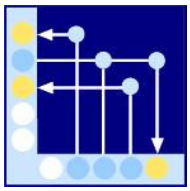
Hochschule Aalen
Fakultät Elektronik und Informatik
Studiengang Informatik



Programmieren in C++

Vorlesung im Wintersemester 2017/2018
Prof. Dr. habil. Christian Heinlein

2. Übungsblatt (23. Oktober 2017)

Aufgabe 3: Deklarationen

a) Was bedeuten die folgenden Deklarationen?
Geben Sie jeweils eine mögliche Initialisierung an!

```
int*& a;
const int*& b;
int const*& c;
int* const& d;
```

✎
Hilfsdeklarationen für die Initialisierungen:

```
// Zeiger auf int.
int* x;

// Zeiger auf ein konstantes int-Objekt.
const int* y;
```

Eigentliche Deklarationen:

```
// Referenz auf einen Zeiger auf int.
// Initialisierung mit einem Zeiger auf int (L-Wert).
int*& a = x;

// Referenz auf einen Zeiger auf ein konstantes int-Objekt.
// Initialisierung mit einem Zeiger auf ein konstantes int-Objekt (L-Wert).
const int*& b = y;
int const*& c = y;

// Referenz auf einen konstanten Zeiger auf int.
// Initialisierung mit einem Zeiger auf int (L- oder R-Wert).
int* const& d = x;
```

✎

b) Deklarieren Sie folgende Objekte:

- ein Feld mit 10 Zeigern auf `int`;
- eine Referenz auf ein solches Feld;
- einen Zeiger auf ein Feld mit 10 `int`s;

- einen Zeiger auf eine Funktion mit Parameter- und Resultattyp `int`;
- ein Feld mit 10 solchen Zeigern.
- eine Referenz auf ein solches Feld.

✎

```
// Feld mit 10 Zeigern auf int.
int* a [10];

// Referenz auf ein solches Feld (mit Initialisierung).
int* (&b) [10] = a;

// Zeiger auf ein Feld mit 10 int-Objekten.
int (*c) [10];

// Zeiger auf eine Funktion mit Parameter- und Resultattyp int.
int (*d) (int);

// Feld mit 10 solchen Zeigern.
int (*e[10]) (int);

// Referenz auf ein solches Feld (mit Initialisierung).
int (*(&f)[10]) (int) = e;
```

Hinweis: Die postfix angewandten Typkonstruktoren `[]` und `()` zur Deklaration von Feldern bzw. Funktionen haben Vorrang vor den präfix angewandten Typkonstruktoren `*` und `&` zur Deklaration von Zeigern bzw. Referenzen. Durch explizite Klammerung kann man den Vorrang jedoch explizit verändern.

Um die Lesbarkeit komplexer Deklarationen zu verbessern, kann man sie schrittweise mittels `typedef` aufbauen, z. B.:

```
// Zeiger auf eine Funktion mit Parameter- und Resultattyp int.
typedef int (*D) (int);
D d;

// Feld mit 10 solchen Zeigern.
typedef D E [10];
E e;

// Referenz auf ein solches Feld (mit Initialisierung).
typedef E& F;
F f = e;
```

✐

Aufgabe 4: Konstante Zeiger und Zeiger auf konstante Objekte

Gegeben seien die folgenden Deklarationen (vgl. §2.4.2):

```
// Gewöhnlicher Zeiger.
char* p;

// Zeiger auf konstantes Objekt.
const char* pc;
// Oder:
char const* pc;

// Konstanter Zeiger.
char* const cp = ...;

// Konstanter Zeiger auf konstantes Objekt.
const char* const cpc = ...;
// Oder:
char const* const cpc = ...;
```

Welche der folgenden Zuweisungen sind zulässig? Begründen Sie Ihre Antwort!

✎

l = r	p	pc	cp	cpc
p	+	−	+	−
pc	+	+	+	+
cp	−	−	−	−
cpc	−	−	−	−

Zuweisungen an `p`: Da `*p` nicht konstant ist, darf man keinen Zeiger `q` zuweisen, für den `*q` konstant ist.

Zuweisungen an `pc`: Da `*pc` konstant ist, darf man beliebige Zeiger `q` zuweisen.

Zuweisungen an `cp` und `cpc`: Da diese Variablen konstant sind, darf man grundsätzlich nicht zuweisen.
Zur *Initialisierung* von `cp` könnte man `p` (und `cp`) verwenden, zur Initialisierung von `cpc` jeden anderen Zeiger.

