SYSTEMPROGRAMMIERUNG: 03. Aufgabenblatt

WS 2013/2014
Prof. Dr.-Ing. C. Gips, Dipl.-Inf. Birgit C. George

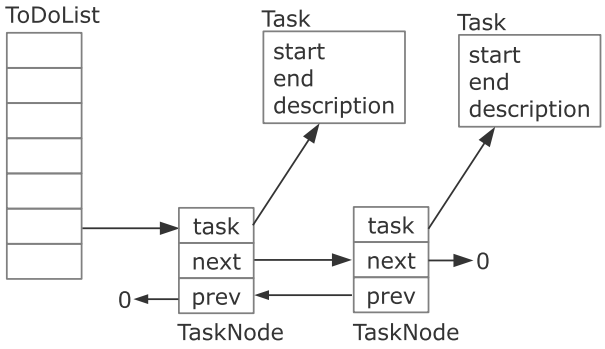
FH Bielefeld
University of
Applied Sciences

Dynamische Speicherverwaltung, GDB

Ausgabe:	**24.10.2013**
Abgabe:	**KW45 (05./08.11.2013)**
Punkte:	**20+0**

Achtung: 2 Wochen Bearbeitungszeit, kein Praktikum in KW44 (Feiertag)

Erstellen Sie eine ToDo-Liste zur Verwaltung von Aufgaben einer Woche in ANSI-C.
Skizze der Datenstrukturen:



Achten Sie darauf, *alle* von Ihnen belegten Ressourcen auf dem Heap-Speicher wieder freizugeben, sobald diese nicht mehr benötigt werden!

1. Aufgabe: Strukturen und Datentypen — 4 Punkte

Deklarieren Sie zunächst geeignete Datentypen (als Strukturen):

1. Eine Aufgabe (`Task`) umfasst die Informationen `start` und `end` (Stunden mit Werten von 0 bis 24) und einen String `description` (nutzen Sie `char[]`), der die Beschreibung der Aufgabe enthält. Definieren und nutzen Sie eine (Präprozessor-) Konstante für die maximale Länge der `description`.
2. Die Aufgaben eines Tages sollen in einer doppelt verketteten Liste abgelegt werden. Definieren Sie sich dazu eine Struktur `TaskNode`, die einen Pointer auf einen konkreten `Task` sowie Pointer auf den jeweils nächsten und vorigen Knoten in der Liste (Typ `TaskNode`) enthält.

Implementieren Sie eine Funktion `void showAll()`, mit der die gesamte ToDo-Liste übersichtlich auf der Konsole ausgegeben werden kann. Wie muss die Signatur dieser Funktion aussehen, damit man sie mit der in `main()` definierten ToDo-Liste (vgl. nächste Aufgabe) aufrufen kann?

Ziel: Umgang mit Strukturen und verketteten Listen

2. Aufgabe: Aufzählungen — 2 Punkte

Die ToDo-Liste (`ToDoList`) umfasst 7 einzelne Tageslisten und soll als Array von Pointern auf die Struktur `TaskNode` implementiert werden. Die ToDo-Liste soll als Variable in der `main()`-Funktion (und nicht als globale Variable) realisiert werden.

Definieren Sie einen Aufzählungstyp `Day`, der den Zugriff auf die Tageslisten erleichtert.

Ziel: Umgang mit einfachen Aufzählungen

3. Aufgabe: Dynamische Speicherverwaltung — 4 Punkte

Schreiben Sie eine Funktion `void deleteDayList(TaskNode*)`, die den gesamten von einer Tagesliste belegten Heapspeicher wieder freigibt. Beachten Sie, daß diese Funktion nicht notwendigerweise mit dem Pointer auf den ersten Eintrag der Tagesliste aufgerufen wird!

Hinweis: Der Aufrufer der Funktion sollte nach dem erfolgreichen Freigeben der Tagesliste den entsprechenden Eintrag in der ToDo-Liste zurücksetzen, damit nicht versehentlich auf nicht mehr gültigen Speicher zugegriffen werden kann. Sie können diese beiden Schritte auch in einer weiteren Funktion kapseln.

Hinweis: Schreiben Sie sich eine Hilfsfunktion `void deleteTask(TaskNode*)`, die den Speicher für den übergebenen Eintrag (sowohl `Task` als auch `TaskNode`) freigibt.

Ziel: Freigabe von Speicher, Iteration durch Liste

4. Aufgabe: Listen durchlaufen — 6 Punkte

Implementieren Sie eine Funktion `int insertTask(TaskNode*, Task*)`, die der per Pointer übergegebenen Tagesliste einen neuen Eintrag mit der (ebenfalls per Pointer übergebenen) neuen Aufgabe hinzufügt. Beachten Sie, daß diese Funktion nicht notwendigerweise mit dem Pointer auf den ersten Eintrag der Tagesliste aufgerufen wird!

Die Funktion gibt den Wert 0 zurück, falls die Aufgabe korrekt eingefügt werden konnte. Überlegen Sie sich mögliche Fehlerfälle und geben entsprechende Fehlerwerte zurück.

Beachten Sie folgende Einschränkungen:

- Die Aufgaben eines Tages sind zeitlich sortiert.
- Die Aufgaben eines Tages dürfen sich zeitlich nicht überlappen.

Ziel: Iteration durch verkettete Liste

5. Aufgabe: GDB — 4 Punkte

Arbeiten Sie sich selbstständig in die Nutzung des Debuggers aus dem GCC-Paket (GDB) ein.

Schreiben Sie sich ein Hauptprogramm, in dem Sie den vorangegangenen Aufgabenteilen implementierte Funktionalität nutzen bzw. aufrufen. Demonstrieren Sie in der Abgabe, wie Sie Ihr Programm mit dem GDB von der Konsole aus debuggen können. Dabei müssen Sie mindestens die bereits vom Debuggen unter Java bekannten Funktionalitäten zeigen: Breakpoints setzen und löschen, sich Variableninhalte anzeigen lassen sowie in eine Funktion springen bzw. zum nächsten Breakpoint fortfahren.

Ziel: Einarbeitung in den GDB