Programmieraufgaben für TINF21CS1, 1.Sem.

1. C-Projekt „studenten“ verbessern und erweitern

Öffnen Sie das Projekt „studenten“ in Visual Studio!
Hinweis: Die meisten Aufgaben lassen sich unabhängig voneinander bearbeiten. Wenn eine bestimmte Aufgabe nicht gelingen sollte, können Sie also oft Ihre aktuelle Dateiversion unter einem sinnvollen Namen speichern und entweder mit dem vorhergehenden Stand oder sogar mit der ursprünglich gegebenen Datei weiterarbeiten.

1.1 Fügen Sie eine Header-Datei namens „datentypen.h“ hinzu, in der Sie die Datentyp-Definitionen unter „// Eigene Datentypen“ auslagern!

1.2 Ersetzen Sie den festen Initialisierungswert bei „studenten_feld_n“ durch den Formelausdruck wie in unseren Übungsprogrammen!

1.3 Die Funktion „vgl_fkt_kurs_nachname_vorname()“ soll so ergänzt werden, dass sich eine aufsteigende Sortierung der Studenten nach dem Kurs, dem Nachnamen und dem Vornamen ergibt.
Hilfestellung (Prototyp in string.h, vergleicht 2 Texte):

```
int strcmp(const char* str1_ptr, const char* str2_ptr);
```

1.4 Nachdem das Feld sortiert worden ist, kann man darin mit der Standardfunktion zur Halbierungssuche nach einem Element suchen. Entsprechender Test-Code ist schon vorbereitet, aber nicht ganz vollständig – Sie müssen ihn ergänzen!
Hilfestellung (Prototypen in stdlib.h; _strdup() reserviert den genau notwendigen Speicherplatz auf dem Heap und kopiert den gegebenen Text dorthin; bsearch() schließt sich nahtlos an qsort() an und wird ab dem 2. Parameter völlig gleichartig aufgerufen; das zu suchende Element muss genauso aufgebaut sein wie die Elemente des Feldes, in dem die Binärsuche erfolgen soll):

```
char* _strdup(const char* text_ptr);
void* bsearch(const void *vergleichselement_ptr, const void*feld_ptr, size_t feld_n,
        size_t element_groesse, int(*vergleichs_fkt_ptr)(const void*, const void*));
```

1.5 Wie Sie leicht feststellen können, ist der gefundene Treffer nicht der einzig mögliche. Was müssten Sie also eigentlich noch programmieren, um alle möglichen Treffer herauszufinden? Welche Art von Datenstruktur wäre hier aus welchem Grunde zu erwägen? – Kurze Erklärungen! (Ihre Antwort können Sie auch gerne hier von Hand eintragen!)

1.6 Im gegebenen Programm muss wiederholt der vollständige Code zum Ausgeben des Feldes mit Studenten angeschrieben werden.
Programmieren Sie daher eine Funktion „studenten_feld_ausgeben( )“, die ein Feld mit Studenten ausgibt. Sehen Sie zusätzlich zur Anfangsadresse des Feldes und der Anzahl der Studenten einen Funktionszeiger als dritten Funktionsparameter vor, damit man beim Aufrufen einstellen kann, welche Funktion die Ausgabe übernimmt. Der Typ der Ausgabefunktion soll demjenigen der beiden vorhandenen Funktionen „drucke_studenten_kurz_by_val( )“ und „drucke_studenten_lang_by_val( )“ entsprechen.
Vergessen Sie den Prototyp nicht!

1.7 Testen Sie Ihre neue Funktion „studenten_feld_ausgeben( )“, indem Sie den Quelltext für die Ausgaben vor und nach dem ersten Aufruf von „qsort( )“ durch eine Anweisung mit „if(0) … else …“ lahmlegen und stattdessen jeweils Ihre neue Funktion aufrufen!

1.8 Meistens ist es einfacher und weniger aufwendig, nur die Adresse des auszugebenden Objektes an die entsprechende Funktion zu übergeben. Entwickeln Sie daher an Hand der Funktion „drucke_studenten_kurz_by_val( )“ die Funktion „drucke_studenten_kurz_by_ref( )“, die nur die Adresse des auszugebenden Studenten als Parameter erwartet und nicht die vollständige Kopie eines Studenten! Verwenden Sie dabei jede der beiden möglichen Schreibweisen genau einmal!
Vergessen Sie den Prototyp nicht!
Zum Testen ergänzen und aktivieren Sie den vorgesehenen Aufruf in der Funktion „main( )“!

