Lernziele: C-Programmierung

Sie kennen den Übersetzungsprozess in C

> Sie können die Prozesse Präprozessor, Compiler und Linker mit Ihren eigenen Worten definieren und kennen deren Aufgaben.

Der **Präprozessor** ersetzt und ergänzt Programmcode.

- Ersetzen von Trigraph-Zeichen
- Zusammenführen von Zeilen, die durch \ aufgeteilt wurden
- Kommentare entfernen
- Präprozessor-Direktiven ausführen und Makros expandieren

Der **Compiler** übersetzt C-Dateien in Objekt-Dateien

- Syntax prüfen
- Objektcode (Maschinencode) erstellen

Der **Linker** erstellt ausführbare Datei

- kombiniert Objektcode und Bibliotheken zu einem ausführbaren Programm

Sie kennen die Datentypen in C und deren Verwendung und Zugriffe

Elementare Datentypen (int, char, float, etc.)

Ganzzahlen

Datentyp	Byte
`char`	1
`short`	min. 2
`int`	min. 2
`long`	min. 4
`long long`	min. 8

- Vorzeichenlos mit `unsigned`: z.B. `unsigned long li`
- Ganzzahlen können auch in Hexadezimal oder Oktal formuliert werden:

```
int d = 42; // Dezimal
int o = 052; // Oktal (0oo)
int h = 0x2A; // Hexadezimal (0xhh)
```

Gleitkommazahlen

Datentyp	Byte
`float`	4
`double`	8
`long double`	min. 8

Characters

Strings werden als Zeichenvektoren (`char`-Array) realisiert:

```
char s[] = "Hello\n";
```

Einige Sonderzeichen:

- `\0` : Null Terminator
- `\a` : Alert
- `\b` : Backspace
- `\f` : Formfeed
- `\n` : Newline
- `\r` : Carriage Return
- `\t` : Horizontal Tab
- `\v` : Vertical Tab
- `\\` : Backslash

Modifizierer

Konstanten sind standardmässig vom Typ `int` (Ganzzahlen) oder `double` (Gleitkommazahlen). Dieses Verhalten kann mit *Modifizierern* beeinflusst werden: * `L` für `long` (Ganzzahlen) * `U` für `unsigned` * `F` für `float` * `L` für `long double` (Gleitkommazahlen)

```
float pi = 3.1415F
```

Typumwandlung (Typecasting):

```
int i;
float f = 3.0F;
i = (int)f;
```

Nicht-elementare Datentypen

Strukturen (`struct`)

TODO

Aufzählungen (`enum`)

Folgen von ganzzahligen Werten, erlauben dem Compiler eine Typenüberprüfung.

```c
enum Boolean { FALSCH, WAHR }; // FALSCH hat den Wert 0
enum States {
IDLE = 1,
IN_USE, // Wert 2
UNDEFINED = 0xFF
};
enum States s = IDLE; // Variablendeklaration
```

Union

TODO

Bitfelder

TODO

Zeiger

Zeiger (Pointer) sind Variablen, die Adressen enthalten.

```c
char c;
char *p;
p = &c;   // Zuweisung der Adresse von c (& : Adress-Operator)
*p = 45;  // c = 45 (* : Inhalts-Operator)
```

Zeigerarithmetik

Jeder Zeiger zeigt auf einen festgelegten Datentyp. Zeigt ein Pointer `pa` auf ein Element eines Vektors, dann zeigt `pa+1` per Definition auf das nächste Element.

Vektoren

```c
int array[10];
array[2] = 42;
// oder
int array[] = {1,1337,42,9};
```

Deklaration von Vektoren

“‘c int static[5] = {0,1,2,3,4}; // statisch, 5 Elemente int* dynamic = (int*) *malloc(sizeof(int)* n); // dynamisch, n Elemente //… free(dynamic); // speicher wieder freigeben

Typendefinition

“‘c typedef unsigned long uint32_t; // C99: include <stdint.h> typedef char String[]; typedef int size_t; // Für enum typedef enum myBoolean_ { FALSE, // Wert 0 TRUE } myBoolean_t;

1. **Sie können Variable in C definieren und korrekt ansprechen.**
   - Dies können Sie für alle eingeführten Datentypen.

```c
float f;
int a,b,c;
```

Sie kennen die Operatoren in C

> Vergleichsoperatoren, logische Operatoren, mathematische Operatoren, Zuweisungsoperatoren, Inkrement- und Dekrementoperatoren als Postfix- und Präfixoperator, bitweise Operatoren, Adressoperator, sizeof, Bedingung, Typkonvertierung.

