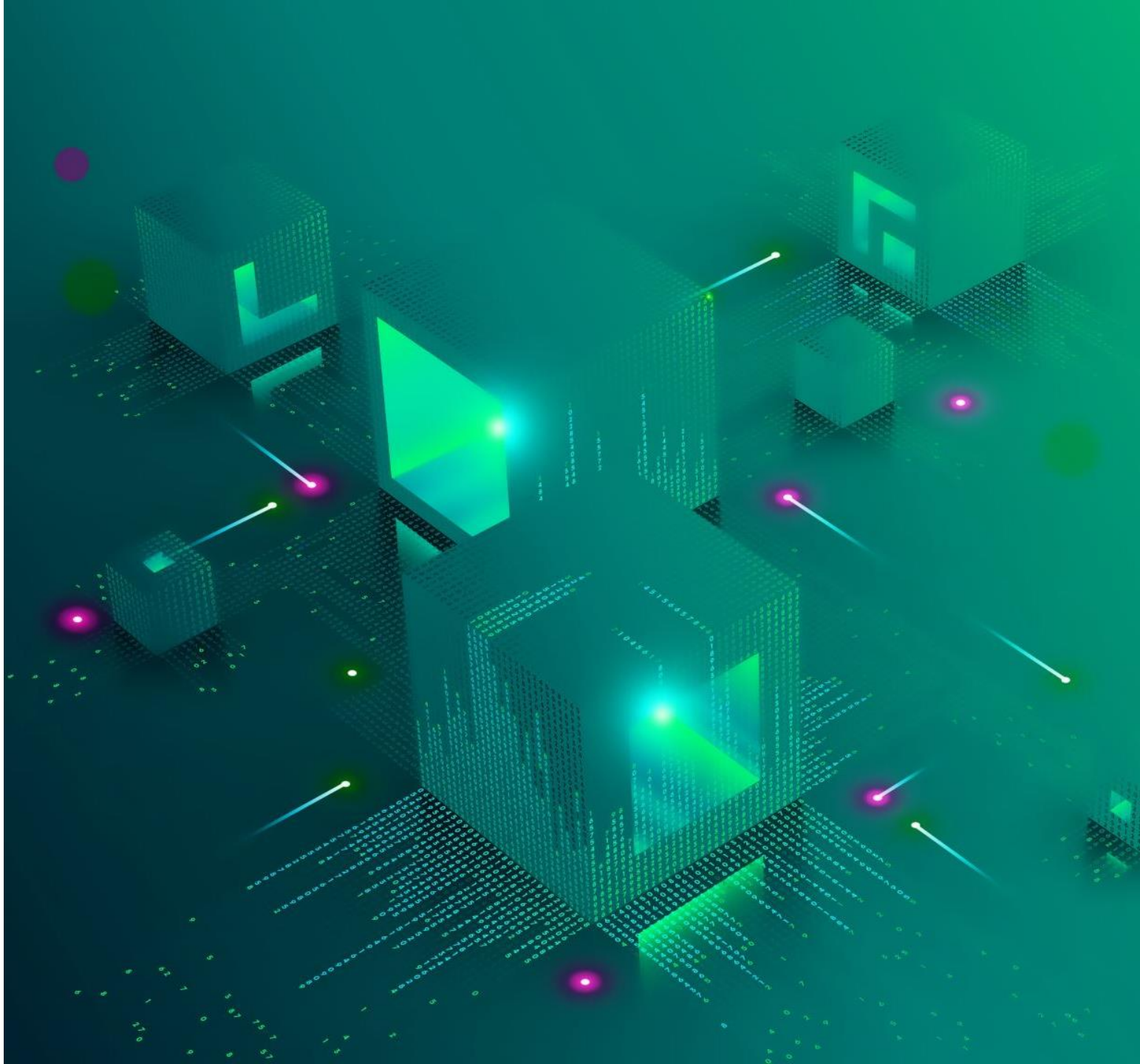
SWB3

Termin 2

SE BB, 23. September 2023

Dmitriy Purgin, MSc



Agenda

- Hausübung
- Typumwandlung
- Funktionen und Parameterübergabe
- Konstanten und Felder
- Speicherklassen
- Zeichenketten

Hausübung

GitHub Classroom

- Hausübungen werden auf github abgegeben
  - Mehr Transparenz
  - Fertige Infrastruktur
  - Tutoren/Lektoren können jederzeit kommentieren/evaluieren
- Doku mit Markdown
- ASSESMENT.md unbedingt ausfüllen!

