Übung 2

Default-Argument, Überladen von Funktionen
Schlüsselwort Const, Referenzen

Aufgabe 1 (Default-Argumente)

Welche Ausgabe erhält man für das folgende C++-Programm?

```
#include <iostream>
 using namespace std;

void paul(int a = 17, int b = 4)
{
  cout << "a = " << a;
  cout << "  b = " << b << endl;
}

int main()
{
  paul();
  paul(20);
  paul(20,1);
  paul(,1);
  return 0;
}
```

Aufgabe 2 (Überladen von Funktionen, Default-Argumente)

In einem Programm ist die Funktion in folgenden Signaturen vereinbart:

(1) `int ohjeh (long i, double r, int j = 3);`

(2) `int ohjeh (int i, long j, double r);`

(3) `int ohjeh (int i, int j);`

Bei der Zuordnung eines Aufrufs zu einer Funktionsdefinition verwendet der Compiler folgendes Schema:

1. Exakte Übereinstimmung: *keine Umwandlung*.
2. Umwandlung von `char` und `short` nach `int` (integrale Erweiterung) sowie von `float` nach `double`

3. Standard-Umwandlungen wie Sie auch in C üblich sind (u.U. in Verbindung mit Informationsverlust, z.B. `int` nach `float` oder `long` nach `int`).

Zum Aufruf kommt die Funktion, welche nach der angegebenen Aufstellung die niedrigstrangige Umformung erfordert.

Welche Funktion wird bei folgenden Aufrufen ausgeführt bzw. welcher Aufruf ist mehrdeutig?

a) `ohjeh (1, 'b');`

b) `ohjeh (1, 1.0, 1);`

c) `ohjeh (2.3, 2.3);`

d) `ohjeh (2.3, 1L, 1);`

Überprüfen Sie Ihre Vermutung durch eine einfache Implementierung der überladenen Funktion ojeh!

```
void ojeh( ....)
{
  cout << ...
}
```

Aufgabe 3 (Schlüsselwort `const`)

In der Definition	ist `name` ...
`const type name = value;`	Konstante vom Typ `type`
`type * const name = value;`	konstanter Zeiger auf eine Variable vom Typ `type`
`type const *name = value;`	(variabler) Zeiger auf eine Konstante vom Typ `type`
`type const * const name = value;`	konstanter Zeiger auf eine Konstante vom Typ `type`

Welche Bedeutung haben die folgenden Definitionen:

```
int i;
int *ip;
int * const cp = &i;
int const ci = 7;
int const *cip;
int const * const cicp = &ci;
```

Welche der folgenden Wertzuweisungen sind zulässig, welche nicht?

```
i = ci;
ci = 8;
*cp = ci;
cp = &ci;
cip = &ci;
cip = cicp;
*cip = 7;
ip = cip;
```

Aufgabe 4 (Referenzen)

Schreiben Sie eine Funktion `swapi`, die zwei Zeiger auf `int`-Werte vertauscht. Verwenden Sie dazu Referenzen.

Aufgabe 5 (Referenzen)

Gegeben sei das Strukturmuster

```
typedef struct bar_t {
     int x;
     int y;
} bar_t;
```

und folgende Funktions-Prototypen:

```
void foo(bar_t& b); // setzt den Wert von x und y auf 1 und 2
void foo(bar_t* b); // setzt den Wert von x und y auf 1 und 2
```

Implementieren Sie die beiden Funktionen.
Erstellen sie eine main-Methode, in der eine Strukturvariable vom Typ `bar_t` initialisiert wird und wenden Sie anschließend die von ihnen erstellten Funktionen auf diese Strukurvariable an.

Welche der beiden Funktionen lässt sich unter C nicht kompilieren? Warum?
Welche Funktion halten Sie für einfacher und warum?