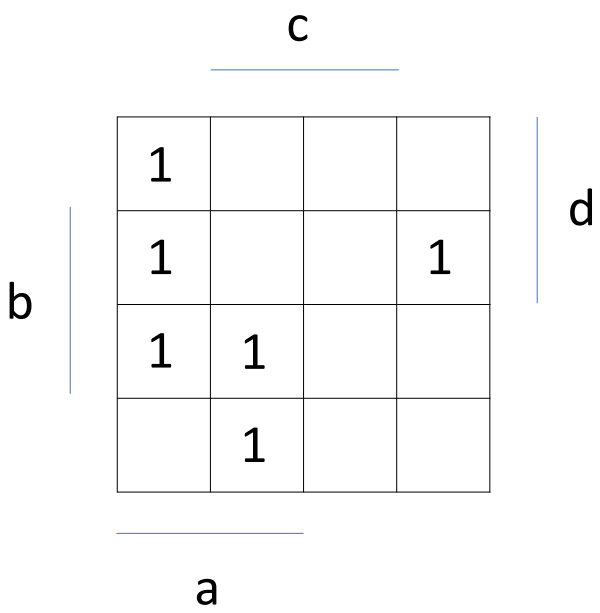
# Programmieraufgabe für TINF22CS1, 1.Sem.

## *1. Boolesche Algebra*

Gegeben ist das folgende KV-Diagramm:




1.1 Schreiben Sie die sich ergebende Boolesche Funktion als vollständige disjunktive Normalform an! (Dort, wo ein Strich gezeichnet ist, soll die entsprechende Eingangsvariable den Wert 1 haben.)

1.2 Vereinfachen Sie die Funktion, indem Sie Primimplikanden (= größtmögliche Gruppen von 1) einzeichnen. Schreiben Sie außerdem die beiden möglichen Funktionsvarianten an!

1.3 Entwickeln Sie ein C-Programm, das die Wertetabelle zum obigen KV-Diagramm erzeugt! Verwenden Sie nur logische Operanden. Sehen Sie die 4 Spalten a, b, c und d sowie die Spalte für den Funktionswert vor. Ergänzen Sie außerdem ganz links drei Spalten, die den Hexadezimal-, den Dezimal- und den Oktalwert zu der jeweiligen Eingangswertekombination angeben. Arbeiten Sie mit je einer Schleife für jede der Eingangsvariablen. Vergessen Sie nicht die Spaltenköpfe ähnlich wie in unserem gemeinsam entwickelten Programm!

1.4 Tragen Sie in das KV-Diagramm die entsprechend Dezimalzahlen in jedes der Felder ein.

1.5 Erstellen Sie eine Kopie des Quelltextabschnittes mit den geschachtelten Schleifen und durchlaufen Sie erneut die 16 Eingangskombinationen, verwenden Sie jedoch diesmal nur eine einzige Schleife und berechnen Sie aus der Laufvariablen mittels der Ganzzahldivision und der Modulfunktion den jeweiligen Wert der 4 Eingangsvariablen. – Funktioniert auch diese Quelltextvariante?

## 2. Bit-Operationen

2.1 Gegeben sind 2 Ganzzahlvariablen „s1“ und „s2“ vom Typ „short“. Diese sollen mittels Bit-Operationen in einer Ganzzahlvariablen „ziel“ vom Typ „int“ (Annahme: hat 4 Byte) vereinigt werden, und zwar auf die Weise, dass die höchstwertige Hexadezimalstelle von „s1“ in die höchstwertigen 4 Bits von „ziel“ wandert. In die Bits mit den Wertigkeiten $2^{27}$ bis $2^{24}$ von „ziel“ wandert die höchstwertige Hexadezimalstelle von „s2“. In die Bits mit den Wertigkeiten $2^{23}$ bis $2^{20}$ soll die zweithöchstwertige Hexadezimalstelle von „s1“ kopiert werden, und in die Bits $2^{19}$ bis $2^{16}$ die zweithöchstwertige Hexadezimalstelle von „s2“. Dieses Vorgehen setzt sich bis zur niederwertigen Stelle von „s2“ fort. – Schreiben Sie ein entsprechendes C-Programm, das diese Funktionalität umsetzt! Testen Sie das Programm zuerst mit aussagekräftigen, fest kodierten Werten für „s1“ und „s2“!

