

C-Klausur Zusammenfassung

Checkliste der Konzepte

- ☐ Variablen
- ☐ If/Else/Ternary
- ☐ Switch
- ☐ While/Do While
- ☐ For
- ☐ Arrays (eindimensional, mehrdimensional)
- ☐ Bitweise Operationen (NOT, AND, OR, XOR, Shiften bei signed und unsigned)
- ☐ Funktionen
- ☐ Pointer (Deklaration, &-Operator, *-Operator, Call by Reference)
- ☐ Pointer/Array-Konvertierung
- ☐ Strings (char Arrays, strcat, strcpy, strcmp, strlen)
- ☐ Dynamischer Speicher (malloc/calloc, free, memset)
- ☐ Structs (Punktoperator, Pointer, Pfeiloperator)
- ☐ Unions (Speicherdarstellung, Access)
- ☐ Enums
- ☐ Header-Dateien
- ☐ Datei Input/Output (fopen, fclose, fseek, ftell, rewind, fread, fwrite, fgetc, fputc, fgets, fputs)
- ☐ Präprozessor-Direktiven (#define, #ifndef)

1. Variablen

- Variablen speichern Daten und haben einen Datentyp, z. B. `int`, `float`, `char`.
- Syntax:

```
int x = 10;
float y = 3.14;
char z = 'A';
```

2. If/Else/Ternary

- **If/Else:** Bedingte Anweisung, die Code basierend auf einer Bedingung ausführt.

```
if (x > 5) {  
    // Code wenn wahr  
} else {  
    // Code wenn falsch  
}
```

- **Ternary Operator:** Kurzform von `if-else`.

```
result = (x > 5) ? 10 : 20; // Wenn x > 5, result = 10, sonst 20
```

3. Switch

- Wird verwendet, um mehrere Bedingungen zu prüfen.

```
switch (x) {  
    case 1:  
        // Code  
        break;  
    case 2:  
        // Code  
        break;  
    default:  
        // Code  
}
```

4. While/Do While

- **While:** Schleife, die solange läuft, wie die Bedingung wahr ist.

```
while (x < 10) {  
    // Code  
}
```

- **Do While:** Schleife, die mindestens einmal läuft, auch wenn die Bedingung falsch ist.

```
do {  
    // Code  
} while (x < 10);
```

5. For

- Schleife, die eine bestimmte Anzahl von Wiederholungen durchführt.

```
for (int i = 0; i < 10; i++) {  
    // Code  
}
```

6. Arrays

- **Eindimensionale Arrays:** Sammlung von Werten gleichen Typs.

```
int arr[5] = {1, 2, 3, 4, 5};
```

- **Mehrdimensionale Arrays:** Arrays mit mehreren Dimensionen.

```
int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

7. Bitweise Operationen

- **NOT (~):** Invertiert alle Bits.

```
~x;
```

- **AND (&):** Setzt ein Bit nur, wenn beide Bits 1 sind.

```
x & y;
```

- **OR (|)**: Setzt ein Bit, wenn eines der Bits 1 ist.

```
x | y;
```

- **XOR (^)**: Setzt ein Bit nur, wenn die Bits unterschiedlich sind.

```
x ^ y;
```

- **Shiften**: Verschiebt Bits nach links oder rechts.

```
x << 2; // Links verschieben  
x >> 2; // Rechts verschieben
```

8. Funktionen

- Funktionen bieten Wiederverwendbarkeit und Modularität im Code.

```
int add(int a, int b) {  
    return a + b;  
}
```

9. Pointer

- **Deklaration**: Zeiger auf eine Variable.

```
int *p;
```

- **&-Operator**: Adressoperator (gibt die Adresse einer Variablen).

```
p = &x;
```

- ***-Operator**: Dereferenzierung (Zugriff auf den Wert an einer Adresse).

```
*p = 5;
```

- **Call by Reference:** Übergibt die Adresse einer Variablen, sodass die Funktion den Wert ändern kann.

```
void changeValue(int *x) {  
    *x = 10;  
}
```

10. Pointer/Array-Konvertierung

- Arrays und Pointer sind eng miteinander verbunden. Ein Array-Name ist ein Zeiger auf das erste Element.

```
int arr[5] = {1, 2, 3, 4, 5};  
int *p = arr; // p zeigt auf arr[0]
```

11. Strings

- **Char Arrays:** Zeichenketten werden als Arrays von `char` gespeichert, diese werden standardmäßig mit einem `'\0'` beendet.

```
char str[20] = "Hallo";
```

- **Funktionen:**
- `strcat(str1, str2)` – Verkettet zwei Strings.
- `strcpy(dest, src)` – Kopiert einen String.
- `strcmp(str1, str2)` – Vergleicht zwei Strings.
- `strlen(str)` – Gibt die Länge eines Strings zurück.

12. Dynamischer Speicher

- **malloc/calloc:** Reservieren von dynamischem Speicher.

```
int *arr = malloc(10 * sizeof(int)); // malloc
int *arr2 = calloc(10, sizeof(int)); // calloc
if (arr == NULL) return 0; // Checken, ob Speicher erhalten
```

- **free:** Gibt den Speicher wieder frei.

```
free(arr);
```

- **memset:** Setzt einen Speicherbereich auf einen bestimmten Wert.

```
memset(arr, 0, 10 * sizeof(int));
```

13. Structs

- **Structs:** Datentyp, der verschiedene Datentypen zusammenfasst.

```
struct Person {
    char name[50];
    int alter;
};
```

- **Punktoperator und Pfeiloperator:**
- Punktoperator (.): Zugriff auf Mitglieder einer Struktur.
- Pfeiloperator (->): Zugriff auf Mitglieder eines Strukturzeigers.

```
struct Person p1;
p1 .alter = 30;

struct Person *ptr = &p1;
ptr ->alter = 30;
```

14. Unions

- **Unions:** Wie eine Struktur, aber alle Mitglieder teilen sich den gleichen Speicherbereich.

```
union Data {  
    int i;  
    float f;  
    char c;  
};
```

15. Enums

- **Enums:** Definieren von benannten Konstanten.

```
enum Wochentage { Montag, Dienstag, Mittwoch, Donnerstag, Freitag };
```

16. Header-Dateien

- Header-Dateien enthalten Funktionsprototypen, Strukturdefinitionen und Konstanten.

```
#include <stdio.h>
```

17. Datei Input/Output

- **Dateioperationen:**
 - `fopen` – Öffnet eine Datei.
 - `fclose` – Schließt eine Datei.
 - `fseek` – Setzt den Dateizeiger auf die angegebene Position.
 - `ftell` – Gibt die aktuelle Position des Dateizeigers zurück.
 - `rewind` – Setzt den Dateizeiger auf den Anfang.
 - `fread` und `fwrite` – Lesen und Schreiben von Daten (beliebig).
 - `fgetc` und `fputc` – Lesen und Schreiben von einem Zeichen.
 - `fgets` und `fputs` – Lesen und Schreiben von Strings (bis zu einem Zeilenumbruch).

18. Präprozessor-Direktiven

- **#define:** Definiert Makros.

```
#define PI 3.14
```

- **#ifndef:** Überprüft, ob ein Makro noch nicht definiert ist.

```
#ifndef PI  
    #define PI 3.14  
#endif
```