1. Was ist eine Header – Datei? Was darf in einer Header-Datei nicht drinnen stehen?

Header – Dateien beinhalten Bekanntmachungen (Deklarationen) und Informationen über den strukturellen Aufbau von neu definierten Datentypen. Header – Dateien werden zur Übersetzung der Quelltextdateien benötigt.

Des Weiteren enthalten Headerdateien Definitionen von Präprozessorkonstanten, Typ- und Strukturdefinitionen und Funktionsdeklarationen. Sie haben das Suffix .h und heißen so, weil sie zur Übersetzungszeit im Kopf des Programms dazugeladen werden.

Headerdateien können entweder selbst geschrieben werden oder sie werden aus den verschiedenen mit dem Compiler mitgelieferten Bibliotheken bezogen.

2. Was sind Qualifizierer?

Werden dem Datentyp `int` Qualifizierer vorangestellt, so wird der Wertebereich verändert.

Datentyp	Bits	Wertebereich
short int	16	0…65535
long int	32	0…4294967295
signed short int	16	-32768…+32767
unsigned short int	16	0…+65535
Signed long int	32	-2147483648…+2147483647
unsigned long int	32	0…+4294967295

Diese Angaben gelten bei 32 Bit Architektur!

3. Was sind Datentypen?

Datentypen sind Definitionsmengen von Daten inklusive aller Operatoren und Funktionen, welche auf dieser Menge definiert sind.

4. Was ist der ASCII – Code?

Der ASCII – Code legt die Codierung der im englischen Sprachraum gebräuchlichen Buchstaben fest. Hierbei handelt es sich um einen 7-bit – Code. Der Wertebereich liegt zwischen 0 und 127. Er enthält im Bereich 1 bis 32 Steuercodes. Die nächsten 96 Codes beinhalten Buchstaben des englischen Sprachraums sowie diverse Sonderzeichen. Dieser Bereich ist genormt. Die Umlaute des deutschen Sprachraumes sowie zusätzliche Sonderzeichen sind im erweiterten ASCII-Code im Bereich von 128 bis 255 enthalten. Dieser Bereich ist nicht genormt.

Zeichen	ASCII-Code - Bereich
0…9	48…57
A…Z	65…90
a…z	97…122

5. Wie viele Gleitpunktdatentypen gibt es in C und wie unterscheiden sich diese?

Das einfach genaue Zahlenformat mit 32 Bit Länge ist als Datentyp `float` in C implementiert. Das doppelt genaue Zahlen Format von 64 Bit Länge ist durch den Datentyp `double` implementiert.

Datentyp	Bits	Genauigkeit	kleinste darstellbare Zahl
`float`	32	7 Stellen	$1{,}2 * 10^{-38}$
`double`	64	15Stellen	$2{,}3 * 10^{-308}$
`long double`	≥64	18 Stellen	$3{,}4 * 10^{-4932}$

Die Gleitpunktdatentypen unterscheiden sich durch die Größe des zur Verfügung stehenden Speicherplatzes.

5.1. `float`:

Bei diesem Datentyp stehen insgesamt 32 Bit zur Datenspeicherung zur Verfügung. Bit Nummer 31 beinhaltet das Vorzeichen (1… negativ, 0… positiv). Die Bits 23-30 beinhalten den Exponenten zur Basis 2. Die restlichen 23 Bit beinhalten die eigentliche Zahl (Mantisse) mit einer Genauigkeit von 7 signifikanten Stellen.

5.2. `double`:

Beim Datentyp `double` stehen zur Datenspeicherung 64 Bit zur Verfügung. Die Bits 0 bis 51 sind die Mantisse, die Bits 52 – 62 der Exponent und Bit 63 das Vorzeichen. Aufgrund der 52 Bit langen Mantisse erreicht dieser Datentyp die doppelte Genauigkeit des Typs float. Die Genauigkeit beträgt hier 15-16 Stellen. Die Genauigkeit kann mittels $\log 2^x$ berechnet werden. x … Länge der Mantisse in Bit. Beim Typ `double` 52 und bei `float` 23 Bit.

Es gibt des Weiteren auch noch den Datentyp long double, welcher je nach Rechnerarchitektur >= 64 Bit ist.

Es gibt unterschiedliche Datentypen, da nicht jede zu speichernde Zahl gleich große Genauigkeiten aufweisen muss. Höhere Genauigkeit erfordert höheren Bedarf an Speicherplatz.

6. Wie kann scanf() ohne den „&“ – Operator verwendet werden?

Der „&“ – Operator liefert die Adresse eines Objektes. Ein Zeiger ist bereits ein Objekt, welches die Adresse eines anderen Objektes (z.B.: einer Variable) enthält. Daher kann der Funktion scanf() direkt ein Zeiger als Parameter übergeben werden, da dieser bereits eine Adresse enthält.

7. Wie lautet die Deklaration von printf() und scanf()?

```
int=printf(char*,…);

int=scanf(char*, void*);
```

char* ist eine Zeichenkette, ja nach Anzahl der Platzhalter im char* muss man nach dem Komma die entsprechenden Variablen anführen. Die Anzahl der Parameter ist 1+n, n…Anzahl der Parameter.

8. Was sind Zeiger?

Ein Zeiger ist ein Objekt (! KEINE VARIABLE!), welches eine Adresse enthält. Zeiger können auf die Adresse von Objekten gesetzt werden. Dazu wird der Adressoperator & verwendet.

`ptr = &a` → Adresse von a.

Mit Hilfe des Dereferenzierungsoperators „*“ kann man dann über den Zeiger auf das Objekt zugreifen.

`wert = *ptr` → Wert, welcher an der in ptr gespeicherten Adresse steht.

Wird ein Zeiger auf „0“ (Null) gesetzt, so wird dieser dadurch ungültig gemacht. (`ptr = 0;`)

Soll eine Funktion mehr als einen Rückgabewert haben, so übergibt man der Funktion Zeiger als Parameter (Call by Reference).

Zeiger auf Funktionen:

```
Typ (*Funktionsname) (Parameterliste)
```

Ist das „*“ außerhalb der Klammer, dann ist der Ausdruck eine Funktionsdeklaration (Rückgabewert: Zeiger vom Typ long).

9. Zeiger – Feld – Dualität:

Der Name eines Feldes steht auch für die Adresse, an der sich das Feld im Speicher befindet. Der Feldname könnte somit direkt einem Zeiger zugewiesen werden, was allerdings zu schlechtem Programmierstil und Unübersichtlichkeit führt.

```
long feld[10];
long *ptr;

ptr = feld;          //gültig, aber unübersichtlich
ptr = &feld[0]; //korrekt
```

Hier wird der Zeiger auf das erste Feldelement gesetzt.