2.2 Erweitern Sie Ihr Programm, indem Sie in der Funktion „main“ prüfen, ob es 2 Kommandozeilenparameter gibt. Entnehmen Sie in diesem Falle den beiden Texten die Zahlenwerte für „s1“ und „s2“! Falls mindestens einer der beiden Kommandozeilenparameter keine gültige Ganzzahl im Hexadezimalzahlenformat darstellen sollte, geben Sie eine entsprechende Fehlermeldung auf „stderr“ aus, die den Quelldateinamen, die Zeilennummer, das Datum und die Uhrzeit enthält. Beenden Sie danach das Programm mit dem Statuswert „EXIT_FAILURE“!
Hinweis: Mittels „sscanf()“ stehen Ihnen die Eingabemöglichkeiten wie bei „scanf()“ zur Verfügung, nur dass nicht von „stdin“, sondern aus einem String gelesen wird.

## 3. Arbeiten mit mehreren Quelldateien

3.1 Öffnen Sie unser gemeinsam erstelltes Projekt zum „getraenke_automat“ und bearbeiten Sie dieses weiter, indem Sie ein neues C-Modul namens „ausgaben.c“ hinzufügen. Es soll die Funktionsdefinitionen für die Ausgabefunktionen aufnehmen, wobei es bisher nur eine einzige reine Ausgabefunktion gibt, nämlich „ausgabeGetraenke()“. Führen Sie alle erforderlichen Schritte sorgsam aus!

3.2 Beantworten Sie die folgenden Fragen:

a) Was würde geschehen, wenn Sie die Funktionsdefinition ins neue Modul kopieren und nicht verschieben würden?

b) Wie nennt man das, was die Präprozessor-Direktiven in der Header-Datei bewirken?

c) Welche neuere Alternative gibt es an Stelle des „#ifndef“-Mechanismus?

d) Warum wird die Header-Datei des neuen Moduls in die Haupt-Datei eingefügt?

e) Was sind die beiden (potentiellen) Gründe dafür, dass die Header-Datei eines neuen Moduls auch in dem neuen Modul selbst eingefügt wird?

## *4. 3D-Feld*

4.1 Bearbeiten Sie das Projekt „feld3d_01“ weiter! Schreiben Sie eine Funktion namens „feld3d_init()“, der die Anfangsadresse des Feldes sowie die Anzahl der Feldelemente in jeder der drei Dimensionen übergeben werden. Die Funktion liefert nichts zurück. Sie soll die Initialisierung des Feldes übernehmen, wie wir es bereits in unserer Funktion „main()“ ausprogrammiert haben. Fügen Sie statt dieses Quelltextabschnittes einen geeigneten Aufruf der neuen Funktion ein. Vergessen Sie nicht, den Prototyp zu erstellen. – Funktioniert Ihr Programm genauso gut wie zuvor?

4.2 Am Ende der Funktion „main()“ wird das 3D-Feld mittels „free()“ wieder freigegeben, wobei wir noch verschiedene Kommentare und weitere Anweisungen zur Veranschaulichung vorgesehen haben. Sehen Sie eine „if(0) - else“-Konstruktion (bzw. „if(1) - else“) vor und im Alternativzweig das Freigeben mittels „realloc()“. Weisen Sie den Rückgabewert dieser Funktion „matrix_3d_anf“ zu und geben Sie anschließend den Wert dieser Variablen aus. – Ist er NULL? Was haben Sie dadurch erreicht?

4.3 Erstellen Sie eine neue Funktion namens „felder_erstellen_wie_java()“, die der vorhandenen Funktion „zugriffsfelder_erstellen ()“ ähnlich ist, jedoch zusätzlich für jede Zeile ein eigenständiges Feld belegt. Speichern Sie in jedem Feldelement seinen Index, wie es bisher schon im Abschnitt „*Das Feld initialisieren…*“ geschieht, allerdings dort bezogen auf das zusammenhängende 3D-Feld, hier jedoch bezogen auf die einzelnen Zeilen. Erstellen Sie den Prototyp. Sehen Sie in der Funktion „main()“ eine „if“-„else“-Konstruktion beim Aufruf von „zugriffsfelder_erstellen ()“ vor und rufen Sie Ihre neue Funktion auf. – Funktioniert Ihr Programm noch immer einwandfrei?