Hausübung 1

- Eine Woche Zeit für Abgabe
  - Für die späteren zwei Wochen
  - Kein Nachfirst, da Peer Reviewers automatisch zugewiesen werden
- Dann eine Woche für Peer Review
  - Das Ziel ist von einander zu lernen
- Tutoren werden stichprobenmäßig die Abgaben bewerten

Programmierstil

- Einzuhalten damit Tutoren/Kollegen leichteres Leben haben
- Variablen & Funktionen in snake_case, z.B. month , nr_of_days , calculate_age()
- im Quelltext alles auf Englisch
  - Variablen als Nomen
  - Funktionen als Verben
- Makros & Konstanten in Blockschrift (Macro Case), z.B. MAX_SIZE
- selbsterklärende Namen (ev. etwas länger)
- kurze Funktionen (Faustregel: soll auf einen Bildschirm passen)
- Leerzeichen nach if und Komma
- keine Tabulatoren, nur Leerzeichen

Typumwandlungen

Wiederholung

- Was ist eine Variable?
- Welche Datentypen in C kennen Sie?

Implizite Typumwandlung

- Wenn der erwartete Datentyp mit dem tatsächlichen nicht übereinstimmt:
  - Initialisierungsanweisungen
  - Arithmetische Ausdrücke
  - Funktionsaufrufe
  - Return-Anweisungen
- Widening conversion: der kleinere Typ wird auf den größeren erweitert (char -> int, int -> long int, float -> double, int -> double)
  - 42 * 3.0; /* 42 wird zu double */
  - Achtung: signed int wird unter Umständen zu unsigned!
- Narrowing conversion: der größere Typ wird abgeschnitten oder umgerechnet
  - char ch = 42; /* OK, int literal wird zu char */
  - int i = 42.42; /* OK, Nachkommastellen werden abgeschnitten */
  - char ch = 4242; /* not OK, unspecified behavior */

- Siehe https://en.cppreference.com/w/c/language/conversion

Implizite Fehler bei impliziter Typumwandlung

- Signed wird zu unsigned:
  - -1 < 10U ergibt false
- Richtige Typumwandlung an der falschen Stelle:
  - double r = 1 / 3; /* Ergbenis ist 0. */
- Überlauf:
  - char ch = 123456; /* ??? */
- Sign Extension:
  - char ch = -1; int i = ch; /* siehe Hexadezimale Darstellung */

