Praxisorientierte Einführung in C++
Aufgabenblatt 3

Christof Elbrechter
celbrech@techfak.uni-bielefeld.de

6. Mai 2014

Aufgabe1: String-Suche (6 Punkte)

Anmerkung: Lösen Sie diese Aufgaben ohne Verwendung der std::string Klasse.

a. (3 Punkte) Schreiben Sie ein Programm, welches erlaubt, zwei Argumente zu übergeben. Das erste Argument soll als Heuhaufen (der Text in dem gesucht werden soll) benutzt werden und das zweite Argument als Nadel (der Suchterm). Das Programm soll ausgeben, an welcher Stelle im Heuhaufen die Nadel das erste Mal auftritt. Beispiel für einen solchen Aufruf:

```
> suche xyxyxyxyabcxyxyxy abc
Antwort: 8
```

b. (3 Punkte) Verwenden Sie die in (a) entwickelte Funktionalität, um eine Eingabedatei **zeilenweise** nach einem Suchterm zu durchsuchen. Die Ausgabe des Programms sollen exakt die Zeilen sein, in denen der Suchterm mindestens einmal gefunden worden ist. Das Programm erhält als Programmargument zunächst den Namen der zu durchsuchenden Datei und als zweites das zu suchende Wort. Zum lesen der Datei können Sie wahlweise einen C-Style `FILE*` oder einen `std::ifstream` (input file stream) verwenden. Nutzen sie das Internet, um herauszufinden, wie die Datei zeilenweise gelesen werden kann.

Aufgabe2: C-String Bibliothek

Implementieren Sie eine *kleine String Bibliothek*. Diese soll am Ende in ein "shared-object" libString.so übersetzt werden. Die Bibliothek soll (wie üblich) in zwei Dateien aufgeteilt werden: "String.h" und "String.cpp". "String.h" soll mittels `#pragma once` vor mehrfachem Einbinden geschützt werden. Wichtig ist, dass diese Aufgabe ohne Verwendung der `std::string`-Klasse und auch ohne Verwendung der Funktionen aus ¡cstring¿ bzw. ¡string.h¿ gelöst werden soll.

a. (3 Punkte) Legen Sie den Grundstein für die Bibliothek, indem Sie folgende Dateien anlegen und mit *ein wenig Basis-Inhalt* füllen:

   1. "String.h"
   2. "String.cpp"
   3. "main.cpp"
   4. "Makefile" (Bonus)

   Nach dem `pragma` soll in "String.h" zunächst ein `typedef` vorgenommen werden, um anstatt eines `char*` den Typenamen `String` verwenden zu können. Zusätzlich soll die erste (triviale) Funktion hinzugefügt werden:

```
void string_show(const String s);
```

   Erbringen Sie die Implementation dieser Funktion in "String.cpp" unter Verwendung des Headers `<iostream>`, welcher erst in "String.cpp" und nicht bereits in "String.h" eingebunden werden soll.
   Implementieren Sie zusätzlich "main.cpp". Hier soll "String.h" eingebunden werden und zum Testen die `string_show` Funktion verwendet werden. Geben Sie einfach mittels `string_show` das erste Kommandozeilenargument aus (falls eines übergeben wurde).

b. (2 Punkte + 3 Bonuspunkte) Übersetzen Sie ihre Bibliothek und erzeugen Sie `libString.so`. Beachten Sie bitte hier, dass "main.cpp" auf gar keinen Fall mit *in* das shared-object gelinkt werden darf, da das "main"-Symbol *in* einer Bibliothek nichts verloren hat. Sobald `libString.so` erzeugt ist, kann dann "main.cpp" übersetzt und gegen `libString.so` gelinkt werden. Das resultierende Programm/Binary soll "string" heißen. Dokumentieren Sie die notwendigen Kompiler/Linker Aufrufe falls sie nicht den Bonusteil lösen.

   Für die Bonuspunkte soll zusätzlich ein *Makefile* erstellt werden, welches die oben genannten Schritte ausführt. Das *Makefile* sollte mindestens folgende *Targets* enthalten:

   - `all` – *Baut* alles
   - `clean` – Löscht alle gebauten Dateien
   - `libString.so` – Kompiliert und linkt die Bibliothek
   - `string` – Kompiliert und Linkt das Programm

   Optional, können auch *generische Rules* verwendet werden um z.B. aus einem `.cpp` File ein `.o` File zu erzeugen.

