

Übung: C Deklarationen

Inhalt

Übung: C Deklarationen	1
1 Einführung.....	1
2 Lernziele	1
3 Übungen	1
3.1 Bestimmen Sie die Bedeutung folgender Deklarationen.....	1
3.2 Was ist die Bedeutung der Namen.....	2
3.3 Schreiben Sie folgende Deklarationen	2
4 Theorie.....	4
4.1 Wozu Deklarationen.....	4
4.2 Alias Typ Deklarationen	4
4.3 Grundform von Funktions- und Variablen Deklarationen	5
4.4 Syntax Regeln von Deklaratoren.....	5
4.5 Spezialfälle.....	6
const/volatile im Zusammenhang mit Pointer Alias.....	6

1 Einführung

In dieser Übung wird die Syntax der C Deklarationen angewendet.

2 Lernziele

- Sie können C Deklarationen lesen
- sie können C Deklarationen schreiben

3 Übungen

3.1 Bestimmen Sie die Bedeutung folgender Deklarationen

Konsultieren Sie die Theorie in Abschnitt 4.

```
double *a;
```

declare a as a pointer to double

```
double *a[5];
```

declare a as an array 5 of pointer to double

```
double *a(void);
```

declare a as function (void) returning pointer to double

```
double (*a)(void)
```

declare a as pointer to function (void) returning double

3.2 Was ist die Bedeutung der Namen

Konsultieren Sie die Theorie in Abschnitt 4.

```
a b; // a: alias to some type as the type of the variable  
  
// b: variable name
```

```
a b(c d); // a: alias to some type as the return type of the function  
  
// b: name of the function  
  
// c: alias to some type (of the first parameter of the function)  
  
// d: the name of the first parameter of the function
```

```
a b[c][d]; // a: alias to some type as the element type of the array  
  
// b: name of the array  
  
// c: some constant defining the size of the first array dimension  
  
// d: some constant defining the size of the second array dimension
```

3.3 Schreiben Sie folgende Deklarationen

Konsultieren Sie die Theorie in Abschnitt 4.

```
// declare f as pointer to function (void) returning pointer to char  
char>(*f)(void);
```

```
// declare tab as array 10 of pointer to function (int) returning int  
int(*tab[10])(int);
```

```
// declare wrap as function (pointer to function (int) returning void)  
// returning void  
void wrap(void(*) (int));
```

```
// declare matrix as array 5 of array 5 of double  
double matrix[5][5];
```

```
// declare argv as array of pointer to char  
char*argv[];
```

```
// declare p as pointer to array 5 of array 7 of char
```

```
char(*p)[5][7];
```

```
// declare a as array of array 5 of array 7 of char
```

```
char a[][5][7];
```

4 Theorie

Die Syntax von nicht-trivialen C Deklarationen kann herausfordernd sein und muss daher geübt werden.

4.1 Wozu Deklarationen

Eine Deklaration gibt an, wie ein Name verwendet werden kann, z.B.

```
int a; // a ist eine einfache Variable vom Typ int
int b[4]; // b ist ein Array von 4 Elementen vom Type int
int c(void); // c ist eine parameterlose Funktion welche einen int
// Wert zurückgibt
enum d { e }; // enum d ist ein Aufzählungstyp
enum d f; // f ist eine einfache Variable vom Typ enum d
enum g { h } i; // i ist eine einfache Variable vom Typ enum g
enum { j } k; // k ist eine einfache Variable von anonymem enum Typ
struct l { int m; }; // struct l ist ein Strukturtyp
struct l n; // n ist eine einfache Variable vom Typ struct l
struct o { int p; } q; // q ist eine einfache Variable vom Typ struct o
struct { int r; } s; // s ist eine einfache Variable von anonymem struct Typ
```

Zu beachten:

- die Aufzählungstypen im Beispiel heißen `enum d` und `enum g` (und nicht `d` bzw. `g`)
- die Strukturtypen im Beispiel heißen `struct l` und `struct o` (und nicht `l` bzw. `o`)
- mit einer `struct` und `enum` Definition kann gleichzeitig z.B. eine Variable definiert werden, wie oben die Variablen `i`, `k`, `q` und `s`
- `struct` und `enum` Typen können anonym bleiben – dadurch können sie aber nicht mehr durch einen Namen angesprochen werden, siehe die obigen Variablen `k` und `s`

4.2 Alias Typ Deklarationen

Für gegebene Typen kann man alternative Namen deklarieren, sogenannte Aliase.