✎

Aufgabe 5: Felder und Zeiger

Implementieren Sie die Bibliotheksfunktionen `strlen`, `strcpy`, `strcat` und `strdup`, die in der Definitionsdatei `<cstring>` definiert sind!

Hinweis: Je nachdem, ob man die Parameter der Funktionen direkt als Zeiger oder aber als Felder interpretiert bzw. verwendet, erhält man unterschiedlich kompakte Implementierungen.

✎

Feldbasierte Implementierung

```
#include <cstdlib> // size_t

// Länge der Zeichenkette s bestimmen.
size_t strlen (const char* s) {
    int i = 0;
    while (s[i] != '\0') i = i + 1;
    return i;
}

// Zeichenkette src nach dest kopieren.
char* strcpy (char* dest, const char* src) {
    int i = 0;
    while (src[i] != '\0') {
        dest[i] = src[i];
        i = i + 1;
    }
    dest[i] = '\0';
    return dest;
}

// Zeichenkette src am Ende von dest anfügen.
char* strcat (char* dest, const char* src) {
    int i = 0, j = 0;
    while (dest[i] != '\0') i = i + 1;
    while (src[j] != '\0') {
        dest[i] = src[j];
        i = i + 1;
        j = j + 1;
    }
    dest[i] = '\0';
    return dest;
}

// Dynamische Kopie der Zeichenkette s erstellen.
char* strdup (const char* s) {
    int n = strlen(s) + 1;
    char* p = new char [n];
    return strcpy(p, s);
}
```

Zeigerbasierte (und kompaktierte) Implementierung

```
#include <cstdlib> // size_t

// Länge der Zeichenkette s bestimmen.
size_t strlen (const char* s) {
    const char* p = s;
    while (*s) s++;
    return s - p;
}
```

```
// Zeichenkette src nach dest kopieren.
char* strcpy (char* dest, const char* src) {
    char* p = dest;
    while (*dest++ = *src++) ;
    return p;
}

// Zeichenkette src am Ende von dest anfügen.
char* strcat (char* dest, const char* src) {
    char* p = dest;
    while (*dest) dest++;
    while (*dest++ = *src++) ;
    return p;
}

// Dynamische Kopie der Zeichenkette s erstellen.
char* strdup (const char* s) {
    return strcpy(new char [strlen(s) + 1], s);
}
```

Anmerkung

Wenn `new` keinen dynamischen Speicherplatz beschaffen kann, wird eine Ausnahme des Typs `bad_alloc` geworfen. In diesem Fall bricht `strdup` mit dieser Ausnahme ab, anstatt – wie im Handbuch beschrieben – einen Nullzeiger zurückzuliefern.

✐

Aufgabe 6: Referenzen

Lösen Sie die folgenden „Referenz-Puzzles“, d. h. sagen Sie die Ausgabe der Programme voraus, bevor Sie sie ausprobieren!

Teilaufgabe 6.a)

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

int& f (int& x, int& y) {
    return x < y ? x : y;
}

int main () {
    int a = 1, b = 2;
    f(a, b) = 3;
    f(a, b) = 4;
    cout << a << " " << b << endl;
}
```

✎

Die Ausgabe lautet: `3 4`

✐

Teilaufgabe 6.b)

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

int x = 1;

void f (int& y) {
    x += y;
    y += x;
}

int main () {
    f(x);
    cout << x << endl;
}
```

✎
Die Ausgabe lautet: `4`
✐

Teilaufgabe 6.c)

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

int a [] = { 1, 2, 3 };
int* p = a + 3;

void f (int& x, int& y) {
    x = x + y;
    y = x - y;
    x = x - y;
}

int main () {
    f(*a, a[1]);
    f(a[2], p[-2]);
    cout << a[0] << " " << a[1] << " " << a[2] << endl;
}
```

✎
Die Ausgabe lautet: `2 3 1`
✐

Teilaufgabe 6.d)

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

int& swap (int& x, int& y) {
    int z = x; x = y; y = z;
    return x;
}

int main () {
    int a = 1, b = 2, c = 3, d = 4;
    swap(a, swap(b, swap(c, d)));
    cout << a << " " << b << " " << c << " " << d << endl;
}
```

✎
Die Ausgabe lautet: `4 1 2 3`
✐