1.9 Ein Mathematiker möchte bei seiner auch von Fortran her gewohnten Indizierung mit 1 bis n bleiben. Können Sie ihm mittels der bereits vereinbarten Variablen „studenten_feld_1_bis_n_ptr“ durch eine schlaue Initialisierung helfen?
Aktivieren Sie sodann den entsprechende Test-Code in der Funktion „main( )“, um ihm zu zeigen, wie gut Ihre Lösung funktioniert!

1.10 Im gegebenen Programm liegen die Studenten unmittelbar im Feld. Viel besser ist es aber, wie bei Java die Studenten als einzelne Objekte auf dem Heap anzulegen und im Feld nur deren Adressen abzuspeichern. Entsprechender Code ist schon teilweise vorbereitet.
Ihre Aufgabe besteht deshalb darin, die Funktion „kopiere_studenten_auf_heap( )“ auszuprogrammieren, die den übergebenen Studenten auf den Heap kopiert. Denken Sie daran, dass auch der Vor- und der Nachname zu kopieren sind.
Falls der Heap beim Belegen des Studenten-Objektes voll sein sollte, geben Sie auf den Fehlerausgabekanal eine Meldung aus, die den Namen der Quelldatei, die aktuelle Zeilenummer sowie den Datums- und Zeitstempel enthält. Beenden Sie außerdem das Programm mit dem Status-Wert EXIT_FAILURE.
Bei den Vor- und Nachnamen dürfen Sie aus Vereinfachungsgründen auf diese Prüfung verzichten.
Aktivieren Sie anschließend zum Testen den vorbereiteten Code in der Funktion „main( )“!
Hilfestellung:

```
char* _strdup(const char* text_ptr);
void* calloc(size_t n_elem, size_t element_groesse);
void* malloc(size_t groesse);
void* realloc(void* heap_ptr, size_t groesse);
void free(void* heap_ptr);
```

1.11 Wenn das soeben erstellte Zeigerfeld sortiert werden soll, können nicht mehr die bisherigen Vergleichsfunktionen verwendet werden. Schreiben Sie daher die Funktion „vgl_fkt_zeiger_matrikel_nr( )“ an, welche bei den durch das Zeigerfeld angebundenen Studenten für eine Sortierung aufsteigend nach ihren Matrikelnummern sorgt. Da die Datenstrukturen jetzt etwas aufwendiger sind, fertigen Sie am besten zuerst eine kleine Skizze der Datenstrukturen an und vergessen darin nicht die Zeiger v1_ptr und v2_ptr, die an die Vergleichsfunktion beim Callback

übergeben werden und die auf 2 von Ihnen willkürlich ausgewählte Feldelemente zeigen. Aktivieren Sie zum Testen den schon vorbereiteten Code in der Funktion „main( )“!

1.12 Erweitern Sie Ihr Programm so, dass sämtlicher von Ihnen belegter Heap-Speicher wieder sauber freigegeben wird, bevor das Programm beendet wird. Korrigieren Sie dabei auch sämtliche Speicheradressen, die zuvor noch auf den Heap verwiesen haben.

1.13 Erstellen Sie ein neues C-Modul namens „ausgaben.c“ einschließlich der zugehörigen Header-Datei und lagern Sie dorthin die Ausgabefunktionen „drucke_studenten_lang_by_val( )“, „drucke_studenten_kurz_by_val( )“ und „drucke_studenten_kurz_by_ref( )“ aus! Rufen Sie sich ins Gedächtnis zurück, warum die Header-Datei einerseits in der C-Datei mit der Funktion „main( )“ und andererseits in der Datei „ausgaben.c“ einzufügen ist!

