types and strings

Philipp Klaus Krause

May 7, 2019

Table of Contents

Table of Contents

Präprozessor

Der Präprozessor wird ausgeführt, noch bevor etwas anderes mit dem Code passiert. Die wichtigsten Präprozessoranweisungen:

- #include
- #define
- #ifdef
- #ifndef
- #if
- #else
- #error

#define

Definiert ein Makro. Wir schreiben erstmal nur Makros ohne Argumente!

```
#define ARRAY_MAX_LEN 17
…
int array[ARRAY_MAX_LEN];
…
for(size_t i = 0; i < ARRAY_MAX_LEN; i++)
…
```

#error

```
#if __STDC_VERSION__ < 201112
#error Dieses Programm benötigt einen C11-Compiler
#endif
```

Table of Contents

Basic types

- `char` (Byte)
- signed und unsigned integer types (Ganzzahlen)
- floating types (Gleitkommazahlen)

Ihr könnt ab nun alle (auch wenn sie nicht explizit auf den folgenen Folien vorkommen) verwenden (mit Ausnahme der Komplexen), solltet diese aber sinnvoll wählen.

Character types

- `char`: 1 Byte (mindestens 8 Bit). Wichtig auch für Strings (die werden heute auch noch vorgestellt)
- `unsigned char`: 1 Byte, als Vorzeichenlose Zahl behandelt

integer types

- `bool` aus `stdbool.h`: `false` (0) oder `true` (1).
- `unsigned char`: Bereich mindestens 0 bis $2^8 - 1$.
- `int`: Bereich mindestens $-2^{15} + 1$ bis $2^{15} - 1$.
- `unsigned [int]`: Bereich mindestens 0 bis $2^{16} - 1$.
- `long [int]`: Bereich mindestens $-2^{31} + 1$ bis $2^{31} - 1$.
- `unsigned long [int]`: Bereich mindestens 0 bis $2^{32} - 1$.

Makros in `limits.h`, z.B. `UINT_MAX` für `unsigned int`. Bei Überlauf: Für unsigned Wraparound, also z.B. für `unsigned int` Arithmetik modulo `UINT_MAX` +1.

size_t

- Findet sich z.B. in `stddef.h`
- Groß genug für Größen und Indices selbst der größten Arrays
- Üblicherweise eine gute Wahl für Größen und Indices von Arrays unbekannter Größe

Table of Contents

Arrays (auch „Felder“genannt)

Mittels `[]` wird die Größe des Arrays bei Initialisierung automatisch gewählt.

```
float matrix[3][4]; /* Array von Arrays:
      matrix ist ein Array mit 3 Elementen,
      matrix[0] ist ein Array mit 4 Elementen. */
int im[2][3] = {{1, 0, -1}, {-2, -1, 0}};
int a[] = {0, 1, 2}; // Hat 3 Elemente
int m[][2] = {{1, 0}, {0, 2}};
int m[2][] = {{1, 0}, {0, 2}}; // Fehler!
```

Später werden wir Zeiger kennenlernen. Arrays sind etwas anderes als Zeiger!

Arrays als Argumente

Arrays werden anders als andere Argumente anderer Typen gehandhabt. Daher wird in der aufgerufenen Funktion keine Kopie des Arrays, sondern das Original verändert.

```
void f(int a[])
{
    a[0] = 17;
}

void g(void)
{
    int a[1] = {0};
    f(a);
            printf("%d\n"); // 17
}
```

Table of Contents

integer promotion

Typen kleiner als int werden bei der Verwendung zu `int` oder `unsigned int`.

```
unsigned char c = 200;
unsigned char d = 250;
int i = c + d; // 450
int j = c - d; // -50
bool l = false;
bool r = true;
int k = l - r; // -1
```

argument promotion

Für Argumente von `printf` und `printf`.

- integer promotions
- `float` nach `double`

Somit können `bool`, `char`, `unsigned char` mit `printf` wie `int` ausgegeben werden, `float` wie `double`.