4.4 Erstellen Sie eine Kopie der Funktionsdefinition von „zugriffsfelder_erstellen()“ unter dem Namen „zugriffsfelder_erstellen_by_ref()“. Wandeln Sie den Rückgabetyp von „double***“ zu „void“ um. Sehen Sie stattdessen einen zusätzlichen Parameter vor, der es ermöglicht, den Inhalt der lokalen Variablen „matrix_3d“ mittels „Call-by-reference“ zurückzureichen. Erstellen Sie den Prototyp. Sehen Sie beim Aufruf von „zugriffsfelder_erstellen()“ eine weitere „if“-„else“-Konstruktion vor und rufen Sie Ihre neue Funktion „zugriffsfelder_erstellen_by_ref()“. – Funktioniert das Programm wie zuvor?

4.5 Würde es negative Auswirkungen haben, wenn Sie am Ende Ihres C-Programms auf das Freigeben des Heap-Speichers verzichten würden? – Begründen Sie Ihre Antwort!

4.6 Was versteht man unter einer „Speicherleiche“?

4.7 Erklären Sie, was sich unter dem Begriff der Heap-Speicherfragmentierung verbirgt und wodurch diese zu Stande kommt!

4.8 Welche scheinbar merkwürdige Situation kann entstehen, wenn der Heap-Speicher stark fragmentiert ist?

4.9 Grenzen Sie mit einigen wichtigen Gesichtspunkten die Fragmentierung einer Festplatte von der Fragmentierung des Heap-Speichers ab!

4.10 Wieso verliert das Problem der Heap-Speicherfragmentierung mit dem Übergang zu 64-Bit-Adressen selbst dann an Bedeutung, wenn sich der physisch verfügbare Hauptspeicher nur maßvoll gegenüber einem 32-Bit-System erhöht?

4.11 Warum ist diese Anweisung
heap_ptr = (<zieldatentyp>*)realloc(heap_ptr,<neue_groesse>);
falsch, und wie müssen Sie richtig vorgehen? Besteht ein Zusammenhang mit Speicherleichen?

## 5. Verkettete Listen

5.1 Aktivieren Sie das Projekt „flexibeldmem02“. Schlagen Sie Folie 467 auf und vollziehen Sie den Quelltext für das Einfügen eines Elementes rechts von demjenigen Element nach, das durch „liste_ptr“ festgelegt wird. Erstellen Sie anschließend diesem Quelltext entsprechend eine Funktion namens „liste_einfuegen()“. Sie soll nichts zurückgeben und ohne globale Variablen auskommen. Die beiden Zeiger auf den Anfang und das Ende der Liste werden mittels „Call-by-reference“ übergeben. Der Zeiger auf das einzufügende Element (z.B. „neu_ptr“) und auf die Einfügeposition (z.B. „liste_ptr“) werden mittels „Call-by-value“-Technik übergeben.
Denken Sie an den Prototyp.
Sehen Sie an den drei Stellen, wo jeweils ein neues Element in die Liste eingefügt wird, eine „if(0) - else“-Konstruktion vor und nutzen Sie Ihre neue Funktion durch einen geeigneten Funktionsaufruf! – Funktioniert Ihr Programm weiterhin?

5.2 Schreiben Sie eine weitere Funktion namens „liste_loeschen()“. Diese soll den Quelltext der Folie 468 zur Grundlage nehmen; die verschiedenen möglichen Situationen werden auf den folgenden Folien bis Nr. 472 durchgespielt. Auch hier müssen die Zeiger auf den Anfang und das Ende der Liste mittels „Call-by-reference“-Technik übergeben werden. Die Adresse des zu löschenden Elementes wird mittels „Call-by-value“-Technik übergeben. Die Funktion soll als Rückgabewert die Adresse des zu löschenden Elementes zurückliefern, da dies beim Programmieren ggf. nützlich sein kann.
Erstellen Sie den Prototyp.
Vereinbaren Sie zwei Listenzeiger „reserve_anf_ptr“ und „reserve_ende_ptr“ und initialisieren Sie diese auf NULL.
Gehen Sie an das Ende der Funktion „main()“ und löschen Sie das mittlere Element, sodann das letzte und schließlich das erste (und mittlerweile einzige) Element, indem Sie Ihre neue Funktion rufen. Fügen Sie die gelöschten Elemente jeweils in die Reserveliste ein, wobei Sie „liste_einfuegen()“ verwenden können. Testen Sie, ob das Durchlaufen noch funktioniert.