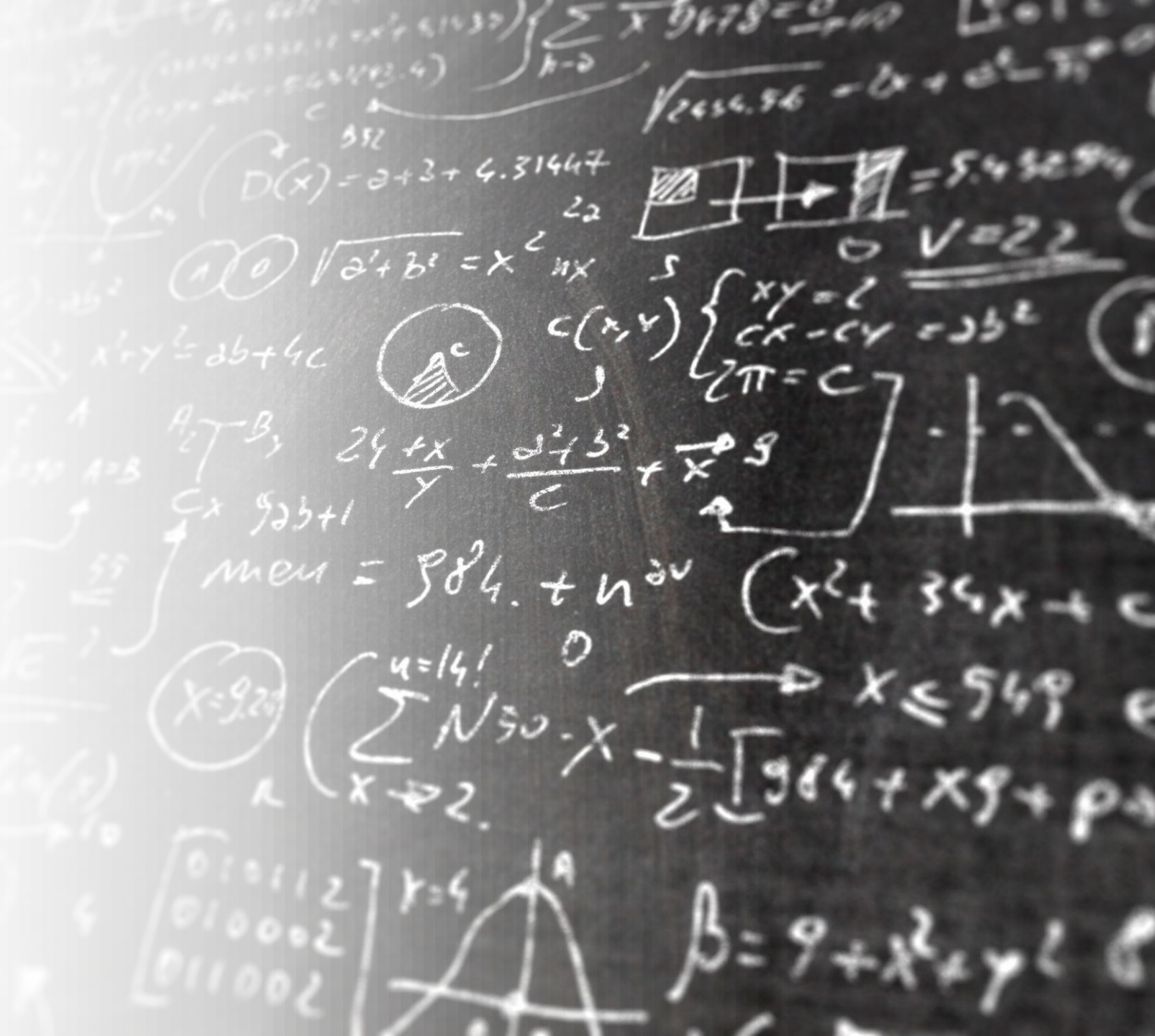
Explizite Typumwandlung

- **(** type-name **)** expression
  - Z.B.: double val = (double)i / 3.0;
- Explicit is better than implicit
  - Klare Intention -> bessere Wartbarkeit

Beispiel

- Calc auf double umbauen
- Siehe v01/main.c

Funktionen und Parameterübergabe



Funktionen

- Deklaration oder Prototyp:
  - Rückgabewert, Name, Parameterliste, gefolgt von ;
- Definition:
  - Rückgabewert, Name, Parameterliste, gefolgt von Funktionskörper in {}

- Deklaration kann mehrfach vorkommen
- Definition darf es nur ein einziges Mal über alle Übersetzungsmodule geben
- Rückgabewert und Parameterliste sind „optional“, aber lieber nicht!

