Prof. Dr. Christian Braunagel

Programmentwurf – Parkhaus-Simulation in C (Teil I)

Kurs: TSA/TSL 25

Vorlesung: Programmieren 1

Studiengang: Embedded Systems

Quartal: Q1 2026

Start des Projekts: 18.02.2026

Finaler Abgabetermin (Teil I & II): 18.03.2026

Bearbeitungszeit: 4 Wochen

Gruppengröße: 3 Studierende

1. Wichtige Informationen

Dies ist der erste von zwei Teilen der Aufgabenstellung für den Programmentwurf „Parkhaus-Simulation in C“. Teil I ist bis zum 03.03.2026 bis 23:59 Uhr (CET) digital abzugeben. Für mehr Details siehe Abschnitt 4 „Abgabeumfang für Teil I“.

Sollten Fragen zur Aufgabenstellung oder Schwierigkeiten bei der Bearbeitung auftreten, melden Sie sich rechtzeitig per E-Mail: braunagel@dhbw-ravensburg.de.

2. Ziel der Aufgabe

Ziel dieser Aufgabe ist die eigenständige Entwicklung im Team eines strukturierten Programms in der Programmiersprache C. Anhand einer Parkhaus-Simulation sollen grundlegende Konzepte der Programmierung, des Algorithmenentwurfs sowie der Teamarbeit angewendet werden.

3. Problemstellung

Zur Bewertung der Auslastung und möglicher baulicher Erweiterungen, sollen Sie eine Simulation des Parkhauses „Rauenegg“ erstellen. Das Parkhaus verfügt über eine definierte Anzahl an Stellplätzen und Fahrzeuge kommen entsprechend einem definierten Zeitverlauf an, parken für eine bestimmte Dauer und verlassen anschließend das Parkhaus. Ist das Parkhaus voll, müssen ankommende Fahrzeuge vor dem Parkhaus in der Warteschlange stehen. Dieser Prozess soll über diskrete Zeitschritte simuliert werden.

3.1. **Funktionale Anforderungen**

Das Programm soll folgende Anforderungen erfüllen:

- Alle Simulationsparameter müssen zur Laufzeit über das Terminal konfigurierbar sein:
  - Anzahl von Stellplätzen
  - Maximale Parkdauer
  - Simulationsdauer (Zeitschritte)
  - Ankunftswahrscheinlichkeit neuer Fahrzeuge in %
  - Zufalls-Seed für reproduzierbare Simulationen
- Ungültige Eingaben müssen erkannt und abgefangen werden
- Das Parkhaus besteht aus:
  - Einer festen Anzahl von Stellplätzen
  - Einer Verwaltung der aktuell parkenden Fahrzeuge
- Jeder Stellplatz ist entweder leer oder belegt.
- Fahrzeuge besitzen mindestens:
  - Eine eindeutige ID
  - Verbleibende Parkdauer
  - Zeitpunkt der Einfahrt
- Fahrzeuge haben eine zufällige Parkdauer
- Es existiert eine maximale Parkdauer, nach der das Fahrzeug das Parkhaus spätestens verlassen muss
- Einparken ist nur bei freien Stellplätzen möglich
- Sind keine freien Stellplätze vorhanden, warten die Fahrzeuge in einer Warteschlange vor dem Parkhaus.
- Fahrzeuge verlassen das Parkhaus nach Ablauf der Parkdauer
- Ausgabe von mind. 5 geeigneten Statistiken pro Zeitschritt sowie am Ende der Simulation, um die Auslastung des Parkhauses und mögliche bauliche Erweiterungen bewerten zu können.
- Die Statistiken pro Zeitschritt sollen auf der Konsole in einem lesbaren Format ausgegeben sowie in eine Text-Datei geschrieben werden.