5.3 Das aktuelle Programm hat den Nachteil, dass für jeden Datentyp die verschiedenen Listenfunktionen erneut angeschrieben werden müssen. Das kann man sich ersparen, indem man einen allgemeinen Listentyp (z.B. „listen_typ“) definiert und in ihm zusätzlich zu den beiden Verkettungszeigern nur einen „void“-Zeiger (z.B. „void *daten_ptr;“) vorsieht. Dieser zeigt auf das Datensatzobjekt des betreffenden Listenelementes. Das Datensatzobjekt wird also getrennt von dem Listenelement eigenständig auf dem Heap angelegt. Im Gegensatz zu neueren Programmiersprachen(versionen) muss der Programmierer hierbei stets selbst aufpassen, dass er in einer bestimmten Liste immer nur die Elemente des für sie vorgesehenen Datentyps ablegt.
Wenden Sie dieses Vorgehen auf das vorliegende Programm an, indem Sie das Projekt kopieren. Erstellen Sie sodann eine Kopie der Definition des Datentyps „struct flexibler_struct_typ“ unter dem Namen „struct listen_typ“, in dem Sie nur die Verkettungszeiger und den zuvor beschriebenen „void *daten_ptr;“ vorsehen. Im „struct flexibler_struct_typ“ löschen Sie dagegen die Verkettungszeiger. Passen Sie Ihr Programm weiter an und erstellen Sie jeweils beim Einrichten eines neuen Listenelementes zusätzlich ein Datenobjekt vom Typ „struct flexibler_struct_typ“, dessen Anfangsadresse Sie in den „daten_ptr“ des entsprechenden Listenelementes eintragen.
Ändern Sie in „liste_durchlaufen()“ den Typ des „einstiegs_ptr“ in „listen_typ“ ab und gestalten Sie die Ausgabefunktion so um, dass kein Zeiger auf einen konkreten Datentyp, sondern eine „nackte“ Adresse („const void*“) übergeben wird. Passen Sie entsprechend auch Ihr übriges

Programm an; bedenken Sie, dass jetzt in den Ausgabefunktionen zuerst ein expliziter Typ-Cast stattfinden muss. – Funktioniert Ihr Programm nach der Umstellung noch?

5.4 Was müssen Sie jetzt beachten, wenn Sie den Heap-Speicher eines Elementes freigeben wollen?

## *6. Sortieren und Suchen*

6.1 Im Projekt „sortieren_suchen01“ haben wir kennengelernt, wie man ein Feld von Gleitkommazahlen sortieren kann. Im nächsten Schritt haben wir das Programm so erweitert, dass wir außerdem ein Feld von Ganzzahlen sortieren können. Denselben Erweiterungsschritt führen Sie nochmals durch, um zusätzlich ein Feld von Zeigern auf Texte zu sortieren:

```
char *sfeld[] = {
 "Caesar", "Doris", "Anton", "Peter", "Fritz"
};
```

Ergänzen Sie den Quelltext so, dass dieses Feld in derselben Weise wie die beiden anderen Felder ausgegeben, sortiert und durchsucht wird. Zum Vergleichen in den Vergleichsfunktionen können Sie die Funktion „strcmp()“ verwenden, deren Rückgabewerte vollständig kompatibel mit „qsort()“ und „bsearch()“ sind. – Werden auch die Namen von Ihrer neuen Programmversion richtig behandelt?

6.2 Nehmen wir an, Sie haben Feldelemente des folgenden Typs vorliegen:
struct feld_typ { double gehalt; int personal_nr; char *name_ptr; };
Überlegen Sie sich, wie eine Vergleichsfunktion aussehen könnte, die für eine aufsteigende Anordnung der Feldelemente sorgt, wobei das erste Kriterium durch den Namen, das zweite durch das Gehalt und das dritte durch die Personalnummer gegeben sein soll!

6.3 Ein noch effizienteres Vorgehen besteht darin, die (ausschließliche!) Vorgehensweise von Java auch in C umzusetzen und im Feld nur die Speicheradressen der Firmenmitarbeiter zu speichern, während diese als einzelne Objekte auf dem Heap angelegt werden. Welche kleinen, aber unerlässlichen Änderungen im Vergleich zur vorhergehenden Teilaufgabe müsste man dann in einer Vergleichsfunktion vorsehen?