Parameterübergabe

- In C gibt es nur “pass-by-value”
  - Parameter sind lokale Variablen für die Funktion, sie werden für die Funktion neu angelegt, und die übergebenen Werte werden reinkopiert
- “Pass-by-reference”, oder Übergangsparameter, kann mithilfe Zeiger gebaut werden
  - Zeiger ist die Adresse von einem Speicherblock. Mit Dereferenzierung greift man auf diesen Speicherblock zu
- Wichtige Operatoren:
  - `address of`: `&variable`
  - `value of`: `*pointer`
  - Siehe https://en.cppreference.com/w/c/language/operator_member_access

Beispiel

- Calc() -> parameter &ok
- Siehe v02/main.c

Konstanten und Felder

Konstanten

- Schlüsselwort `const`
  - Eine „normale“ Variable, deren Wert nicht mehr geändert werden kann(ohne Tricks)
- Präprozessor #define
  - Man definiert ein Macro mit einem Wert
  - Vor Kompilierung wird der Wert textuell eingesetzt

Felder

- T name[size]: statische Größe, zur Compilezeit definiert
  - Kontinuierlicher Speicherblock am Stack
  - Größe: sizeof(T) * size
  - Initialisierung mit {}: int array[5] = {9, 1, 2, 3, 4};
- T name[size1][size2]...[sizeN]
  - Mehrdimensionale Felder auch möglich
  - Initialisierung mit verschachteteln {}: int array[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
- Dynamische Größe zur Laufzeit über Zeiger möglich
  - Bei einer späteren Übung

