1. Einfache Struktur mit Ausgabe

Definiere eine Struktur Person mit den Feldern `name` (String) und `alter` (Ganzzahl). Erstelle eine Variable dieser Struktur und gib die Werte aus.

2. Zeiger auf eine Struktur

Nutze einen Zeiger auf eine Struktur, um die Werte einer Person-Struktur auszugeben.

3. Struktur per Funktion übergeben

Schreibe eine Funktion printPerson, die eine Struktur als Argument erhält und die Werte ausgibt.

4. Struktur per Zeiger an Funktion übergeben

Erweitere die vorherige Aufgabe so, dass die Struktur per Zeiger übergeben wird.

5. Dynamische Speicherallokation für eine Struktur

Erstelle eine Struktur Person dynamisch mit `malloc` und gib die Werte aus.

6. Struktur innerhalb einer Struktur

Definiere eine Struktur Adresse mit `stadt` und `plz`. Füge diese in eine `Person`-Struktur ein.

7. Array von Strukturen

Speichere mehrere `Person`-Strukturen in einem Array und gib sie aus.

8. Zeiger auf ein Array von Strukturen

Erweitere die vorherige Aufgabe, indem du einen Zeiger verwendest.

9. Struktur mit Zeiger auf Zeichenkette

Nutze einen `char *`–Zeiger für den Namen in der `Person`-Struktur.

10. Verkettete Liste mit Strukturen

Implementiere eine einfache verkettete Liste mit der Struktur `Node`, die einen `int`-Wert und einen Zeiger auf das nächste Element enthält.

11. Verkettete Liste mit Einfüge- und Löschoperationen

Erweitere die vorherige verkettete Liste um Funktionen zum **Einfügen am Ende** und **Löschen eines Knotens** nach Wert.

12. Dynamisches Array von Strukturen

Erstelle ein **dynamisches Array von Personen**, dessen Größe vom Benutzer festgelegt wird.

13. Struktur mit Funktionszeigern

Definiere eine Struktur Rechner, die zwei Integer-Werte speichert und **Funktionszeiger für Addition und Multiplikation** enthält.

14. Binärbaum mit Strukturen

Erstelle eine Binärbaum-Struktur mit einer Funktion zum **Einfügen eines Wertes** und **Inorder-Traversierung**.

Ein Binärsuchbaum ist eine hierarchische Datenstruktur, in der jeder Knoten höchstens zwei Kindknoten hat:

- **Linker Kindknoten** enthält kleinere Werte als der aktuelle Knoten.
- **Rechter Kindknoten** enthält größere Werte als der aktuelle Knoten.

Die Implementierung umfasst drei Hauptfunktionen:

1. **Erstellen eines neuen Knotens (newNode)**
2. **Einfügen eines Wertes in den Baum (insert)**
3. **Inorder-Traversierung (inorder)**

15. Hash-Tabelle mit Verkettung

Implementiere eine **einfache Hash-Tabelle** mit **Verkettung für Kollisionen**