Von Zeiger-Feld-Dualität kann auch gesprochen werden, wenn ein Feld als Parameter an eine Funktion übergeben wird. Wie in Kapitel 13.5 erwähnt, wird ein Feld nicht als Kopie an eine Funktion übergeben. Es wird nur die Adresse des Feldes übergeben. Dies kann mit der sizeof– Anweisung überprüft werden.

	Zeigerschreibweise	Feldschreibweise
Adresse	`ptr + n`	`&feld[n]`
Objekt	`*(ptr + n)`	`feld[n]`

Softwaretechnisch gesehen sind Felder und Zeiger total verschiedene Konstrukte. Felder sind beim Übergeben an eine Funktion ein Zeiger auf das Feld (Call by Reference). Der Feldname kann wie ein Zeiger verwendet werden.

10. Was sind Zeiger auf Zeiger? Wann werden sie verwendet?

Zeiger können auf jeden beliebigen Datentyp zeigen, daher ist es auch möglich Zeiger auf Zeiger zu definieren. Zeiger auf Zeiger werden hauptsächlich verwendet um zwei Zeiger zu tauschen (Swap-Zeiger).

```
SwapZeiger(long **p1, long **p2)
{
      long *h;
      h=*p1;
      *p1=*p2;
      *p2=h;
}
```

Das Haupteinsatzgebiet von Zeigern auf Zeiger ist die dynamische Erzeugung von mehrdimensionalen Arrays wie beispielsweise Matrizenberechnungen.

11. Kann eine Funktion einer Funktion übergeben werden?

Eine Funktion kann mittels eines Zeigers auf eine Funktion an eine andere Funktion übergeben werden.

12. Wie definiert und verwendet man Zeiger auf Funktionen?

In C können Zeiger auf Funktionen vereinbart werden. Sie werden hauptsächlich bei der Verwendung von externen Bibliotheken, z.B. bei grafischen Benutzerschnittstellen eingesetzt.

Um die Adresse einer Funktion zu erhalten, wird wie bei Pointern auch hier der Adressoperator „&“ verwendet.

12.1. Definition:

Typ (*Funktionsname) (Parameterdatentyp);

Mittels einer „normalen“ Zuweisung wird dem Pointer die Adresse der Funktion zugewiesen.

```
long *fptr(long, long);
fptr=&function;
```

12.2. Verwendung:

Zeiger auf Funktionen können ähnlich zu „normalen“ Zeigern gesetzt und kopiert werden. Bei der Verwendung ist auf die Übereinstimmung der Datentypen zu achten.

Funktionsaufruf:

```
long x;
x=fptr(Variablenliste);
```

Zeiger auf Funktionen werden bei „call-back“ – Funktionen, die beim Eintreten von bestimmten Ereignissen vom Betriebssystem aufgerufen werden.

Eine weitere Anwendung ist bei Menüs. Hierbei werden die sind die einzelnen Einträge des Menüs in Strukturen beschrieben, die jeweils einen Zeiger auf die Menüfunktion speichern. Bei der Auswahl eines Menüpunktes wird dann diese Funktion aufgerufen.

13. Erklärung typedef:

Mittels `typedef` können neue Typnamen für Variablen, Funktionen und Strukturen selbst definiert werden.

Dies bringt eine Vereinfachung, wenn längere Datentypbezeichnungen öfter verwendet werden.

13.1. Abgeleitete Datentypen:

Mit der `typedef` – Vereinbarung können neue Typnamen erzeugt werden. Diese sind äquivalent mit der Bezeichnung wofür sie stehen.

```
typedef char *string;
```

Damit wurde ein neuer Typname string definiert. Dieser Typname ist identisch zu handhaben wie alle anderen Datentypen.

```
char *text1 = “Hallo“;
string text2 = „Welt“;
```

Beide Variablendefinitionen sind vollkommen identisch.

13.2. Strukturen:

Typedef kann auch bei Strukturen verwendet werden.

```
typedef struct Punkt_s
{
   struct Punkt_s *naechster;
   double x, y;
}Punkt_t;
```

Hier kann jetzt anstatt der Struktur eine neu Variable einfach über den Typnamen Punkt_t erstellt werden.

13.3. Funktionen:

Mit typedef können auch Typnamen für Zeiger auf Funktionen generiert werden.

```
typedef long Funktion_t(long, long);
```

Hier wird allgemein eine Funktion Funktion_t erzeugt, welche als Parameter zwei long – Variable erwartet. Eine Funktionsdefinition sieht wie folgt aus:

```
Funktion_t *function;
```

14. Unterschied zwischen Definition und Deklaration:

14.1. Deklaration:

14.1.1. Variablen:

Die Deklaration einer Variablen, ist deren Bekanntmachung an den Compiler. Es werden der Name und der Datentyp bekannt gegeben. **Wichtig: Es wird KEIN Speicher reserviert!**

Es können nur globale Variablen deklariert werden. Alle anderen Variablen müssen definiert werden.

14.1.2. Funktionen:

Die Deklaration einer Funktion funktioniert gleich wie bei Variablen. Es Handelt sich hierbei nur um eine Bekanntmachung des Funktionsnamen, der Parameterdatentypen und des Datentyps des Rückgabewertes.

```
long Funktionsname(long, long,…);
```

Die Namen der Parameter können dabei weggelassen werden. Nur die Anzahl der Übergabeparameter muss richtig sein.

14.2. Definition:

14.2.1. Variablen:

Die Definition einer Variablen ist deren vollständige Beschreibung. Das bedeutet, es muss der Datentyp und der Variablenname bekannt gegeben werden. Der Unterschied zur Deklaration liegt darin, dass bei der Definition der Speicherplatz reserviert wird. Es müssen alle Variablen definiert werden, lediglich globale Variablen können deklariert werden.

14.2.2. Funktionen:

Die Definition einer Funktion umfasst deren vollständige Beschreibung. Anders als bei der Deklaration müssen hier der Funktionsname, der Datentyp des Rückgabewertes, die Namen der Übergabeparameter, sowie der Funktionsrumpf angegeben werden.