c. (8 Punkte – 1 Punkt pro Funktion) Fügen Sie ihrer String-Bibliothek folgende Funktionen hinzu. Alle Funktionen, die einen String zurückgeben, sollen diesen mittels `new []` allozieren.

```
// gibt die Laenge des Strings aus (ohne 0-Terminator)
int string_length(const String s);

// kopiert den uebergebenen String
String string_copy(const String s);
```

```
// konkateniert 2 Strings
String string_concatenate(const String a, const String b);

// extrahiert einen substring aus s. Falls offset oder len ungueltig
// sind, soll 0 zurueckgegeben werden
String string_substring(const String s, int offset, int len);

// gibt aus, ob der Anfang von a gerade b entspricht
bool string_starts_with(const String a, const String b);

// gibt aus, ob das Ende von a gerade b entspricht
bool string_ends_with(const String a, const String b);

// gibt zurueck , ob a und b gleicht sind
bool string_compare(const String a, const String b);

// gibt den ersten offset zurueck an dem b in a vorkommt
// (oder -1) falls kein Match vorhanden
int string_find(const String a, const String b);
```

d. (2 Punkte) Implementieren Sie in "main.cpp" Tests, die Ihre Bibliothek testen und ihre Funktionsweise demonstrieren.

Aufgabe3: Funktionsüberladung (7 Punkte)

Hinweis: Dokumentation zu den weiter unten genannten Funktionen `atoi`, `atof` und `strcmp` finden Sie auf UNIX-Systemen leicht mit dem Programm "man":

```
> man atoi
```

Aber auch *google* könnte hilfreich sein.

a. (2 Punkte) Schreiben Sie drei Funktionen namens `parse` mit den folgenden Signaturen:

```
void parse(const char *, bool &);
void parse(const char *, int &);
void parse(const char *, float &);
```

Die Funktionen sollen den als erstes Argument übergebenen C-String untersuchen und in einen Wert vom Typ des zweiten Arguments übersetzen. Hierfür können Sie die Funktionen

```
int atoi(const char *nptr);
```

und

```
double atof(const char *nptr);
```

aus dem Header `<cstdlib>` benutzen. Für boolsche Werte sollen die Strings `"true"` und `"false"` unterschieden werden können.
Für den Vergleich zweier C-Strings können Sie die Funktion

```
int strcmp(const char *s1, const char *s2);
```

aus dem Header `<cstring>` benutzen. Das Ergebnis der Umwandlung soll dann durch überschreiben des zweiten Arguments zurückgegeben werden.
Falls bei einer der Teilaufgaben ein (parse)-Fehler auftritt, soll dieses durch eine entprechende Fehlermeldung, welche i.d.R nach `std::cerr` (auch in `<iostream>`) geschrieben wird, angezeigt werden. Das Ergebnis soll in diesem Fall auf 0 bzw. "false" gesetzt werden.

b. (2 Punkte) Schreiben Sie ein einfaches Programm, welches mithilfe der in Aufgabenteil a. definierten Funktionen das erste Kommandozeilenargument (also `argv[1]`) in Werte von allen drei Typen übersetzt. Geben Sie danach die übersetzten Werte mithilfe von `std::cout` auf der Konsole aus.

c. (3 Punkte) Spalten Sie das Programm aus Aufgabenteil b. in drei Dateien auf: "main.cpp", "parse.cpp" und "parse.h". "main.cpp" sollte dabei nur die Definition der Funktion `main` enthalten und `parse.h` mittels `#inlcude` einbinden. "parse.h" sollte die Funktionsdeklarationen für die drei `parse`-Funktionen aus Aufgabenteil a. enthalten und "parse.cpp" dementsprechend die Definitionen[1].

Aufgabe4: Kontrollstrukturen (7 Punkte)

Es soll eine Funktion `char to_upper(char c)` in unterschiedlichen Versionen implementiert werden. Die Funktion soll übergebene Kleinbuchstaben in Großbuchstaben umwandeln und alle anderen Buchstaben und Sonderzeichen direkt zurückgeben.

a. (2 Punkte) Implementieren Sie die Funktion `to_upper_switch`, welche ein `switch`-Statement verwendet, um zwischen den einzelnen Buchstaben zu unterscheiden. Benötigen Sie `break`-Anweisungen? Können Sie das Spezial-Label `default:` verwenden, um nicht über alle 256 möglichen `char`-Werte *switchen* zu müssen?

b. (2 Punkte) Implementieren Sie die Funktion `to_upper_sub`, welche zur Umwandlung von Klein- in Großbuchstaben die Tatsache ausnutzt, dass `'a'-'A'= 'b'-'B'= 'c'-'C'` usw. gilt.

c. (3 Punkte) Implementieren Sie die Funktion `to_upper_lut`. Diesmal soll eine statische Lookup-Table[2] (LUT) vewendet werden. Die LUT soll vom Typ `static const char[256]` sein und automatisch beim ersten Aufruf initialisiert werden. Die eigentliche Umwandlung kann nun durch einen simplen *table-lookup* erfolgen. Achten Sie beim LUT Zugriff darauf, dass `char` nicht zwangsweise `unsigned` ist.

[1] Falls Sie nicht `g++` versenden, so nutzen Sie die Dokumentation Ihres Compilers, um herauszufinden wie "main.cpp" und "parse.cpp" zu einem Programm übersetzt/gelinkt werden kann.

[2] http://en.wikipedia.org/wiki/Lookup_table