typedef

- In C (leider) etwas zahnlos, kaum mehr als eine textuelle Ersetzung

- Gründe zur Nutzung:
  - Oft und gerne in Herstellerbibliotheken benutzt
  - Erleichtert die Wiederverwendung auch in eigenen Programmen
  - Bei der Deklaration sog. Funktionszeiger fast unverzichtbar

- Syntax
  - typedef bekannte typen neuer_typ;
  - neuer_typ endet gerne auf _t oder _T (zur schnellen Erkennung)

Datentypen bool, enum

- bool (eigentlich _Bool)
  - Garantierte Werte false und true (als 0 und 1)
  - Minimaler Speicherverbrauch (meist uint8_t)
  - in stdbool.h deklariert

- enum
  - „Verkleidung“ für einen Integer
  - automatisch: Disjunkte (nicht überlappende) Zuweisung Symbol zu Wert
  - manuell: Übersteuerung (dann auch gleiche Werte) möglich
  - In C keine Trennung der Namensbereiche (in C++ schon)

Integerbaukasten

- **Neue Formatangaben für die printf/scanf-Familie**
  - Bisheriger Formatangabe (z.B. d) wird ersetzt durch ein Makro
    - `#include <inttypes.h>`
  - Regeln zum Ersetzen einer bisherigen Formatangabe, z.B. d
    - Stelle bei printf ein PRI voran -> PRId
    - Stelle bei scanf ein SCN voran -> SCNd
    - Hänge die Längenangabe (Bitanzahl) hinten an -> d16
  - Das Makro muss im Quelltext außerhalb der Anführungszeichen stehen

- Beispiele für `int16_t x; uint8_t y;`
  - `printf("x=%d, y=%u.",x,y);` -> `printf("x=%"PRId16",y=%"PRIu8".",x,y);`
  - `scanf ("%d", &x);` -> `scanf ("%"SCNd16"", &x);`

Makros

- **Eigenschaften**
  - werden vom C-Preprocessor, sind eine reine Textersetzung (kein C-Konstrukt)
  - werden mit #define angelegt und mit #undef gelöscht

- **Arten** (das ist hier eine künstliche Unterscheidung)
  - Existenzmakros: haben keinen Ersetzungstext
    - #define NUR_ZUM_TEST
    - zur bedingten Kompilierung (#ifdef, …)
  - ohne Parameter: werden immer mit demselben Text ersetzt
    - #define ANZAHL_MESSWERTE 1000
    - meist für zentrale Definition einer Konstante (Konsistenz im Quelltext)
  - mit Parameter: Parameter in () werden je nach Aufruf ersetzt
    - #define MYEXIT(code, a) { printf(a); exit(code); }
    - meist Funktionsersatz

Bedingte Kompilierung

- Ausschluss oder Einschluss von Quelltext
  - wird vom C-Preprocessor verarbeitet
  - Folge: Bedingungen werden beim Kompilieren verarbeitet

- Bedingungsabfragen mit
  - #ifdef/#ifndef oder #if
  - #elseif oder #elif
  - #else
  - #end
- Bei Verwendung von #if:
  - Existenzabfragen mit defined() bzw. not defined()
  - Arithmetik (+, >) und Logik (&&, ||) möglich