Eine Funktion kann auch definiert werden, ohne vorher deklariert zu werden. Dafür muss die Funktionsdefinition vor dem Hauptprogramm (`main()`) und vor allem vor dem ersten Funktionsaufruf erfolgen. Also dort, wo sonst die Deklaration zu finden wäre.

```
long ergebnis(long wert1, long wert2)
{
    return wert1+wert2;
}
```

15. Wie funktioniert die Parameterübergabe in C? Welche Arten der Parameterübergabe gibt es?

Beim Funktionsaufruf werden die in runder Klammer () angegebenen Parameter an die Funktion übergeben. Die einzelnen Objekte in der Parameterliste werden durch Komma (,) getrennt. Die Objekte werden in Form einer Kopie an die Funktion übergeben (Call by Value).

```
long function(long var1, long var2);
ergebnis= function(wert1, wert2);
```

Die Variablen `var1` und `var2` werden mit den in den Variablen `wert1` und `wert2` gespeicherten Daten initialisiert. Innerhalb der Funktion kann mit diesen Variablen ganz normal gearbeitet werden.

Bei Feldern würde diese Art der Parameterübergabe sehr viel Speicherplatz in Anspruch nehmen. Daher wird bei Feldern die Adresse des ersten Feldelementes übergeben. Es wird also ein Zeiger übergeben (Call by Reference). Im Endeffekt wird mit dem Originalfeld gearbeitet.

```
long AusgabeFeld(long feld[], long len)
```

Der erste Parametererwartet ein Feld mit beliebiger Länge. Um der Funktion die Länge des Feldes mitzuteilen ist der zweite Parameter `len` erforderlich.

16. Was sind mehrdimensionale Felder und wie werden sie im Speicher repräsentiert?

Mehrdimensionale Felder, sind Felder bei denen jedes Element wieder ein Feld ist. Sie haben daher mehrere Indizes.

```
long feld[a][b][c]…;
```

```
long matrix[i][j];
```

Letztes Beispiel definiert eine Matrix mit `i` Zeilen und `j` Spalten. Um die einzelnen Elemente ansprechen zu können, werden auch wie bei eindimensionalen Feldern die Indizes für `i` $0 \ldots (i-1)$ und `j` $0 \ldots (j-1)$verwendet.

Im Speicher werden die Elemente hintereinander wie bei eindimensionalen Feldern angelegt.

matrix[i][j]		j		
i	0	E[0][0]	E[0][1]	E[0][2]
	1	E[1][0]	E[1][1]	E[1][2]
		0	1	2

i=0,j=3 FEHLER

im Speicher	i=0,j=0			i=1,j=0		
	E[0][0]	E[0][1]	E[0][2]	E[1][0]	E[1][1]	E[1][2]

Die Nummerierung der Elemente von $0 \ldots (n-1)$ ist wichtig. Würde das dritte Element der ersten Zeile mit matrix[0][3] angesprochen werden, würde auf den Speicherplatz des Elementes matrix[1][0] zugegriffen werden.

17. Wie kann ein Feld elegant an eine Funktion übergeben werden?

Ein Feld kann elegant als rekursive Struktur an eine Funktion übergeben werden. Dies hat vor allem den Vorteil, dass die Anzahl der Elemente unbegrenzt und variabel ist.

```
struct Eintrag_s
{
      struct Eintrag_s *nächster;
      long wert;
};
```

18. Sortierverfahren:

18.1. Minimum – Suche (Selection – Sort):

Minimum- und Maximum – Suche arbeiten mit demselben Verfahren.

Es ist ein einfaches aber sehr langsames Verfahren. Es wird in einer Folge von Zahlen das kleinste bzw. größte Element gesucht und gegen das Element an der Stelle 0 getauscht. Dann wird das nächst größere bzw. kleinere Element gesucht und gegen Das Element an der Stelle 1 getauscht. Dieses Verfahren wird solange ausgeführt, bis alle Elemente geordnet sind. Zum Umsortieren wird der Dreieckstausch verwendet.

```
long feld[4]={2,1,4,7};
long x;

x=feld[0];
feld[0]=feld[3];
feld[3]=x;
```

18.2. Bubble – Sort:

Bei diesem Verfahren werden immer zwei benachbarte Elemente miteinander verglichen. Ist das linke Element größer als das Rechte, so werden die zwei Elemente getauscht. Je größer eine Zahl ist, desto weiter nach rechts wandert sie in der Zahlenfolge. Da dieser Vorgang dem Aufsteigen eines Bläschens im Wasser gleicht, wird dieses Verfahren Bubble-Sort genannt.

Im schlechtesten Fall sind N-1 Sortierdurchläufe notwendig. Je öfter das Verfahren durchlaufen wurde, desto kürzer werden die nächsten Durchläufe.

18.3. Quicksort:

Quicksort ist ein sehr schnelles Sortierverfahren. Es erwartet die zu sortierenden Daten in einem Feld. Dieses Verfahren arbeitet rekursiv.

Quicksort bringt in jedem Schritt ein Element an seine korrekte Position.

FORTSETZEN!!

18.4. Heap – Sort:

Betrachtet man Heaps, so ist keine aufsteigende sequenzielle Sortierung der Elemente erkennbar. Dennoch ermöglichen Heaps das effiziente Sortieren von Feldern in ungefähr $N \log_2(N)$ Schritten. Der Sortieralgorithmus lautet:

Entferne die Wurzel aus dem Heap. Widerhohle den Vorgang solange, bis der Heap leer ist. Die Elemente liegen dann in sortierte Reihenfolge im Feld vor.

Entfernt man einen beliebigen Knoten aus dem Heap, so wird dieser zunächst mit dem letzten Element des Heaps getauscht. Nach dem Entfernen muss die Heapstruktur wieder repariert werden. Entfernt man alle Wurzeln, so wird das Element mit dem größten Schlüssel nach hinten gestellt. Diese Tatsache nutzt Heap – Sort aus.

19. Was ist ein String?

Literale von Zeichenketten sind Zeichenfolgen, welche in doppelte Anführungszeichen gesetzt sind.

„Hallo Welt!“

Zeichenketten in C sind „nullterminierte“ Folgen von Zeichen, welche in Feldern von Zeichen gespeichert werden.

„Nullterminiert“ bedeutet, dass eine Zeichenkette mit einer ‚\0‘ abgeschlossen wird. Das bedeutet weiter, dass eine Zeichenkette mit 5 Zeichen 6 Speicherplätze benötigt.

‘H‘ ‘A‘ ‘L‘ ‘L‘ ‘O‘ ‘\0‘

Das Null-Byte ist das Zeichen mit dem ASCII-Code 0. Es ist also die Zahl „0“ (\0), nicht zu verwechseln mit dem Zeichen „0“ (Null).
Dieses Null-Byte wird automatisch an den Text angehängt.