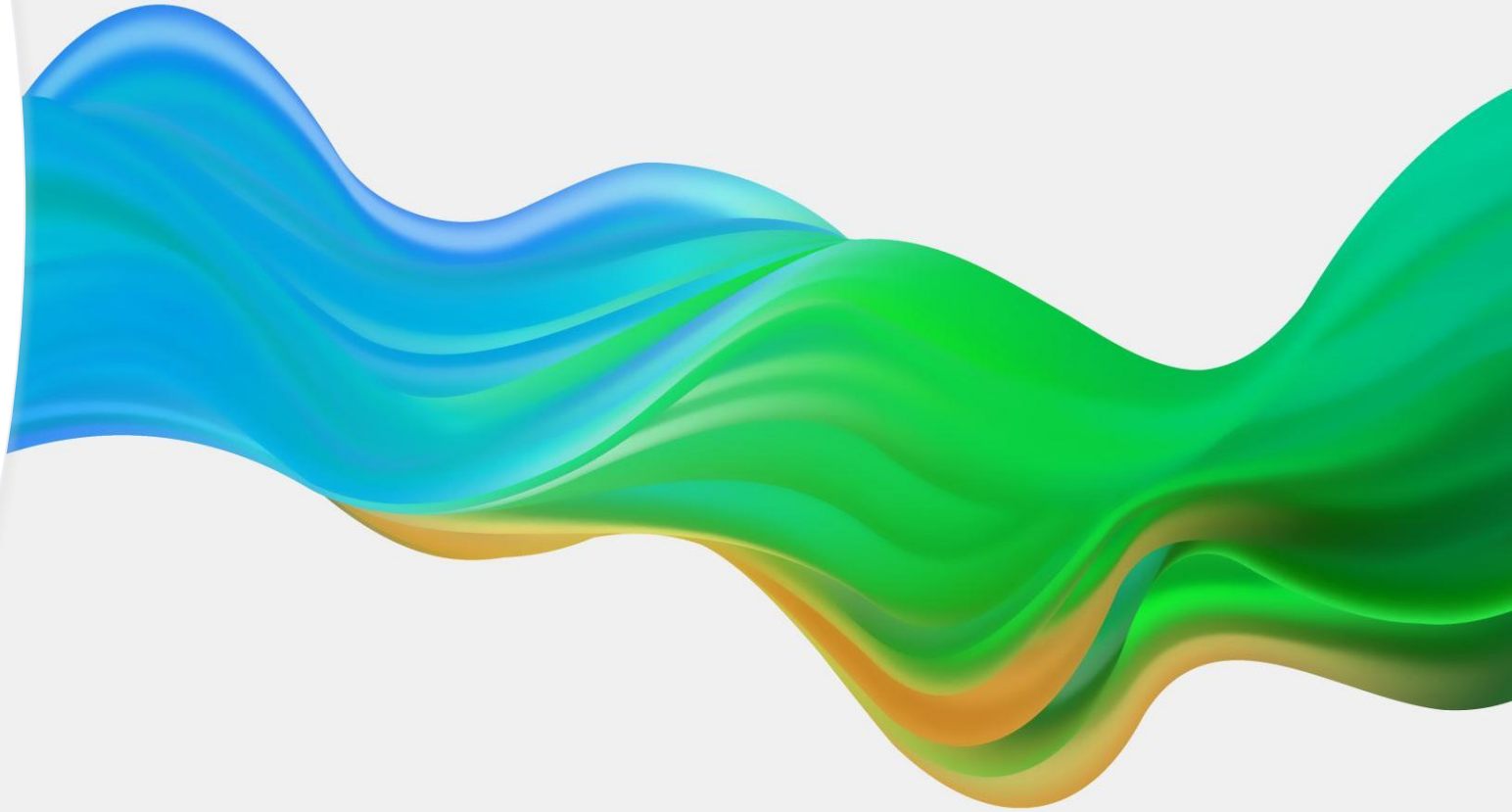
Felder als Zeiger

- T array[] ist äquivalent zu T* für Parameterübergabe, d.h. Arrays werden „by-reference“ übergeben.
  - Die statische Größe geht dabei verloren
  - Bei Verwendung von Arrays als Parameter sollte man die Größe immer ein zusätzliches Argument übergeben
  - Bsp: int main(int argc, char* argv[])
- Der Ausdruck `array[n]` ist das selbe wie `*(array + n)`, und umgekehrt

Beispiel

- Siehe array.c
- Überlauf: Siehe overflow.c

Speicherklassen



Speicherklassen: Lebensdauer

- Storage duration (Lebensdauer von Variablen)
  - Automatic (default) – existiert nur bis Ende des Blocks
  - Static – existiert immer während Programmausführung
  - Allocated – wird von Entwickler*In bestimmt
    - Dynamische Speicherallokation, wird in späteren Übungen behandelt

