Praxisorientierte Einführung in C++
Aufgabenblatt 1

Christof Elbrechter
celbrech@techfak.uni-bielefeld.de

19. April 2012

Aufgabe1: Variablen (2 Punkte)

a. Schreiben Sie ein Programm, welches drei Variablen vom Typ `int` deklariert. Die erste Variable soll nicht initialisiert werden, die zweite soll mit einer Anweisung der Form `int Bezeichner = Wert;` deklariert und initialisiert werden und die dritte soll mit der "Konstruktorschreibweise" deklariert und initialisiert werden. Zur Erinnerung:

```
Typ Bezeichner(Wert);
```

Anschließend soll das Programm den Inhalt der drei Variablen auf der Konsole ausgeben.
Führen Sie das Programm mehr als einmal aus. Führen Sie auch andere Programme auf dem Rechner aus. Fällt Ihnen etwas an den Werten der uninitialisierten Variablen auf? Wenn ja, was? Ändert sich etwas am Wert der uninitialisierten Variable, wenn Sie diese nicht im lokalen Scope der `main` Funktion, sonder im globalen Scope deklarieren?
Hinweis: Für die Ausgabe können Sie `std::cout` benutzen, wie Sie es schon im "Hello World"-Beispiel gesehen haben.

Aufgabe2: Schleifen, Bedingungen (3 Punkte)

a. Schreiben Sie ein Programm, welches alle Primzahlen aus dem Intervall $]0, 100]$ ausgibt. Das Programm soll alle Zahlen in einer einzelnen Zeile ausgeben und die einzelnen Zahlen mit einem einzigen Space voneinander trennen. Nach der letzten Zahl muss ein Zeilenumbruch ausgegeben werden. Verwenden Sie nicht den Algorithmus "Sieb des Eratosthenes", sondern die Brute-Force Methode, in der Sie für jede einzelne Zahl testen, ob sie eine Primzahl ist. Informieren Sie sich im Internet, wie der "Modulo-operator" in C++ verwendet wird.
Hinweise: Wie schon in der Vorlesung angedeutet, kann man eine `for`-Schleife z.B. folgendermaßen realisieren:

```
for (int i = 0; i < 100; ++i) {
        // Hier nimmt i nacheinander die Werte 0 bis
        // einschliesslich 99 an
```

```
}
```

Eine `if`-Abfrage ist ähnlich einfach. Seien `x` und `y` zwei Variablen vom Typ `int`, dann kann man diese z.B. folgendermaßen auf Gleichheit testen:

```
if (x == y) {
        // Die hier stehenden Anweisungen werden nur ausgefuehrt,
        // wenn x und y den gleichen Wert haben.
}
```

.

Aufgabe3: Makros (5 Punkte)

C++ verwendet einen sog. Präprozessor, welcher automatisch Textersetzungsregeln expandiert. Das wohl prominenteste Beispiel für die Verwendung des Präprozessors sind `#include`-Anweisungen, welche automatisch vom Präprozessor durch den Inhalt der entsprechenden Datei ersetzt werden. Tauchen darin weitere `#include`-Anweisungen auf, werden auch diese auf gleiche Weise aufgelöst usw. Zusätzlich können mit dem Präprozessor auch eigene (optional auch parametrisierte) Textersetzungsregeln (sog. Makros) definiert werden. Präprozessoranweisungen beginnen immer mit einem `#`-Symbol:

```
// Syntax Makrodefinition (parameter-liste ist optional)
#define MAKRO_NAME(PARAMETER-LISTE) EXPANDIERTE_VERSION

// Beispiel (parametrisiert)
#define SQUARE(X) (X*X)
// Beispiel (nicht parametrisiert)
#define NUM 10

// Anwedung: nicht parametrisierte Makros koennen
// wie Konstanten behandelt werden;
// parametrisierte Makros wie Funktionsaufrufe
int main(){
  int square_numbers[NUM];
  for(int i=0;i<NUM;++i){
    square_numbers[i] = SQUARE(i);
  }
}
```

Makros dienen zur dynamischen Code-Generierung und sollten auf jeden Fall sparsam verwendet werden. Daher ist diese Aufgabe mehr als Fingerübung zu verstehen. **Wichtig:** Bei einer Makrodefinition sollte immer *ausreichend* geklammert werden. In der Aufgabe soll insb. Punkt-vor-Strich-Rechnung bewahrt werden. Weitere Vor- und Nachteile werden in der nächsten Vorlesung diskutiert.

a. Definieren Sie die folgenden Makros:
   `ADD`, `SUBSTRACT`, `MULTIPLY` und `DIVIDE`,
   die jeweils zwei Argumente erwarten und die Grundrechenarten mithilfe der in die Sprache eingebauten Operatoren implementieren. Implementieren Sie mithilfe dieser

Makros die folgende mathematische Formel:

$$\sum_{i=0}^{27} \frac{i^2}{i+1} - \frac{1}{i^3+1} \tag{1}$$

Achten Sie darauf, dass z.B. auch `MULTIPLY(1+2, 3+4)` das richtige Ergebnis (21) liefert. Verifizieren Sie ihr Ergebnis ggf. mithilfe von "normalem" C++-Code.
Wichtig: Verwenden Sie `float` Variablen! Ein `int i` kann mit `(float)i` in einen `float` "gecastet" werden. Achten Sie auch darauf, dass Ganzzahl-Literale (z.B. in dem Ausdruck `1/4`) immer vom Typ `int` sind. Wenn `1/4` also nicht 0 ergeben soll, muss mindestens einer der Operanden vom Typ `float` sein (also: `1.0/4` oder `1/4.0` oder auch `((float)1)/4`.

b. Kommentieren Sie alle `#include`-Direktiven aus und finden Sie heraus, wie g++ nur den Präprozessor über ihren Code laufen lässt (Suchen Sie z.B. im Linux-Manual von g++ indem Sie `man g++` in der bash eingeben oder verwenden Sie google). Was ist die Ausgabe? Und sieht diese so aus wie Sie es erwartet hätten?

Aufgabe4: Parser (3 Punkte)

a. Schreiben Sie ein Programm, welches die Sprache $(aaa)^*$ akzeptiert. Das ist die Sprache, die aus Wörtern "", "aaa", "aaaaaa", "aaaaaaaaa", usw., besteht. Das zu parsende Wort soll als C-String-Konstante in `main()` deklariert werden.

```
int main() {
  const char *string = "aaabbbccc";

  // hier kommt dann ihr Code
}
```

Die Ausgabe des Programms sollte "OK" lauten, wenn das Wort akzeptiert wurde und "Nicht OK" falls nicht.