Dies hat den Vorteil, dass beliebig lange Texte abgespeichert werden können. Der Text endet immer mit dem Null-Byte.

Zum Feststellen der Länge einer Zeichenkette wird die Funktion `strlen()` verwendet.

19.1. Welche Funktionen sind auf Zeichenketten definiert?

Funktion	Beschreibung
`strcat(s1,s2)`	Anhängen von `s2` an `s1`
`strcmp(s1,s2)`	Lexikographischer Vergleich der Strings `s1` und `s2`
`strcpy(s1,s2)`	Kopiert `s2` nach `s1`
`strlen(s)`	Anzahl der Zeichen in String `s` (= `sizeof(s1)-1`)
`strchr(s, c)`	Sucht Character `c` in String `s`
`strstr(s, t)`	Sucht Zeichenkette `t` in `s` und liefert den Zeiger auf das erste Zeichen Vorkommnis

Es gibt noch weitere Funktionen auf Zeichenketten.

20. Was sind variante Strukturen?

Variante Strukturen haben große Ähnlichkeit mit regulären Strukturen, die mit `struct` definiert werden. Während jedoch die Attribute von regulären Strukturen im Speicher hintereinander liegen, belegen die Attribute von varianten Strukturen denselben Speicherbereich, sie liegen quasi übereinander.

Dies bedingt, dass beim Schreiben eines Attributs alle anderen überschrieben werden und somit ungültig werden. Die Werte der überschriebenen Attribute sind somit nicht mehr definiert.

Variante Strukturen werden dort verwendet, wenn mehrere Attribute zu einer Struktur zusammengefasst werden sollen, welche nicht gleichzeitig auftreten können. Zum Beispiel ein geometrisches Objekt kann entweder ein Kreis oder ein Rechteck sein, aber nie beides gleichzeitig.

Definiert werden variante Strukturen mit dem Schlüsselwort `union`. Die Definition erfolgt äquivalent zu Strukturen mit `struct`.

```
Union Unionname_u
{
      Typ Attribute;
};
```

21. Was sind rekursive Strukturen?

Rekursive Strukturen sind Strukturen, welche Attribute enthalten, die auf Strukturen desselben Typs verweisen.

```
struct Eintrag_s
{
      struct Eintrag_s *nächster;
      long wert;
};
```

Das Attribut `*nächster` ist ein Zeiger auf ein weiteres Objekt dieser Struktur.

Rekursive Strukturen können sehr sinnvoll im Bereich der dynamischen Speicherverwaltung eingesetzt werden.

22. Was sind rekursive Funktionen? Was passiert wenn die Abbruchbedingung erfüllt ist?

Funktionen, welche sich selbst aufrufen nennt man rekursiv.

Jeder Durchlauf der Funktion trägt zur Gesamtlösung bei.

22.1. Rekursive Algorithmen:

Viele Algorithmen lassen sich in gleichartige Teilprobleme zerlegen. Kennzeichen rekursiver Algorithmen ist, dass die Gesamtlösung durch lösen gleichartiger Teilprobleme erreicht wird.

Regeln für rekursive Algorithmen:

- Die Aufgabe wird in gleichartige Teilprobleme zerlegt. Die Teilprobleme werden Rekursiv gelöst
- Jede Rekursion trägt zur Gesamtlösung bei.
- Für mindestens eine Kombination der Funktionsparameter muss die Rekursion beendet werden. Dieser Fall muss auch tatsächlich auftreten, sonst terminiert der Algorithmus nicht und ist daher per Definition kein Algorithmus mehr.

In C werden rekursive Funktionen mittels eines Stapels realisiert, wobei jede Rekursionsebene ihr eigenes Variablenset am Stapel bekommt.

Durch jeden Selbstaufruf der Funktion wird der Stapel (Stack) größer, ist eine Abbruchbedingung erfüllt, also wenn eine Funktion beendet wird, wird die zugehörige Rekursionsebene zerstört und zur vorherigen Ebene zurückgekehrt.

Verwendet werden rekursive Algorithmen zum Beispiel bei Quicksort und beim Traversieren von Bäumen.

23. Was ist eine einfach verkettete Liste und wie ist diese aufgebaut?

Unter einer Liste versteht man eine sequenzielle Aneinanderreihung von Elementen, deren Reihenfolge explizit festgehalten wird. Eine Liste besteht aus Knoten, darin werden einerseits die zu speichernden Daten und andererseits die Verkettung zum nächsten Knoten. Der Anfang der Liste, der Listenkopf, wird separat behandelt.

In C wird ein sogenannter Knoten durch einen Verbunddatentyp realisiert, welcher die Daten und einen Verweis auf das nächste Listenelement beinhaltet. Der Kopf verweist auf das erste Listenelement. Ein Feld von Knoten muss nur noch initialisiert werden und die Verkettungen erstellt werden.

Der letzte Knoten muss speziell gekennzeichnet sein, damit das Ende der Liste erkannt werden kann. Hierfür wird der Verweis auf das nächste Listenelement im letzten Element auf `NULL` gesetzt.

24. Wie funktioniert das „Einfügen“ in Listen?

Um ein neues Element in eine bestehende Liste einzufügen, muss diese Liste an der gewünschten Stelle aufgebrochen werden und der neue Datensatz dort eingefügt werden. Ein neues Element am Listenanfang einzufügen ist sehr einfach, solange die Feldgrenze dabei nicht überschritten wird. Soll ein Element an einer anderen Stelle in der Liste eingefügt werden, so muss die Liste aufgebrochen werden. Der zukünftige Vorgänger des neuen Elements muss auf das neue Element verkettet werden und das neue auf das Nachfolgerelement vom Vorgänger.

25. Was ist eine Ringliste?

Eine Ringliste ist eine sehr selten verwendete Form von Listen. Dabei speichert der letzte Knoten einen Verweis an den Listenanfang, wodurch eine Rekursion ermöglicht wird.

26. Erklärung Datenstruktur Schlange:

Die Datenstruktur Schlange arbeitet nach dem Prinzip FIFO (first in first out). Mit den Methoden `put` und `get` können Elemente in die Schlange gestellt werden und wieder entnommen werden. Das als erstes eingefügte Element wird auch als erstes wieder entnommen. Schlangen sind vorzugsweise mit Listen implementiert.