3.2. Technische Vorgaben

- Programmiersprache: C
- Versionierungswerkzeug: git
- Verwendung Remote Repository: GitHub unter github.com

- Arbeitsplanung im Team über: GitHub Projects unter github.com
- Verwendung passender Datenstrukturen zur Modellierung von Fahrzeugen & Parkhaus
- Die Warteschlange vor dem Parkhaus soll über eine dynamische Speicherverwaltung erfolgen.
- Sinnvolle Aufteilung in Funktionen
- Modularisierung Ihres Programms, um Wartbarkeit und Erweiterbarkeit sicherzustellen
- Saubere Kommentierung des Codes
- Berücksichtigung eines idiomatischen Programmierstils und Konventionen aus der Vorlesung

4. Abgabeumfang für Teil I

Für den ersten Teil dieses Projekts sind bis zum genannten Termin in Abschnitt I „Wichtige Informationen“ folgende Dokumente abzugeben:

a) Erste Quellcode- und Header-Dateien
b) Übersicht zu den geplanten Statistiken
c) Format der geplanten Ausgabe z.B. als Grafik, Screenshot oder Konsolenausgabe mit Testdaten
d) Übersicht zu den festgelegten Datentypen und Strukturen (z.B. direkt im Quellcode oder Header-Dateien)
e) Funktionsprototypen zu den geplanten Funktionen mit einheitlichem Header (siehe Anlage als Beispiel)
f) Jeweils ein verständlicher und nachvollziehbarer Begründungstext für alle oben genannten Punkte
g) Beschreiben Sie das geplante Programm komplett in auskommentierten Pseudocode in den dazugehörigen Quellcode- und Header-Dateien
h) Mindestens 3 zentrale Funktionen (d.h. nicht main() oder kleinere Hilfsfunktionen wie z.B. swapArrays()) sollen als Flussdiagramm dargestellt werden.

Das Programm muss zu diesem Zeitpunkt NICHT kompilierbar sein. Es soll stattdessen ein erstes Gerüst des geplanten Programms ersichtlich sein. Quellcode- und Header-Dateien sowie jegliche andere Dokumentation findet sich in Ihrem GitHub Repository.

Die Abgabe erfolgt über Moodle (siehe Bereich „Programmentwurf“) vor der genannten Deadline (siehe auch Abschnitt „Wichtige Informationen“) mit folgendem Inhalt:

- ein Dokument mit Ihren Namen, Matrikelnummern und dem Datum sowie dem Link zu Ihrem öffentlich geschalteten Repository auf github.com
- ein zip-File Ihres GitHub Repositories

- Link und Zugriff auf ihr Project (Kanban Board) auf github.com
- weitere Dokumente mit durchgeführten Aufgaben wie Diagramme oder Dokumentation, wenn diese nicht in Ihrem Repository enthalten sind

Bitte achten Sie darauf Ihr GitHub Repository „öffentlich“ sichtbar zu machen und mich als Mitglied für Ihr GitHub Project hinzuzufügen. Ansonsten ist eine vollständige Bewertung nicht möglich.

5. Bewertungskriterien für Teil I und Teil II

ANTEIL	KRITERIUM	INHALT
35 %	Funktionale Korrektheit und Algorithmische Umsetzung	Programm kompilierbar auf GitHub Codespace; Funktionale Anforderungen erfüllt; Programmausgabe sinnvoll und übersichtlich; erzeugte Unit-Tests laufen fehlerfrei durch; Schwache Statistiken führen zu Punktabzug
25 %	Code-Struktur und Programmierstil	Lesbarer und erweiterbarer Code; Modularisierung beachtet; Integrierung von Fehlervermeidungskonzepten; sinnvolle Wahl der Unit-Tests; Berücksichtigung unserer Code-Convention
25 %	Teamarbeit	Verwendung gemeinsames Planungs-Tool; Klare Aufgabenteilung in Planungstool ersichtlich; Balance der Code-Beiträge der Teammitglieder sowie Entwicklungsprozess in git ersichtlich (regelmäßige, atomare Commits mit sauberen Commit-Messages); Ersichtliche Diskussionen/Reviews in Pull-Request
15 %	Dokumentation & Begründung	Kommentierter Quellcode; alle geforderten Dokumente vorhanden; Qualität und Fehlerfreiheit der Dokumentation; Nachvollziehbare Entscheidungsprozesse

6. Anlagen

- **Template für Funktionsheader-Dokumentation**

```
/**
 * @brief Kurze Beschreibung der Funktion.
 *
 * Ausfuehrlichere Beschreibung, falls erforderlich, was die
 * Funktion genau macht.
 *
 * @param[in]  param1 Beschreibung von Parameter 1 (Eingabe)
 * @param[out] param2 Beschreibung von Parameter 2
 *                    (Ausgabe/Zeiger)
 * @return     Beschreibung des Rueckgabewerts
 */
Rückgabetyp Funktionsname(Datentyp1 param1, Datentyp2 *param2) {
    // Implementierung
}
```