++-Schlüsselwort und liefert, aufgerufen für einen Datentyp, einen Zeiger auf diesen Typ, der auf freien Speicherplatz für eine Instanz dieses Typs zeigt. Hat der Typ einen Konstruktor ohne Argument, wird dieser automatisch dabei aufgerufen. `new` erfüllt also eine ähnliche Funktion in C++ wie `malloc()` für C (siehe `man malloc`), aber initialisiert den Speicher darüberhinaus nach Maßgabe eines Konstruktors. (Mit `delete`, angewandt auf durch `new` gewonnenen Speicher, kann man diesen wieder freigeben.)

Im weiteren Code bedeutet die Anweisung

```
n->l = tree(n->l).insert(e);
```

folgendes: Für den linken Teilbaum `tree(n->l)` wird die Member-Funktion `insert()` mit dem Argument `e` aufgerufen, Resultat ist ein neuer `node` (mit dem alten linken Teilbaum, um einen `node` mit `e` erweitert), dieser wird `n`-Komponente des neuen linken Teilbaums. (Genaugenommen wird hier nur dann eine echte Zuweisung vorgenommen, wenn der linke Teilbaum vorher leer war, also `n->l == NULL`.)

Schließlich wird der `node this->n` zurückgegeben, `this` (als Schlüsselwort in C++) bedeutet in einer Member-Funktion einen Zeiger auf das implizite Argument, also auf das Objekt selbst, zu dem die Member-Funktion gehört. `*this` ist also das Objekt – in diesem Fall der linke Teilbaum – selbst, mit der `n`-Komponente `this->n`.

Der Aufruf für das Programm kann – wenn der Name der kompilierten Datei `btree` ist – zum Beispiel geschehen durch

```
./btree < textdatei | less
```

wobei die Eingabe dann aus einer (auch sehr grossen) Textdatei genommen wird, deren einzelne Worte dann alphabetisch mit Häufigkeitsangabe ausgegeben wird.

Aufgabe 9.1. *Studieren Sie die Funktion des Unterprogramms* `getword` *in allen Einzelheiten, inclusive* `man isalpha`. *Wie werden hier Satzzeichen oder "White Space"aus der Textdatei behandelt?*

Aufgabe 9.2. *Erweitern Sie die Funktionalität, so dass zu jedem Knoten=Wort auch noch die Zeilenzahlen seines Auftretens festgehalten werden.*

Hier könnte eine weitere Strukturkomponente in der Struktur `struct node` *hilfreich sein: Ein weiterer String, der die Zeilennummern aufnimmt.*

Das Programm verwendet den Datentyp `string` aus der C++ Standardbibliothek, siehe hierzu

```
http://www.cplusplus.com/reference/string/
```


Kommandozeilen-Argumente werden in C++ wie in C behandelt.

Der Zugriff auf Dateien ist einfach: Durch den Befehl `ifstream from(argv[1]);` wird ein Objekt `from` vom Typ `ifstream` (Eingabestrom) definiert, über den die Zeichen der Datei `argv[1]` mit den üblichen Mitteln wie für `cin` (`cin.get(),...`) der Reihe nach gelesen werden können. Dabei ist `from` ein frei gewählter Name. Die Anweisung `ifstream from(argv[1]);` ist analog zu einer Deklaration mit Initialisierung wie etwa `int anzahl = 3;` zu sehen: `ifstream` ist ein Datentyp, `from` das Objekt mit frei gewähltem Namen (wie `anzahl`, das initialisiert wird auf die Datei namens `argv[1]`. Ganz analog arbeitet der Befehl `ofstream to(argv[2]);`, hier ist `to` das Objekt vom Typ `ofstream`, eine Datei, in die geschrieben wird.

```
// fileio.cc   Lesen und Schreiben von Dateien

#include <iostream>
#include <fstream>
#include <stdlib.h>
using namespace std;

void error(int i, char*  s, string t = "") {
  cerr << s << ' ' << t << '\n'; exit(i);
}

int main(int argc, char* argv[] ) {

  if (argc != 3) {
    cerr << "Falscher Aufruf, korrekter Aufruf:\n";
    error (1, argv[0], "Quelldatei Zieldatei");
  }

  ifstream from(argv[1]); // Deklaration eines ifstream namens from
                          // der aus der Datei namens argv[1] liest.
  if (!from) error(2, argv[1], "kann nicht zum Lesen geoeffnet werden.\n");

  ofstream to(argv[2]);
  if (!to) error(3, argv[2], "kann nicht zum Schreiben geoeffnet werden.\n");

  char c;

  while (from.get(c))
    to.put(c);

  from.close(); to.close();

  return 0;
}
```

Aufgabe 9.3. *Testen Sie das kompilierte Programm* `fileio`*, indem Sie Befehle wie*

```
fileio quelle ziel; echo $?
```

eingeben, wobei `"quelle"` *und* `"ziel"` *auch weggelassen werden bzw. durch Namen nicht existenter Dateien ersetzt werden. Die Status-Variable* `$?` *zeigt dann den durch* `exit` *zurückgegebenen* `int`*-Wert. Die Status-Variable* `$?` *gilt für die* `bash`*-Shell. Für* `csh` *oder* `tcsh` *lautet sie* `$status`.

Aufgabe 9.4. *Modifizieren Sie das Programm* `fileio.cc` *so, daß ein Aufruf des ausführbaren Programms* `fileio` *folgendes tut:*

`./fileio` *(ohne Parameter): Programm liest von der Tastatur (*`cin`*) und schreibt die gelesenen Daten auf den Bildschirm (nach* `cout`*).*

`./fileio quelle` *(ein Parameter): Liest aus Datei* `quelle` *und schreibt den Inhalt nach* `cout`*,*

`./fileio quelle ziel` *(zwei Parameter): Liest aus Datei* `quelle`*, schreibt den Inhalt in die Datei* `ziel`*,*

`./fileio quelle_1 ... quelle_n ziel` *(mehr als zwei Parameter): Liest der Reihe nach aus den Dateien* `quelle_1, ..., quelle_n` *und schreibt alles in die Datei* `ziel`*, die Dateien* `quelle_1, ..., quelle_n` *werden also aneinandergehängt.*

Aufgabe 9.5. *Modifizieren Sie* `btree.cc` *so, daß der Aufruf mit einem Argument*

```
btree datei
```

die Wörter aus der Datei `datei` *liest.*