27. Was ist ein Zirkularpuffer?

Ein Zirkularpuffer ist eine Implementierung der Schlange und arbeitet nach demselben Prinzip. Ein Unterschied zur Schlange ist jener, dass Zirkularpuffer in Feldern implementiert sind. Die Einfüge- und Entnahmeposition sind durch die Indizes im Feld gegeben. Nach dem Einfügen eines neuen Elements wird der Index der Einfügeposition erhöht. Ebenso wird bei der Entnahme verfahren. Nach dem Entnehmen eines Elements wird der Index der Entnahmeposition erhöht. Wird die Feldlänge überschritten, so wird der jeweilige Index auf den Feldanfang gesetzt (Herkunft des Namens: Zirkularpuffer).

Zirkularpuffer werden gerne zur asynchronen Kommunikation zweier Prozesse verwendet.

Problemsituationen: overflow, underflow

28. Wie funktioniert eine doppelt verkettete Liste (DLL):

DLL … double linked list

Im Gegensatz zu einfach verketteten Listen wird bei DLL auch die Verkettung zum Vorgänger gespeichert.

Der Vorteil dieser Art von Listen liegt darin, dass ein Rückwärtsbewegen in der Liste möglich ist, da der Vorgänger bekannt ist. Somit kann mit Referenzen auf Knoten sehr effektiv gearbeitet werden.

Ein Nachteil von doppelt verketteten Listen ist, dass beim Einfügen von neuen Elementen oder beim Umsortieren zusätzlich zum Nachfolger auch noch der Vorgänger richtig gesetzt werden muss. Dies ist eine zusätzliche Fehlerquelle.

29. Wie funktioniert das Einfügen in eine doppelt verkettete Liste? Wie implementiert man diese Funktion?

Ein Knoten in einer doppelt verketteten Liste wird als Verbunddatentyp realisiert. Ein Knoten speichert die Daten, einen Verweis auf den Vorgänger und einen Verweis auf den Nachfolger. Es handelt sich hierbei also um eine rekursive Struktur.

Wird ein neues Element in die Liste eingefügt, so müssen die Verweise von 3 Knoten verändert werden.

Ein Beispiel:

Konten: x, y

```
x->Nachfolger =y
y->Vorgänger =x
```

Neuer Knoten: z wird zwischen x und y eingefügt

```
z-> Vorgänger =x
z-> Nachfolger = y
x->Nachfolger = z
y->Vorgänger = z
```

30. Erklärung Bäume:

Bäume werden beim Programmieren häufig verwendet. Es gibt unterschiedlichste Arten davon, wie etwa Bäume zur Speicherung von Daten, Bäume zum Sortieren, Bäume für Syntaxanalyseprobleme, geometrische und mathematische Probleme oder für die Datenkomprimierung.

Listen stellen eine eindimensionale Datenstruktur, im Gegensatz dazu können Bäume als mehrdimensionale „Erweiterungen“ gesehen werden. Ein Knoten hat bei einem Baum nicht nur einen Nachfolger, sondern kann auch mehrere haben. Die Verknüpfung zweier Knoten wird Kante genannt, Knoten ohne Nachfolger heißen Endknoten. Als Wurzel eines Baumes wird jener Knoten genannt, der keine Vorgänger hat.

Bäumen könne auch rekursiv definiert werden. Dabei besteht ein Baum aus mehreren Subbäumen, deren Wurzeln durch einen gemeinsamen Knoten verkettet sind.

Es gibt verschiedene Knoten:

- Wurzel: dieser Knoten hat keinen Vorgänger
- Parent: Ist der Vorgängerknoten eines weiteren Knoten
- Endknoten, äußerer Knoten: hat keine Nachfolger
- Innerer Knoten

Ein allgemeiner Baum mit n Knoten hat $n - 1$ Kanten.

31. Was ist ein binärer Baum?

Ein binärer Baum hat pro Knoten genau 2 Nachfolger. Die Knoten eines solchen Baumes beinhalten die Daten und die Verkettungen zu den Nachfolgern.

Ein binärer Baum mit N_I inneren Knoten hat maximal $N_I + 1$ äußere Knoten.

Bei einem vollständigen Baum sind die Blätter von links nach rechts angefügt.

32. Binäre Suchbäume:

Ein binärer Suchbaum ist ein binärer Baum, bei dem jeder Knoten eine besondere Bedingung erfüllt. Alle Knoten im linken Unterbaum eines Knotens haben kleinere Schlüssel. Alle Knoten im rechten Unterbaum eines Knoten haben größere Schlüssel.

Für gleiche Schlüssel wurde die Vereinbarung getroffen, dass diese am linken Unterbaum angehängt werden.

Es ist nicht möglich einen Baum in einen vollständigen binären Baum umzuwandeln und gleichzeitig die Bedingung für binäre Suchbäume zu erfüllen. Da in einem Baum mehrere Knoten mit dem gleichen Schlüssel vorkommen können, ist dies nicht möglich.

33. Was versteht man unter Traversierung?

Die Traversierung eines Baumes beschreibt, wie dieser durchlaufen – traversiert- werden kann. Dieser Aufgabe ist besonders schön rekursiv zu lösen.

Es gibt verschiedene Methoden, die Traversierung vorzunehmen:

- Level – Order
- Preorder- Traversierung: B-A-C
- Inorder- Traversierung: A-B-C
- Postorder – Traversierung: A-C-B

Die Methoden Preorder, Inorder und Postorder sind sehr ähnlich. Unterschiedlich ist nur die Reihenfolge, in der die Knoten durchlaufen werden.

Die Methode Level – Order kommt bei Heap-Strukturen zum Einsatz. Dabei werden die Element zeilenweise durchlaufen von links nach rechts.

34. Suchen eines Elements in einem binären Suchbaum:

Soll in einem Baum nach einem bestimmten Element gesucht werden, so muss dieser traversiert werden. Die günstigste Reihenfolge der Abfrage:

- Ist der gesuchte Schlüssel kleiner als jener des aktuellen Knoten, wird nach links verzweigt.
- Ist der gesuchte Schlüssel größer als jener des aktuellen Knoten, wird nach rechts verzweigt.
- Ist der gesuchte Schlüssel gleich dem aktuellen, so ist der Knoten gefunden.

Bei gleichverteilten Schlüsseln, werden bei erfolgreicher Suche in einem vollständigen Baum etwa $\log_2(N) - 1$ Vergleiche durchgeführt.

Eine erfolglose Suche wird erst festgestellt, wenn ein Endknoten erreicht ist. Es sind zirka $\log_2 N$ Vergleiche notwendig.

35. Einfügen eines neuen Elements in einen binären Suchbaum:

Um einen Knoten in einen binären Suchbaum einzufügen, wird das Element „erfolglos“ gesucht, das bedeutet, dass die Suche erst abbricht, wenn ein Endknoten erreicht ist.