2. C-Projekt „verallgemeinern“ fertigstellen

Hinweis: *Dieses Projekt wird Ihnen möglicherweise schwerer fallen. Unternehmen Sie zumindest einen Versuch, es umzusetzen, wenden Sie aber nicht zu viel Zeit auf, wenn es Ihnen nicht vollständig gelingen sollte!*

2.1 Arbeiten Sie jetzt mit dem Projekt „verallgemeinern“. Die darin enthaltene C-Quelldatei „verallgemeinern01.c“ enthält eine leicht überarbeitete Fassung des Beispielprogramms „quicksrt.c“ von Goll (S.490 in der 7. Aufl.). Lassen Sie das Programm laufen! Verdeutlichen Sie sich den ungefähren Ablauf; wie Sie sehen können, dient die Funktion „quickSort( )“ nur als Einstieg in rekursiv umgesetzte Sortierfunktion „zerlege( )“.

2.2 Erstellen Sie eine neue Funktion „vgl_fkt_int_aufsteigend( )“, die zwei Ganzzahlen miteinander derartig vergleicht, dass sich eine aufsteigende Sortierung ergibt. – Welche sehr einfache Quelltextvariante ist hier im Gegensatz zu Gleitkommazahlen möglich?

2.3 Erstellen Sie eine neue Funktion „mein_zerlegen( )“, indem Sie die Funktion „zerlege( )“ kopieren und umbenennen. Bevor Sie den Prototyp erstellen, überlegen Sie sich, wie der Kopf dieser Funktion umgestaltet werden muss, damit sie verallgemeinert werden kann und dadurch befähigt wird, mit Feldern beliebigen Datentyps umzugehen. Verwenden Sie den verallgemeinerten Kopf sodann für den zugehörigen Prototyp.

2.4 Erstellen Sie eine neue Funktion „mein_qsort( )“, die genau dieselbe erweiterte Funktionssignatur wie „qsort( )“ aufweist. Rufen Sie in dieser Funktion jedoch nicht wie Goll „zerlege( )“ auf, sondern Ihre eigene Funktion „mein_zerlegen( )“ mit den zutreffenden Parametern.

2.5 Fügen Sie in der Funktion „main ( )“ beim Aufruf von „quickSort( )“ eine „if(0)…else…“-Anweisung ein und sehen Sie als Alternative zum Aufrufen von „quickSort( )“ das Aufrufen Ihrer Funktion „mein_qsort( )“ vor.

2.6 Widmen Sie sich jetzt dem Umwandeln Ihrer Funktion „mein_zerlegen( )“ in die allgemeine Form. Bedenken Sie, dass keine direkten Vergleiche zwischen Feldelementen mehr erlaubt sind, sondern stets die Vergleichs-Callback-Funktion zu rufen ist. Berücksichtigen Sie weiterhin, dass Sie die void-Zeiger immer zuerst in char*-Zeiger umwandeln müssen. Direkte Zugriffe auf Feldinhalte sind in allgemeiner Form nicht mehr möglich. Zum Tauschen von Elementen muss auf den Trick zurückgegriffen werden, dass in einer Schleife so viele Bytes miteinander getauscht werden, wie es der Groesse der Feldelemente entspricht. Vergessen Sie außerdem nicht, dass Sie jeden Offset bzw. Index, den Sie auf einen char*-Zeiger anwenden, jetzt selbst mit der Feldelementgröße multiplizieren müssen.

2.7 Testen Sie das Programm mit Ihren neuen Sortierfunktionen. - Wenn es richtig arbeiten sollte, können Sie zusätzlich den Datentyp des Feldes von int nach double ändern sowie die Formatierungsplatzhalter der zutreffenden Funktionsaufrufe von „printf( )“ anpassen. Die Testausgaben in „mein_zerlegen( )“ müssen Sie entweder lahmlegen, anpassen oder falsche Ausgaben hinnehmen. Außerdem benötigen Sie eine weitere Vergleichsfunktion „vgl_fkt_double_aufsteigend( )“. – Funktioniert das Sortieren auch bei Gleitkommazahlen?

2.8 Außerdem können Sie die neuen Sortierfunktionen in das andere Projekt übernehmen und dort an Stelle von „qsort( )“ aufrufen. – Verhalten sich Ihre verallgemeinerten Sortierfunktionen auch dort richtig?