Speicherklassen: Bindung

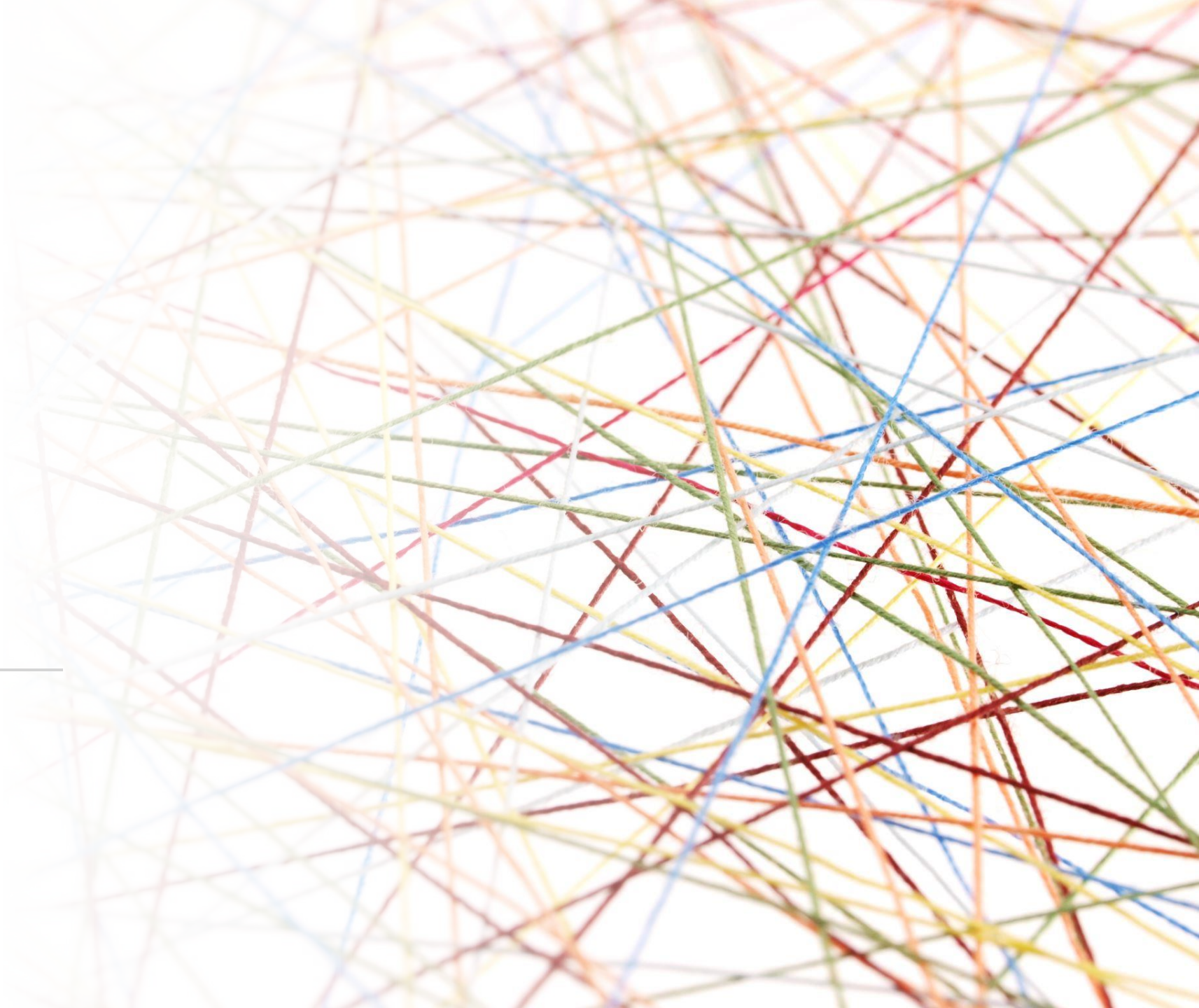
- Linkage (Bindung für Variablen und Funktionen)
  - No linkage – ist nur innerhalb eigenes Blocks gültig
  - External – ist in anderen .c Files unter dem selben Namen gültig
    - Variablen außerhalb von Funktionen mit `extern` und ohne `static`
    - Alle Funktionenen ohne `static`
    - Ähnlich wie „public“ für das Übersetzungmodul
  - Internal – ist nur innerhalb eines .c Files sichtbar
    - Ähnlich wie „private“ für das Übersetzungsmodul
    - Mit `static` gekennzeichnet
    - Statische Variablen außerhalb von Funktionen im .c File, alle Funktionen mit `static`

- Siehe https://en.cppreference.com/w/c/language/storage_duration

Beispiel

- Local static: Siehe v03/calc.c
- Global extern: Siehe v04/calc.c
- Global static: Siehe v05/calc.c

Zeichenketten



Zeichenketten als Felder

- Zeichenketten in C sind eindimensionale Arrays von char, mit \0 am Ende (Null-Terminierung)
  - char hello[] = „Hello“; /* hello wird char[6] */
- Problem: Benutzereingabe
  - Man verwendet Zeichenketten, die „groß genug“ sind, z.B. 100 Zeichen
  - Library-Funktionen mit Größenbeschränkung sollen bevorzugt werden
  - Auf den Überlauf achten!

Bibliothekfunktionen

- Unter <string.h>, siehe https://en.cppreference.com/w/c/string/byte
- Einzelne Zeichen klassifizieren oder manipulieren
  - Ob eine Ziffer, Leerzeichen, Buchstabe, Upper-/Lowercase usw
- Zeichenketten überprüfen oder manipulieren:
  - atoi(), atof() – in eine Zahl konvertieren
  - strlen() – Länge einer Zeichenkette berechnen
  - strcmp(), strncmp() – Zeichenketten vergleichen
  - strcat(), strncat() – Zeichenketten konkatenieren
  - strcpy(), strncpy() – Zeichenketten kopieren

Bibliothekfunktionen

- Man beachte: Es gibt Funktionen mit `str` und mit `strn`
  - Die Variante mit `strn` bearbeitet nur angegeben Anzahl der Zeichen
  - Die Variante mit `str` bearbeitet bis Null-Zeichen vorkommt – das kann zu Buffer Overflow führen!

Beispiel

- fgets, strncmp: Siehe v05/main.c