Das erste Element in einen Suchbaum einzufügen ist trivial. Beim nächsten Element muss ermittelt werden, ob der Schlüssel des neuen Elements größer oder kleiner ist als der des bereits vorhandenen Knoten.

Ist der Schlüssel kleiner, so wird der Knoten Links angehängt, ist er größer wird er rechts angehängt.

36. Löschen eines Elements in einem binären Suchbaum:

Ein Element aus einem binären Suchbaum zu löschen ist etwas aufwändiger als nur zu sortieren, einen Knoten einzufügen, traversieren oder zu suchen. Die Schlüssel sind meist untrennbar mit der Struktur von Bäumen verbunden.

Beim Löschen eines Knotens kommt es zu einer Beschädigung der Baumstruktur. Diese muss anschließend wieder repariert werden.

Wird ein Knoten gelöscht, so muss ein anderer gefunden werden, der den Platz des gelöschten Knotens einnimmt, um die Baumstruktur zu erhalten.

Dieser Knoten muss so gewählt werden, dass die Bedingungen für einen binären Suchbaum erhalten bleiben. Somit ist der gesuchte Knoten jener mit dem nächst kleinerem oder größerem Schlüssel. Dies garantiert, dass kein anderer Schlüssel dazwischen liegt.

36.1. Algorithmus zum Löschen eines Knotens:

Buch Seite 285.

37. Was ist ein Heap und für welche Operationen ist er gut geeignet?

Eine Heap – Struktur ist eine sogenannte Prioritätswarteschlange, die es ermöglicht den größten Eintrag zu finden, zu löschen und neue Elemente einzufügen.

Es ist eine besondere Art eines binären Baumes, der als Feld gespeichert wird. Jeder Knoten muss dabei die sogenannte Heap – Bedingung erfüllen:

37.1. Heap – Bedingung:

Der Schlüssel jedes Knoten eines Heaps ist größer als sein Nachfolger. Falls auch gleiche Schlüssel auftreten können muss der Schlüssel jedes Knotens größer oder gleich dem seiner Nachfolger sein.

Ein Heap ist ein vollständiger binärer Baum. Im Unterschied dazu lautet die Bedingung für binäre Suchbäume, dass der Schlüssel des linken Nachfolgers kleiner und der Schlüssel des rechten Nachfolgers größer ist. Daher sind die Elemente in einem Baum vollständig anders verteilt.

Jedem Knoten wird ein fortlaufender Index zeilenweise zugewiesen. Diese Indizes entsprechen den Feldindizes. Der Index „0“ existiert nicht.

Die Implementierung im Feld bringt mehrere Vorteile:

- Heaps speichern keine expliziten Verknüpfungen, wodurch zum Feststellen des Vorgängers oder Nachfolgers nur einfache arithmetische Operationen durchgeführt werden müssen.
- Die Nachfolger eines Elements mit dem Index i haben die Indizes $2i$ und $2i + 1$. Umgekehrt sind die Vorgänger zu ermitteln. Der Index eines Vorgängers ergibt sich zu: $i/2$. Jede Ebene hat 1 Element mehr als alle höheren Ebenen. Jede vollständige Ebene hat doppelt so viele Elemente als die vorhergehende.

- Der Nachteil der Feldimplementierung ist die starre Struktur, da explizite Verknüpfungen fehlen. Heaps bieten jedoch so viel Flexibilität, um effiziente Algorithmen für Prioritätsschlangen implementieren zu können.

37.2. Methoden für Heaps:

- Einfügen eines neuen Elements: Um ein neues Element in den Heap einzufügen, wird es an die nächste freie Position gestellt, also an die Stelle nach dem letzten Eintrag. Dadurch wird höchst wahrscheinlich die Heapbedingung verletzt. Der Heap muss neu sortiert werden.

- Traversieren über einen Heap ist trivial, da durch einfache arithmetische Berechnungen der Nachfolger und auch der Vorgänger berechnet werden kann. Hauptsächlich werden die Level – Order- oder Preorder – Methode angewendet.

- Entfernen eines Elements: Es wird ein Trick angewandt. Das zu löschende Element wird mit dem letzten Element getauscht und dann dieses gelöscht. Dadurch wird ein aufreißen der Heapstruktur verhindert und der Heap um ein Element verkürzt. Das bedeutet die Feldgröße wird um 1 verringert.

Nach dem Löschen oder Einfügen eines neuen Elementes in den Heap, muss dieser wieder repariert werden. Zum Reparieren des Heaps gibt es die Funktionen UpHeap und DownHeap.

37.2.1. UpHeap:

Mit dieser Methode wird ein Heap repariert. Es wird am Ende des Heaps (ganz hinten, am unteren Ende) begonnen. Die Funktion arbeitet von Unten nach Oben, daher auch der Name. Es wird verglichen, ob der Schlüssel des Vorgängers kleiner ist. Ist dies der Fall werden die Elemente getauscht. Das Element steigt somit im Heap hinauf. Diese Aktion wird solange wiederholt, bis der Heap durchsortiert und die Heapbedingung erfüllt ist. Diese Methode wird beim Einfügen von neuen Elementen in einen Heap verwendet.

37.2.2. DownHeap:

Diese Methode arbeitet top down, also von Oben nach Unten. Dabei wird ein Element mit seinem Nachfolger verglichen. Ist der Schlüssel von einem der beiden Nachfolger größer, so wird das Element mit seinem Nachfolger getauscht. Sind beide Nachfolger größer, so wird das Element mit dem Nachfolger, der den größten Schlüssel hat getauscht. Haben beide Nachfolger gleiche oder größere Schlüssel, so wird je nach Implementierung mit dem linken oder rechten Nachfolger getauscht. Diese Methode setzt einen korrekten Heap voraus, bis auf ein Element, welches richtig eingeordnet werden soll. Sie wird beim Löschen oder Ersetzten eines Eintrages verwendet.

38. Erklärung Hash:

Ein Hash eignet sich besonders gut zum effizienten Speichern und Suchen von Datensätzen. Das Verfahren beruht auf dem Prinzip des Feldes, in welchem die Datensätze geeignet gespeichert und mittels eines generierten Index gesucht werden.

Der Schlüssel wird dabei in eine Tabellenadresse umgewandelt. Dadurch kann bei einer Suche sofort auf diese Tabellenadresse, ohne den vollständigen Datensatz zu durchlaufen, zugegriffen werden.

Ziele dieses Verfahrens sind eine effizientere Nutzung der verfügbaren Speicherkapazität und ein schnellerer Zugriff.

Der Index wird von der Hash-Funktion generiert.

