Übung 1: Grundlagen

Zweck:

- *Eclipse* mit *Plugin CDT (C/C++ Development Tools)* kennenlernen
- Umgang mit *Zeiger* in C++ kennen
- Ein Beispiel einer *Klasse* analysieren und erweitern

Aufgabe 1: *Eclipse* mit *CDT*

Ziele:

- Installation Eclipse+*CDT*
- Ein C++-Projekt in *Eclipse/CDT erstellen*, *kompileren*, *linken* und *debuggen*.

Im Verzeichnis *a1_EclipseCDT* befindet sich die Datei *EclipseCDT.pdf* (auch hier als Beilage).
Darin wird die Installation, Konfiguration und Verifikation von Eclipse/CDT erläutert.
Diese Datei soll entsprechend durchgearbeitet werden.

Aufgabe 2: *Zeiger*

Es soll ein neues Eclipse-Projekt angelegt werden.

Nun sollen die Beispiele mit *Zeigern* gemäss Skript Abschnitte *Zeiger* und *Zeiger-Arithmetik* (Folien 20-22) durchgearbeitet werden.

Intialer Setup:

```
int i1 = 1;
int i2 = 2;
int i3 = 3;
...
int* p;
p = &i2;
...
int** pp;
pp = &p;
```

Dabei soll von der *Variablen* `i2` die *Adresse* und der *Inhalt*, von den *Zeigern* jeweils zusätzlich noch der *Inhalt wohin der Zeiger zeigt*, auf die Konsole ausgegeben werden:

```
Adresse der Variable 'i2'          = 0x??
Inhalt der Variable 'i2'           = ????

Adresse des Zeiger 'p'             = 0x??
Inhalt des Zeiger 'p'              = ????
Inhalt worauf Zeiger 'p' zeigt     = ????

Adresse des Zeiger 'pp'            = 0x??
Inhalt des Zeiger 'pp'             = ????
Inhalt worauf Zeiger 'pp' zeigt    = ????
Inhalt worauf jener Zeiger zeigt,
worauf Zeiger 'pp' zeigt  ;-)      = ????

p = p + 1 :
Adresse des Zeiger 'p'             = 0x??
Inhalt des Zeiger 'p'              = ????
Inhalt worauf Zeiger 'p' zeigt     = ????
```

Aufgabe 3: Swap und Maximum

Es soll eine Funktion implementiert werden, welche zwei Zahlen (`int`) miteinander austauscht (*swap'en*) und zusätzlich als `return`-Wert den grösseren der beiden Werte zurückgibt.

Beispiel:

```
int i1  = 1;
int i2  = 2;
int max = swapMax(i1, i2);
cout << max << endl;    // Ausgabe: 2
cout << i1 << endl;     // Ausgabe: 2
cout << i2 << endl;     // Ausgabe: 1
```


Aufgabe 4: Klasse *Bruch*

Bei dieser Aufgabe analysieren wir die Funktionsweise einer bestehender Klasse, wobei wir die Konzepte noch gar nicht wirklich kennen ;-)
Damit wollen wir herausfinden, ob C++-Programme intuitiv gestaltet werden können.

1. Studium der beigelegten Klasse *Bruch* (auf dem Papier, ohne Eclipse ! ;-) :
   - Bruch.h: Header-File der Klasse Bruch
   - Bruch.cpp: Implementations-File der Klasse Bruch
   - BruchAppl.cpp: Anwendungsprogramm welches die Klasse Bruch benutzt

   Welche Diskrepanzen bestehen zwischen Header- und Implementationsfile?

2. Analyse des Anwendungsprogrammes *BruchAppl.cpp* :
   Es soll auf einem Blatt Papier die erwarteten Ausgaben des Programmes auf die Konsole notiert werden (ohne Eclipse ;-)

3. Wieso könnte die letzte Anweisung in *BruchAppl.cpp* (`bruch2 * 4`) wohl möglich sein?

4. Erstellen eines Arbeits-Verzeichnis.
   Kopieren von *`Bruch.h, Bruch.cpp`*, *`BruchAppl.cpp`* und *`Makefile`* von Moodle dorthin.

5. Erstellen eines C++-Projektes in Eclipse für diese Dateien:
   Bei diesem Projekt wird das sog. *Makefile* mitgeliefert (meint, dass EclipseCDT dieses nicht selbst erstellen muss/soll):
   Menü: *File* > *New* > *Project...* > *C/C++* > *Makefile Project with Existing Code*
   Toolchain for Indexer Settings: *Cygwin GCC* (für Windows/Cygwin)

6. Welche Differenzen sind vorhanden gegenüber obigen Notizen von Punkt 2 wenn die Applikation zur Ausführung gebracht wird?

7. Es sollen die fehlenden Operatoren für *Division*, *Addition* und *Subtraktion* implementiert werden.

8. Das Anwendungsprogramm *BruchAppl.cpp* soll entsprechend erweitert werden um die neuen Operatoren zu testen.

9. Es soll ein zusätzlicher Operator für die Berechnung des Kehrwertes (als *operator!()*) erstellt und getestet werden.

10. Es sollen weiter Funktionalitäten nach eigenen Ideen implementiert werden (z.B. Kürzen des Bruches).

Beilagen:
- Eclipse/CDT
- Bruch.h
- Bruch.cpp
- BruchAppl.cpp