Table of Contents

stdio.h

`printf` conversion specifications: % gefolgt von optionalem

- long: l
- long long: ll
- size_t: z

und nichtoptionalem

- int: d (dezimal)
- unsigned int: u (dezimal), x (hexadezimal)

```
printf("%zu%%\n", (size_t)100);
```

`printf` conversion specifications: % gefolgt von optionalem

- long (nur für long double): L

und nichtoptionalem

- double: f (dezimal), e (exponential), g (je nach Wert)
- string (array von char): s
- %: %

Table of Contents

Strings (auch „Zeichenketten“genannt)

Strings sind Nullterminierte Arrays von `char`:

```
char test1[] = "Hallo!"; // Tut das selbe wie unten:
char test2[] = {'H', 'a', 'a', 'l', 'l', 'o', '!', 0};
assert (test1[6] == 0);
```

Ein bischen Zeiger

Manchmal werden Felder als Zeiger behandelt (und die Zeiger als Felder). Die Details folgen irgendwann später. Jetzt verwenden wir dies erstmal nur als Syntax für Arrays von Strings unterschiedlicher Länge:

```
const char *a[] = {"Hallo", "Ade"};
```

main

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    for(int i = 0; i < argc; i++)
        printf("Argument %d: %s\n", i, argv[i]);
    return 0;
}
```

Table of Contents

stdlib.h: strto*

Zeichenkette in Zahl wandeln, bei Ganzzahlen mit Angabe der Basis. Die Bedeutung des zweiten Arguments wird später vorgestellt, wir verwenden dort erstmal immer 0.

- `strtol`: `long`
- `strtoul`: `unsigned long`
- `strtof`: `float`
- `strtod`: `double`

Beispiel für `long int strtol(const char nptr[], char **endptr, int base);`:

```
char zahl[] = "120";
assert(strtol(zahl, 0, 8) == 80);
```

stdlib.h: rand

Gibt eine Pseudozufallszahl im Bereich 0 bis `RAND_MAX` (welches mindestens 32767 ist) zurück.

```
for(int i = 0; i < 20; i++)
{
    char Zifernstring[] = "X";
    Zifernstring[0] = '0' + rand() % 10;
    printf("Pseudozufällige Ziffer: %s");
}
```

math.h / tgmath.h: sin, cos, tan, sqrt, pow

Funktionen für `double` in `math.h`:

- `sin`: Sinus
- `cos`: Kosinus
- `tan`: Tangens
- `sqrt`: Quadratwurzel
- `pow`: Exponentiation
- `ceil`: Obere Gaußklammer
- `floor`: Untere Gaußklammer
- `fabs`: Betrag

Man kann auch `tgmath.h` verwenden, dann klappt es für `float`, `double`, `logn double`.

string.h: strcmp

Vergleicht zwei Zeichenketten

```
assert(strcmp("test", "test") == 0);
assert(strcmp("test", "test1") < 0);
assert(strcmp("test3", "test2") > 0);
```

string.h: strlen

Länge der Zeichenkette in Byte (ohne terminierende 0).

```
assert(strlen("test") == 4);
assert(strlen("") == 0);
```

string.h: strerror

Was passiert, wenn z.B. bei `strtol` der Eingabstring sich nicht in eine Zahl wandeln lässt? Manche Funktionen setzten im Fehlerfalle die Variable `errno` (in `errno.h`) vom Typ `int` auf einen Wert ungleich 0.

```
char zahl[] = "999999999999999999999999";
strtol(zahl, 0, 10);
if(errno)
    printf("Fehler: %s\n", strerror (errno));
```

stdio.h: snprintf

```
int snprintf(char s[], size_t n, const char format[];
```

Ausgabe in string statt auf die Standardausgabe. Eine Gelegenheit für Buffer-Overflow-Exploits bei hinreichend großer Wahl des Parameters n.

```
#define BUFSIZE 32
char buffer[BUFSIZE];
snprintf(buffer, BUFSIZE, "%s%s%s", "Hallo", ", ", "Welt!");
printf("%s\n", buffer);
```

stdio.h: puts

Gibt einen String gefolgt von einem Zeileumbruch aus. Die beiden folgenden Aufrufe bewirken also das selbe:

```
puts(string);
printf("%s\n", string);
```

stdlib.h: mblen

Länge des nächsten Zeichens in Byte (oder 0, falls das nächste Zeichen 0 ist, oder -1 falls es kein gültiges Zeichen ist).

```
mblen("test"); // 1
mblen(""); // 0
```