38.1. Hash-Funktion:

Die Hash-Funktion hat die Aufgabeumfangreiche Daten(z.B.: Texte) in kurze, möglichste eindeutige Identifikationen (den Hash-Wert des Textes) umzuwandeln. Dazu berechnet die Funktion über einen Suchschlüssel den Index für die Hash-Tabelle.

Es empfiehlt sich, die maximale Größe des Hashes (Anzahl der möglichen Schlüssel) mitanzugeben, da die Funktion für verschiedene Hashes verwendbar sein soll.

38.2. Hashtabelle:

In der Hashtabelle befinden sich die eindeutigen Adressen. Eine Suchanfrage wird zunächst mit Hilfe der Hash-Funktion in eine Tabellenadresse umgewandelt. Mit dieser eindeutigen Adresse wird der Datensatz in der Hash-Tabelle gesucht. In der Praxis wird die Tabelle in einem Feld implementiert. Die Hashtabelle wird dynamisch erzeugt, damit sie vielseitig verwendbar ist. Die Länge des Feldes muss für die Hash – Funktion gespeichert werden. Aus praktischen Gründen sollte die Feldlänge eine Primzahl sein. Eine spätere Änderung der Feldlänge hat nicht triviale Folgen. → neue Indizes werden anders berechnet→ alte Einträge sind nicht mehr auffindbar.

38.3. Kollisionsbeseitigung:

Da Hash-Funktionen im Allgemeinen nicht eindeutig sind, können unterschiedliche Schlüssel zum selben Hash-Wert, also zum selben Feld in der Tabelle führen. Dieses Ergebnis wird als Kollision bezeichnet. In diesem Fall muss die Tabelle mehrere Werte an der selben Stelle aufnehmen. Eine Möglichkeit zur Kollisionsbeseitigung ist es, Datensätze mit gleichem Index in Listen zusammenzufassen. (**getrennte Verkettung**)

39. Für welche Operationen ist ein Hash besonders gut geeignet? Wie groß sollte die Hashlänge gewählt werden?

Hashes eignen sich besonders gut zum Suchen von Daten und somit zur Datenerhaltung. Die Qualität von Hashes hängt stark von der Hash-Funktion, der Feldlänge und den Schlüsseln der zu speichernden Daten ab. Sie speichern keine Reihenfolgen, Hierarchien oder sonstige Abhängigkeiten von Daten. Ist die Datenmenge M bekannt, so sollte man die Hashlänge stets größer wählen.

$N \geq 3M, N$ als Primzahl von Vorteil.

40. Wie löscht man Elemente aus einem Hash?

In der Liste werden die Zeiger auf den Nachfolger des zu löschenden Elements gesetzt und anschließend wird der Eintrag als ungültig markiert.

41. Was wird aus dem Hash, wenn man bei getrennter Verknüpfung Hashlänge 1 wählt.

Verkettete Liste

42. Erklärung dynamische Speicherverwaltung in C

Die Speicherverwaltung ist der untersten Ebene im Aufbau eines Programmes zuzuordnen. Dabei wird der vergebene Speicher in geeigneten Datenstrukturen erfasst und verwaltet. Aus einem Programm wird Speicher einer bestimmten Größe angefordert, im Programm verwendet und nach dem Gebrauch wieder retourniert. Das Betriebssystem rundet die angeforderte Speichergröße auf durch 8 teilbare Längen auf, um Zerstückeln des Speichers in viele kleine Blöcke zu verhindern.

43. Welche Funktionen werden in C angeboten und wie funktionieren sie?

43.1. malloc():

Mit der Funktion `malloc()` ist es möglich Speicher anzufordern. Als einziger Parameter muss die benötigte Speichergröße in Byte angegeben werden. Um die benötigte Speichergröße herauszufinden wird die Funktion `sizeof()` verwendet. Wird dieser Funktion ein Datentyp übergeben, so liefert sie dessen Speicherplatzbedarf in Byte.

`sizeof(long)` liefert 4Byte (32Bit), ebenso kann mit jedem anderen Datentyp vorgegangen werden, auch mit selbst definierten Datentypen und Strukturen.

Werden zum Beispiel 4 Speicherplätze vom Datentyp long benötigt, so muss der Funktion malloc() folgendes übergeben werden:

```
4*sizeof(long)
ptr= malloc(4*sizeof(long));
```

Die Funktion `malloc()` liefert bei erfolgreicher Speicherplatzreservierung einen Pointer auf die Adresse des ersten Speicherplatzes. Werden mehrere Speicherplätze desselben Typs reserviert, so wird auch immer nur der Zeiger auf den ersten geliefert. Die weiteren reservierten Plätze haben eine um 1 höhere Adresse als der vorhergehende. Konnte der angeforderte Speicherplatz nicht reserviert werden, so liefert die Funktion „0“. Der Pointer ist also ungültig.

43.2. calloc():

Die Funktion `calloc()` funktioniert genau gleich wie die Funktion `malloc()`. Der einzige Unterschied ist der, dass sie anstatt einem Argument zwei erwartet. Sie erwartet die Argumente:

- Anzahl der Speicherplätze
- Größe eines Speicherplatzes.

```
ptr=calloc(4, sizeof(long));
```

Auch diese Funktion liefert bei erfolgreicher Reservierung den Zeiger auf den ersten Speicherplatz. Bei nicht erfolgreicher Reservierung liefert sie „0“.

43.3. realloc():

Mit dieser Funktion können Speicherplätze nicht nur erzeugt werden, sondern auch in ihrer Größe verändert werden. Sie erwartet daher 2 Argumente:

- den alten Speicherblock
- die neue Größe des gesamten Speicherblockes.

```
ptr=calloc(4, sizeof(long));
ptrneu=realloc(ptr, 8*sizeof(long));
```

Schlägt der Aufruf fehl, so liefert auch diese Funktion „0“. Des Weiteren ist nicht gewährleistet, dass der Speicherplatz nach dem Verändern der Größe noch an derselben Adresse wie vor der Reallozierung steht. Die Verwendung dieser Funktion ist daher oft Problematisch. Zeigt ein Zeiger auf einen Speicher der von `realloc()` verändert wird, kann es zu Fehlern kommen, da dieser Zeiger bei Verschiebung des Speichers ungültig wird.

43.4. free():

Wird ein dynamischer Speicher nicht mehr benötigt oder das Programm beendet, so sollte dieser wieder freigegeben werden. Hierfür gibt es in C die Funktion `free()`. Auf einen Speicher, der bereits freigegeben wurde, darf nicht mehr zugegriffen werden. Außerdem darf ein bereits freigegebener Speicher kein zweites Mal freigegeben werden, da es dadurch zu Programmabstürzten kommen kann.

