Praxisorientierte Einführung in C++
Aufgabenblatt 4

Christof Elbrechter
celbrech@techfak.uni-bielefeld.de

24. April 2014

Aufgabe1: std::strings

Stringverarbeitung mit den C-Style Strings (`char*` oder `char[]`) ist relativ kompliziert. Für bequeme Stringverarbeitung steht in der C++-Standard-Bibliothek eine spezielle String-Klasse zur Verfügung. Die Klasse kann mittels

```
#include <string>
```

eingebunden werden.
Hier ein paar Hinweise zur Verwendung:

```
std::string s;                  // erzeugt einen leeren String
std::string s2 = "Hallo";       // erzeugt einen String mit dem Inhalt "Hallo"
std::string s3("Welt");         // erzeugt einen String mit dem Inhalt "Welt"
std::string k = s2 + " " + s3; // verkettet die Strings s2 und " " und s3
char &c = s2[3];                // refereziert den 3. Buchstaben (hier 'l')
bool b = k.substr(0,5) == s2;   // muesste true sein!
```

Machen Sie sich mit der string Klasse und deren Funktionen vertraut. Nutzen sie hierfür beliebige Quellen z.B. (`http://www.cppreference.com/wiki/string/start`).

a. (2 Punkte) Schreiben Sie ein Programm, welches einen String über die Standardeingabe (`std::cin`) erwartet und dessen Länge ausgibt. Verwenden Sie hierfür die Klasse std::string.

b. (3 Punkte) Schreiben Sie ein Programm, welches 2 Strings als Programmargumente erwartet und **alle** Positionen ausgibt, an denen der zweite String auf den ersten "matched". Hierfür sollte die std::string Klasse verwendet werden.

Aufgabe2: Funktions-Templates

In Anlehnung an die in der Vorlesung vorgestellte `swap`-Funktion welche zwei Objekt-Instanzen vertauscht, sollen `shift`-Template-Funktionen implementiert werden.

a. (3 Punkte) Implementieren Sie das Funktionstemplate zunächst mit folgender Signatur:

```
template<class T> void shift(T *data, int len, int shiftDistance);
```

Die Funktion soll die Elemente des Arrays `data` (welches die Länge `len` hat), um `shiftDistance` Stellen nach links oder nach rechts verschieben. Ein negatives Vorzeichen bedeutet eine Verschiebung nach rechts. Das Verschieben soll zirkulär erfolgen. Hier einige Beispiele:

```
A = {1,2,3,4,5};
shift(A,5,-1)  -> {5,1,2,3,4}
shift(A,5,-2)  -> {4,5,1,2,3}
```

```
shift(A,5,1) -> {2,3,4,5,1}
shift(A,2,-1)  -> {2,1,3,4,5} :-)
```

Ist es möglich das Verschieben so zu implementieren, dass nur ein einziges temporäres Objekt – also kein ganzes Array der Länge `len` – notwendig ist?

b. (3 Punkte) Implementieren Sie das Funktionstemplate nun mit einer anderen Signatur:

```
template<class T, int shiftDistance> void shift(T *data, int len);
```

Die Funktionsweise soll analog sein. Was bedeutet die Tatsache, dass `shiftDistance` nun ein Template-Parameter ist, für die Art und Weise, wie die Funktion zu verwenden ist. Demonstrieren Sie die unterschiedliche Verwendung beider `shift`-Funktionen in einer `main`-Funktion welche zwei Programmargumente ausliest, und das *geshiftete* Wort ausgibt. Können tatsächlich beide Varianten hierfür verwendet werden?

```
#include <cstring>
#include <iostream>
#include <cstdlib>
int main(int n, char **args){
  if(n != 3) {
    std::cout << " Usage: shift WORD DISTANCE" << std::endl;
    exit(EXIT_FAILURE);
  }
  char *word = args[1];
  int len = strlen(word);
  int shiftDistance = atoi(args[2]);
  ...
}
```