Das Schema dabei ist einfach:

- 1) schreiben Sie eine Variablen Deklaration mit dem gegebenen Typen
- 2) schreiben Sie davor das Schlüsselwort `typedef` – dadurch wird der Name, welcher im ersten Schritt für die Variable stand, zum Alias des gegebenen Typs

Der Alias Typ Name kann überall verwendet werden anstelle des originalen Typ Namens.

```
typedef int a; // a ist ein Alias für int
typedef int b[4]; // b ist ein Alias für ein Array von 4 int
typedef enum d f; // f ist ein Alias für enum d
typedef enum g { h } i; // i ist ein Alias für enum g
typedef enum { j } k; // k ist ein Alias für den anonymen enum Typ
typedef struct l n; // n ist ein Alias für struct l
typedef struct o { int p; } q; // q ist ein Alias für struct o
typedef struct { int r; } s; // s ist ein Alias für den anonymen struct Typ
```

Zu beachten:

- durch die Alias Deklaration können anonyme Typen trotzdem benannt werden
- einem Alias sieht man nicht an wofür er steht, z.B. mit obigem Alias für `b` kann man folgendes schreiben: `b x; printf("sizeof(x)=%zd\n", sizeof(x));` das Resultat ist 16, falls `sizeof(int)=4` ist – somit ist `x` ein Array von vier `int` Elementen, auch wenn das auf den ersten Blick nicht ersichtlich ist

4.3 Grundform von Funktions- und Variablen Deklarationen

Die Grundform einer Funktions- oder Variablen Deklaration ist folgendermassen:

Typ Deklarator ;

Typ steht für einen **Typ Namen** wie `int`, `char`, `struct s`, `enum e`, ... oder einen **Alias Namen**. Optional kann vor oder nach dem Typ- oder Alias Namen das Schlüsselwort `const`¹ stehen.

Deklarator gibt an, ob es eine einfache Variable, ein Array, eine Funktion oder ein Pointer ist – oder (rekursiv) eine Kombination davon. Mehr dazu gleich.

Beispiele:

```
int a;           // "int" ist der Typ, "a" ist der Deklarator
int (a);        // der Deklarator kann geklammert werden, der Typ nicht
int a[3];       // "int" ist der (Element) Typ, "a[4]" ist der Deklarator
int a(int);     // "int" ist der (Return) Typ, "a(int)" ist der Deklarator
int *a;         // "int" ist der (Objekt) Typ, "*a" ist der Deklarator
```

4.4 Syntax Regeln von Deklaratoren

Deklaratoren haben dasselbe Schema wie Operatoren bei Ausdrücken: es gibt eine Priorität und eine Assoziativität.

Priorität	Assoziativität	Elemente	Beschreibung	Beispiel
1	-	(...)	Gruppierung	<code>int (*a[5])(void);</code>
2	links-rechts ((*[...] [...])	<code>x[...]</code> <code>x(...)</code>	Array Funktion	<code>int a[5];</code> <code>int f(void);</code>
3	rechts-links (* (const x))	<code>const</code> <code>volatile</code> <code>*</code>	konstant flüchtig Pointer	<code>int * const a;</code> <code>int * volatile a;</code>

Beispiele

```
int a;           // declare a as int
int (a);        // declare a as int

int a[];        // declare a as array of int
int a[2];       // declare a as array 2 of int

int *a[];       // declare a as array of pointer to int
int (*a)[];     // declare a as pointer to array of int

int a[2][3];    // declare a as array 2 of array 3 of int

const int (* const a)[3]; // declare a as const pointer to array 3 of const int

int f(void);    // declare f as function (void) returning int

int *f(int);    // declare f as function (int) returning pointer to int
int (*f)(int); // declare f as pointer to function (int) returning int
```

Sie können diese Deklarationen auf folgender Web Page überprüfen: <http://www.cdecl.org>.

¹ oder `volatile` oder eine Kombination von beiden

4.5 Spezialfälle

const/volatile im Zusammenhang mit Pointer Alias

Problemstellung:

```
typedef char * ptr_t; // ptr_t is an alias for char *
ptr_t const p1;      // is this now const char * or char * const?
const ptr_t p2;      // is this now const char * or char * const?
p1 = NULL;          // allowed or not? depends on the answers above
p2 = NULL;          // allowed or not? depends on the answers above
```

Lösung:

Die C Sprach-Designer haben entschieden, dass, egal wo bei der Deklaration **const** (oder **volatile**) steht, **const** interpretiert wird, wie wenn es unmittelbar links vom Variablen Namen stehen würde. Somit sind beide, **p1** und **p2** folgendermassen zu interpretieren:
char * const p;

Somit sind hier weder **p1 = NULL;** noch **p2 = NULL;** erlaubt.