Objektorientierte Programmierung mit C++ (WS 2010)

Dr. Andreas F. Borchert, Tobias Brosch
Institut für Angewandte Informationsverarbeitung
Universität Ulm

Blatt 4: Abgabetermin 17. November 2010

1 Makefiles

In der Lösung zu Blatt 3 ist ein kleines `Makefile` enthalten. Ladet euch die Lösung von der Vorlesungshomepage und öffnet das Makefile im Editor Eurer Wahl. Ein `Makefile` wird von dem Programm `make` verwendet, um so genannte Ziele (oder englisch targets) zu generieren. Ein Ziel steht am Zeilenanfang gefolgt von einem Doppelpunkt. Nach dem Doppelpunkt werden Dateien aufgelistet, von welchen ein Ziel abhängt, d.h. ändert sich eine der Dateien von denen ein Ziel abhängt, wird das Ziel neu generiert.

Woher weiß `make`, wie es ein Ziel generiert? Nach der Zeile, in der das target und dessen Abhängigkeiten definiert wurden, kann nach einem Tab-Character '\t' (keine Leerzeichen!) ein beliebiges Shell-Kommando angegeben werden um das Ziel zu generieren (oder auch beliebige andere Kommandos die absolut unnütze Dinge machen → siehe Beispielmakefile).

Aufgabe: Macht euch mit dem Aufbau dieses `Basis-Makefiles` vertraut!

2 Fun With Arrays

Wie in Java oder C können auch in C++ arrays angelegt werden. Im Gegensatz zu Java kontrolliert allerdings niemand, ob wir versuchen außerhalb der Arraygrenzen zuzugreifen. Achtet also darauf, dies nicht zu tun (Verhalten undefiniert!). Wir werden uns zunächst auf non-dynamic Arrays beschränken, d.h. insbesondere, dass die Größenangabe des Arrays konstant sein muss!

```
// create a non-dynamic array of 10 doubles
// this requires the number within the brackets to be const!
// dynamic arrays can be created using new [] operator
// more on this later
const int n = 10;
double myArray[10]; // all elements not initialized!
double myArray[n]; // equivalent to above
// access elements or assign with operator[]:
double tmp = myArray[0]; // hard to tell the content of tmp! Why?
myArray[0] = 10;
```

```
tmp = myArray[0]; // now tmp = 10
for(int i = 0; i < 10; ++i)
        myArray[i] = i;
for(int i = 0; i < 10; ++i)
        std::cout << myArray[i] << std::endl;
// undefined - no exception, no compile error! Try to avoid!
std::cout << myArray[10] << std::endl;  // hard to find error!
// if you are lucky, you get a segmentation fault at runtime
```

Ein Array ist ein const-Pointer auf einen Speicherbereich, an dem nacheinander Daten vom spezifizierten Typ liegen. Beim Zugriff gibt die Zahl in eckigen Klammern den Offset (in Arrayelementgrößen) zum Beginn an. Dementsprechend kann jeder Zeiger ebenfalls mit dem `operator[]` dereferenziert werden:

```
double tmp(10);
double *tmpPointer = &tmp;
double tmp2 = tmpPointer[0]; // tmp = 10 = tmp2
tmp2 = tmpPointer[1]; // undefined and compiler won't complain
```

Wie übergibt man ein Array einer Funktion?

```
void myArrayFunction(double myArray[], int n);
// alternativ:
void myArrayFunction(double *const myArray, int n);
// note param "n"! otherwise function will not know size of array
```

Wie gibt man ein Array zurück? Gar nicht. Man übergibt ein Array mit non-const Datentyp und schreibt in das bereits allokierte Array:

```
// Warning: User must make sure that i and j are valid indices!
void swap(double myArray[], int i, int j) {
        double tmp = myArray[i];
        myArray[i] = myArray[j];
        myArray[j] = tmp;
}
void zeros(double a[], int n) {
   // iterator style: catch start address, i.e. a (begin-iterator)
   // get end-iterator: a + n is a pointer pointing to one beyond
   // last element! This is valid but don't try to dereference!
   // *(a + n) // error - undefined!
   for(double* it = a; it != a + n; ++it)
      *it = 0;
}
```

Aufgaben:

Achtet darauf, dass Alles was const sein kann, auch const deklariert wird!

1. Schreibt eine Funktion `dot`, die zwei double-Arrays übergeben bekommt und das Skalarprodukt der beiden zurückgibt. Verwendet hierbei eine `while`-Schleife.
2. Schreibt eine Funktion `sum`, die zwei double-Arrays übergeben bekommt und in das erste übergebene Array die Summe der beiden schreibt. Verwendet hierzu eine `for`-Schleife und die Iterator Variante!
3. Schreibt eine Testfunktion `testSumDot`, die ein Array der Länge `n` anlegt, die Werte auf 1 setzt und mit einer `if` Abfrage überprüft, ob `n == dot(myArray, myArray, n)` und eine entsprechende Meldung ausgibt. Anschließend soll `sum(myArray, myArray, n)` aufgerufen werden und überprüft werden, ob alle Werte in `myArray` anschließend gleich 2 sind. Im Fehlerfall soll eine Meldung ausgegeben werden.
4. Schreibt die Deklarationen der Operatoren und deren Test in eine Datei Namens `operators.h` und die Implementierungen in eine Datei Namens `operators.C`.
5. Schreibt eine Datei `main.C`, welche die Testfunktion aufruft.
6. Nehmt das Makefile aus der Musterlösung von Blatt 3 und schreibt nach dessen Vorlage ein Neues Makefile (ohne copy & paste), das euch das Testprogramm generiert. Achtet darauf, alle Abhängigkeiten richtig einzutragen.

Submitted eure Lösung mit

```
submit cpp 4 main.C operators.h operators.C Makefile
```

Viel Spaß!