Aufgabe3: Spezialisierung von Funktions-Templates

a. (5 Punkte) Reimplementieren Sie die Funktion `parse` aus Aufgabe 2 vom letzten Übungsblatt als Template-Funktion mit folgender Signatur:

```
template<typename T> T parse2(const std::string &text)
```

Da uns bis jetzt keine Funktion zur Verfügung steht, um aus einem String einen beliebigen Typen zu parsen, kann das allgemeine Template in folgender Weise implementiert werden:

```
template<typename T> T parse2(const std::string &text){
  return T(); // hier wird also irgendein T zurueckgebeben!
}
```

Diese Variante liefert einen beliebigen Wert zurück. Schreiben Sie Template-Spezialisierungen für die Typen aus der Aufgabe 2 (`float`, `int`, `bool`). Dafür kann der Header "parse.h" eingebunden werden. Hier ist eine Beispielimplementation (Bitte beachten Sie, dass sich die Signaturen im Vergleich zum letzten Übungsblatt verändert haben):

```
// parse.h
#include <cstdlib>
#include <cstring>

inline int parse_int(const char *s){ return atoi(s); }
inline bool parse_bool(const char *s){ return !strncmp(s,"true",4); }
inline float parse_float(const char *s){ return atof(s); }
```

Die Funktionen können dann innerhalb der Template-Spezialisierungen von parse2 verwendet werden. Zeigen Sie die Verwendung des Templates innerhalb einer `main()`-Funktion.
Anmerkung: Verwenden Sie für diese Aufgabe nur eine Datei (z.B. "03.cpp"). Darin sollte:

- Der Header "parse.h" eingebunden werden

- Das allgemeine Template inline deklariert und definiert werden
- Die Spezialisierungen `inline` deklariert und definiert werden
- Die Verwendung des Templates für `float`, `int` und `bool` innerhalb einer `main`-Funktion demonstriert werden.

Hinweis: Ein `std::string` kann mittels der Member-Funktion `c_str()` in einen const char* *umgewandelt* werden.

```
int toInt(const std::string &text){ return atoi(text.c_str());  }
```

Aufgabe4: Dynamische Speicherverwaltung

In dieser Aufgabe wollen wir ein wenig mehr über die Unterschiede von Stack und Heap Herausfinden:

a. (2 Punkte) Schreiben Sie ein Programm mit einer Funktion, die immer wieder 1024*1024 Bytes Speicher auf dem Stack alloziert, bis dieser voll ist. In jedem Schritt soll ein Zähler inkrementiert und ausgegeben werden, um herauszufinden, wieviel Speicher alloziert werden konnte. Gibt es einen Weg, dieses relativ kompakt zu lösen? Was passiert eigentlich, wenn der Stack *überläuft*?

b. (2 Punkte) Schreiben Sie nun ein Programm, welches immer wieder 1024*1024 Bytes Speicher auf dem Heap alloziert und dieses ausgibt. Um sicherzustellen, dass der Compiler sinnlose Allokationen nicht einfach weg-optimiert, sollten Sie alle Optimierungen abschalten (g++: -O0) und zusätzlich den jeweils allozierten Speicher mittels `memset` auf null setzen.
Wichtiger Hinweis: Wenn der RAM Speicher Ihres Systems knapp wird, wird automatisch statt RAM Speicher virtueller Speicher alloziert. Das Auslagen von Speicher auf die Festplatte kann Ihr System allerding so lahmlegen, dass sich nichteinmal mehr das Programm beenden lässt, welches ständig weiter Speicher alloziert. Um dieses zu verhinden, können Sie im Linux-System auf der Console mittels

```
ulimit -S -v 1000000
```

den maximalen Speicher für eine Applikation auf z.B. 1GB begrenzen. **Das sollten Sie unbedingt vorher machen!**. Was passiert, wenn der Heap voll ist?