more types and strings

Philipp Klaus Krause

May 21, 2019

Table of Contents

Table of Contents

Automatische Variablen

Automatische Variablen vereinfachen Regeln bei GNU make.

- $@ Ziel der Regel
- $< erste Abhängigkeit
- $^ alle Abhängigkeiten

```
keilnumbertest: keildigit.o keilnumber.o keilnumbertest.o
    $(CC) $(CFLAGS) $(LIBRARIES) $^ -o $@
```

Patternregeln

Statt für jede zu erstellende Datei eine eigene Regel anzulegen, können mit % Muster verwendet werden:

```
%.o: %.c $(HEADERS)
    $(CC) $(CFLAGS) -c $< -o $@
```

Table of Contents

struct mit Tag

```
struct point_t
{
    int x;
    int y;
}
…
struct point_t point = {1, 1};
struct point_t *p = &point;
…
point.x++;
p->y++; // Wie (*p).y++
```

Hinweis: Jede struct lässt sich mit `{0}` intialisieren (dann werden alle Member auf 0 initalisiert, auch rekursiv.).

struct ohne Tag

```
struct
{
    int x;
    int y;
} point = {1, 1};
…
point.x++;
p->y++; // Wie (*p).y++
```

typedef

Führt einen neuen Namen für einen Typen ein.

```
typedef int Ganzzahl;
typedef float vektor_t[4];
typedef struct {unsigned int x, y;} punkt_t;
…
Ganzzahl i;
vektor_t u, v;
punkt_t p, q;
```

Unvollständige Typen

```
struct x; // Unvollständige Struktur
typedef struct x x_t;
struct x *p1; // Zeiger auf unvollständige Strukur
x_t *p2; // Zeiger auf unvollständige Strukur
struct x s1; // Fehler
x_t s2; // Fehler
```

Unvollständige Typen

Um Variablen des Typs deklarieren zu können, muss dieser erst vervollständigt werden.

```
struct x; // Unvollständige Struktur
struct x s1; // Fehler
struct x {int a; int b;};
struct x s2; // OK
```

Table of Contents

Zeichensätze

Basic source character set, basic execution character set. Die folgenden 95 Zeichen gibt es darin immer; diese passen auch immer in ein Byte.

- A … Z
- a … z
- 0 … 9
- ! ” # % & ’ ( ) * + , - . / : ; < = > ? [ ] _ ^ { | }
-

Dazu (implementierungsabhängig) extended character set. Jedes Zeichen passt in ein `wchar_t` (aus `wchar.h`)

Locale

Wie Zeichen kodiert sind, und welche verfügbar sind (im extended character set) hängt vom Locale ab. Außerdem beinflusst das locale auch Sprache, Zahlendarstellung, Zeit- und Datumsformate, etc. Unter GNU/Linux und Hurd lässt sich überlicherweise mittels

```
#include <locale.h>
…
setlocale(LC_ALL, "C.UTF-8");
```

ein Locale erhalten, dass ganz Unicode umfasst, und in dem Strings UTF-8-kodiert sind, sich aber ansonsten wie das Standardlocale verhält. Die Zeichen in `wchar_t` sind üblicherweise immer Unicode-Codepoints (wenn das so ist, so ist das Makro `__STDC_ISO_10646__` definiert).

Zeichen

- `'c'`: Zeichen als `char` (aber gecastet nach `int`)
- Escape-Sequenz erforderlich: `'\''`, `'\\'`, möglich: `'\"'`
- `L'c'` : Zeichen als `wchar_t`

Zeichenketten

- `"..."` : Array von `char`, Kodierung implementierungsabhängig (üblicherweise einfach in der Kodierung der Quellcodedatei).
- Escape-Sequenz erforderlich: `'\"'`, `'\\'`, möglich: `'\''`
- `u8"..."` : Wie oben, aber immer UTF-8-kodiert
- `L"..."` : Wie oben, aber array von `wchar_t`

Die Anzahl an Bytes für ein Zeichen ist höchstens `MB_LEN_MAX` (Konstante aus `limits.h`).

strncmp aus string.h

`int strncmp(const char *s1, const char *s2, size_t n)`
Vergleicht Zeichenketten (wie `strcmp` einfach Byteweise, ohne das Locale zu berücksichtigen), aber nur die ersten `n` Bytes.

```
assert(!strncmp("Übung", "Ü", strlen("Ü"));
```

wcslen aus wchar.h

```
size_t wcslen(const wchar_t *s)
```

Länge des Strings.

```
assert(strlen("Universitaet") == 12);
assert(strlen("Universität") >= 11);
assert(wcslen(L"Universität") == 11);
```

wctomb, mbtowc aus stdlib.h

```
int mbtowc(wchar_t *pwc, const char *s, size_t n)
int wctomb(char *s, wchar_t wc)
```

Wandelt einzelne Zeichen zwischen `char` und `wchar_t`

```
const char *s = "Ä";
wchar_t w;
mbtowc (&w, s, strlen(s));
assert(w == L'Ä');
```

wcstombs, mbstowcs aus stdlib.h

```
int mbstowcs(wchar_t *pwcs, const char *s, size_t n)
int wcstombs(char *s, wchar_t pwcs, size_t n)
```

Wandelt Strings zwischen `char` und `wchar_t`. `n` ist die Obergrenze der Azahl der geschriebenen `wchar_t` bzw. `char`.

```
const wchar_t *w = L"Universität";
char *s = malloc (wcslen(w) * MB_LEN_MAX + 1);
wcstombs (s, w, wcslen(w) * MB_LEN_MAX + 1);
assert(!strcmp(s,"Universität"));
free (s);
```

Table of Contents

Konstanten

Nicht zu verwechseln mit `const`

- Ganzzahlkonstanten
- Gleitkommakonstanten
- Zeichenkonstanten (`'c'`, `L'c'`)

Damit können konstante Ausdrücke gebildet werden, die als Arraygrößen verwendbar sind (so sie einen Ganzzahltyp haben, wofür unter Umständen ein Cast nötig ist).

Ganzzahlkonstanten

Wert:

- Dezimal
- Hexdezimal: Präfix `0`
- Oktal: Präfix `0`

Typ:

- Ohne Suffix: `int`
- `unsigned` : Suffix `u`
- `long`: Suffix `l`
- `long long`: Suffix `ll`

Falls nötig, wird automatisch ein größerer Typ verwendet.

```
0; // Oktal, Typ int
0xa5ul; // Hexadezimal, Typ unsigned long
68000; // Dezimal, Typ int oder long
```