Arithmetische Operatoren * `+` : Addition, Vorzeichen * `-` : Subtraktion, Vorzeichen * `*` : Multiplikation, Umwandlungsoperator * `/` : Division * `%` : Modulo

Zuweisung * `=` : Zuweisung * `+=` : Kombinierte Zuweisung

Es exisiteren folgende kombinierte Zuweisungen `+=` , `-=` , `*=` , `/=` , `%=` , `&=` , `|=` , `^=` , `<<=` , `>>=`

Inkrement / Dekrement * `++a` : Inkrement (Präfix): a = a+1 * `a++` : Inkrement (Postfix) * `--a` : Dekrement (Präfix) * `a--` : Dekrement (Postfix)

Vergleichsoperatoren * `==` : Gleichheit * `!=` : Ungleichheit * `>` : Grösser * `<` : Kleiner * `>=` : Grösser gleich * `<=` : Kleiner gleich

Logische Operatoren * `!` : Logisches NICHT * `||` : Logisches ODER * `&&` : Logisches UND

Bitweise Operatoren * `&` : Bitweises UND * `|` : Bitweises ODER * `^` : Bitweises XOR * `~` : Bitweises NICHT * `<<` : Linksshift * `>>` : Rechtsshift

```
int x = 0x1F; // 0001 1111
x = x << 1;   // 0011 1110
```

Datenzugriff & Speicherberechnung * `*` : Dereferenzierung * `.` : Elementzugriff bei Struktur oder Zeiger * `->` : Elementzugriff über Zeiger (`ptr->element` entspricht `(*ptr).element`) * `&` : Adresse * `sizeof` : Grösse eines Datentyps ermitteln

Ternärer Operator

```
a ? b : c ;
// entspricht:
if( a ) { b } else { c }
```

Beispiel:

```
max = (a > b) ? a : b;
```

Sie kennen die Kontrollstrukturen in C

> Sequenz, Selektion (if und switch) und Iteration (for, while, do while).

if-else

```c
if (a) {
  //...
} else if (b) {
   //...
} else {
   //...
}
```

switch

```c
switch (a) {
  case 1 : doThis();
      break;
  case 2 : doThat();
      break;
  default : somethingElse();
      break;
}
```

for

```c
//for (Initialisierung, Abbruchbedingung, Finalisierung)
for (int i = 0; i < 10; i++) {
  printf("%d\n", i);
}
```

while

“‘c while(a) { //… }

do-while “‘c do { //… } while (a);

goto

```c
label:
// some code
goto label;
```

Mit `break` können Schleifen vorzeitig verlassen werden, mit `continue` wird mit dem nächsten Schleifendurchlauf weitergefahren.

Sie kennen das Konzept der Zeiger und können es anwenden

> Programmargumente, Strings, call-by-reference, Zeiger auf Funktionen

TODO

Sie können wesentliche Funktionen der C-Standardbibliothek nutzen

`- Sie nutzen die Funktionen zur gepufferten Ein- und Ausgabe.`

Ein- und Ausgabe von Zeichen * `getchar()` : liefert das nächste Zeichen der Standardeingabe (oder EOF) * `putchar(int c)` : schreibt ein Zeichen in die Standarausgabe

Formatierte Ausgabe

```
printf("String: %s, Number: %d, Float: %3.2f\n", "Hello", 5, 99.9);
```

Funktionen mit Zeichenvektoren

- `size_t strlen(const char* str)` : Anzahl Zeichen in einer Zeichenkette
- `char* strcpy(char* dest, const char* src)` : Zeichenkette kopieren
- `char* strcat(dest, name)` : Zeichenkette verketten ‘ **Funktionen zur Dateiverwaltung**

TODO

Sie kennen die Elemente der modularen Programmierung in C

> Sie können eine Funktion definieren und deklarieren und kennen die Unterschiede. Sie können Funktionen korrekt aufrufen. Sie können die Begriffe: Rückgabewert, Parameter (aktuell und formal), Variable (lokal und global) definieren.

TODO

Parameter (formal)

TODO

> Sie verstehen die dahinterliegenden Konzepte und wissen in konkreten Situationen, wann Parameter, Rückgabewerte und Variablen einzusetzen sind. Sie können sich dabei angemessen für eine bestimmte Speicherklasse von Variablen (z.B. auto, register, extern) entscheiden.

TODO

> Sie kennen das Konzept von call-by-value und call-by-reference Parametern und wissen, wie diese definiert werden.

TODO

Sie können entscheiden, ob eine bestimmte Teilaufgabe in einer selbstständigen Funktion gelöst werden sollte.

TODO

Sie können ein Programm angemessen auf mehrere Dateien verteilen

Sie wissen, welche Programmteile in einer Quelldatei zusammengefasst werden sollten. Sie wissen, was eine Definitionsdatei ist. Sie wissen, welche Elemente in einer Definitions- und welche in einer Quelldatei stehen.

Sie kennen Präprozessordirektiven

#define, #if, #ifndef, #endif, #include

TODO

Sie kennen das Konzept von variablen Parameterlisten

TODO

Sie können dynamische Datenstrukturen implementieren

Sie können die Funktionen malloc und free korrekt verwenden.

TODO

Sie können zu einem gegebenen, einfachen Problem ein korrektes C-Programm auf Papier schreiben

Das Programm darf keine logischen Fehler enthalten und möglichst wenige syntaktischen Flüchtigkeitsfehler.

```c
#include <stdio.h>

int main(int argc, char **argv) {
   printf("Hello world!\n");
}
```

14. **Sie können ein gegebenes C-Programm lesen und dessen Funktionsweise verstehen.**

Material zum Üben: * https://www.ioccc.org/years.html