Nach dem Freigeben eines Speichers sollte der Zeiger auf den Speicher ungültig gemacht werden, indem er auf „0“ gesetzt wird.

```
if(ptr)
{
    free(ptr);
    ptr = 0;
}
```

Wird die Funktion `free()` auf einen ungültigen Zeiger angewandt, so ergibt das keinen Fehler sondern wird ignoriert.

Um mit diesen allozierten Speichern arbeiten zu können sollte eine Abfrage gemacht werden, ob die Speicherallozierung erfolgreich war.

z.B.:

```
if(ptr=calloc(4, sizeof(long)))
```

```
{
	//Verwendung des Speichers
}
```

44. Was sind Memory - Leaks?

Memory – Leaks entstehen, wenn ein Programm dynamisch Speicher alloziert und diese Speicherressourcen nicht mehr an das System zurückgibt. Durch diese Memory – Leaks wird der Speicher immer geringer und es kann zu langsameren Reaktionen von Programmen und vom Betriebssystem kommen.

Memory – Leaks können durch folgende Fehler auftreten:

- Verlust eines Zeigers auf einen Speicher, weil dieser Zeiger überschrieben wurde
- Verlust des Speichers durch Verlust des Zeigers
- Verlust des Zeigers durch Rücksprung
- Verlust des Zeigers bei der Rückgabe von Funktionen

45. Welche Fehlerarten gibt es (Numerik)?

45.1. Modellfehler:

Bei jeder Modellbildung wird eine Abstraktion der Realität durchgeführt, dabei werden bekannte und unbekannte Größen vernachlässigt und Vereinfachungen der tatsächlichen Verhältnisse vorgenommen. Der Fehler, der dadurch entsteht, wird Modellfehler genannt. Der Beitrag dieses Fehlers zum Gesamtfehler kann aufgrund der Vernachlässigung unbekannter Größen nicht abgeschätzt werden. Es kann lediglich der Fehler der vernachlässigten bekannten Größen abgeschätzt werden.

45.2. Datenfehler:

Datenfehler entstehen durch Ungenauigkeiten bei der Erfassung von Daten, da diese nur bis zu einer gewissen Genauigkeit erfasst werden können. Der dadurch entstehende Fehler lässt sich meist abschätzten. Dies ist jedoch von der Messmethode abhängig (z.B.: bei Längen, Geschwindigkeiten,…). Die Auswirkung dieser Fehler können mittels Konditionsuntersuchungen abgeschätzt werden.

45.3. Verfahrensfehler:

Mathematische Probleme werden oftmals mit iterativen Näherungsverfahren durchgeführt, welche sich an die Lösung herantasten. Ist die Lösung hinreichend genau, so wird das Verfahren abgebrochen und es entsteht der Abbruchfehler.

Diskretisierungsfehler entstehen bei einem Übergang von einem kontinuierlichen auf ein diskretes System. Ein Intergral wird beispielsweise numerisch durch Summation durchgeführt. Hierbei entsteht gegenüber der tatsächlichen Lösung des Integrals ein Fehler.

Verfahrensfehler lassen sich durch erhöhten Aufwand oft beliebig verkleinern, da aber jedes Verfahren nach einer bestimmten Zeit abgebrochen werden muss entstehen immer Abbruchfehler. Dieser ist im günstigsten Fall die Darstellungsgenauigkeit des internen Datentyps.

45.4. Rundungsfehler:

Punktzahlen können auf einem Computer bis zu einer bestimmten Genauigkeit dargestellt werden. Diese Genauigkeit ist vom Datentyp abhängig. Bei jeder Rechenoperation wird das Ergebnis auf einen darstellbaren Wert abgebildet – gerundet. Der Unterschied zwischen dem exakten Wert und dem gerundetem Wert ist der Rundungsfehler oder Rechenfehler.

46. Wie kann der relative Fehler berechnet werden?

Der absolute Fehler ergibt sich zu:

$$absoluter\ Fehler = Näherungswert - exakter\ Wert$$

Der relative Fehler einer numerischen Berechnung ergibt sich aus:

$$relativer\ Fehler = \frac{absoluter\ Fehler}{Bezugsgröße}$$

Die Bezugsgröße ist meist der exakte Wert. Man erhält folgende Gleichung:

$$relativer\ Fehler = \frac{absoluter\ Fehler}{exakter\ Wert} = \frac{Näherungswert - exakter\ Wert}{exakter\ Wert}$$

$$= \frac{Näherungswert}{exakter\ Wert} - 1$$

47. Was sagt die Konditionszahl aus? Was ist die ideale Konditionszahl?

Die Konditionszahl K gibt Auskunft über die Empfindlichkeit eines Systems auf Änderungen der Eingangsdaten.

$$K \geq \frac{Fehler\ der\ Lösung}{Fehler\ der\ Eingangsdaten}$$

Ein System mit kleiner Konditionszahl ist Änderungen gegenüber unempfindlicher als ein System mit großer Konditionszahl.

Die ideale Konditionszahl ist:

$$K_{ideal} = 1$$

48. Was ist die kleinste mögliche Zahl?

Die Header-Datei `limits.h` gibt Auskunft über Grenzen ganzzahliger Zahlentypen. Diese Grenzen sind in Konstanten gespeichert:

- LONG_MAX: größte positive Zahl
- LONG_MIN: kleinste negative Zahl

Die Header-Datei `float.h` gibt die Schranken für die Datentypen float, double und long double an:

- DBL_MAX: größte positive Zahl
- DBL_MIN: kleinste(normalisierte) positive Zahl >0
- DBL_EPSILON: Differenz zwischen 1 und der kleinsten Zahl größer als 1

Die kleinste denormalisierte Zahl ergibt sich mit:

$$\frac{DBL\_MIN}{2^{52}}$$

Normalisiert heißt, dass eine 1 direkt hinter dem Komma steht.

49. Was passiert bei der Summation von 2 Doubles? Welche Fehler können dabei auftreten?

Bei der Summation von zwei Punktzahlen werden die beiden Mantissen addiert. Hierfür müssen allerdings die Exponenten gleich sein. Ist die Zahl um den Faktor 2^{52} größer, so wird 0 addiert. In der Computerarithmetik gilt weder das Assoziativgesetz noch das Distributivgesetz. Der Exponent der kleineren Zahl wird auf den Exponenten der größeren gebracht. Dabei wird die Mantisse nach rechts verschoben, wobei gesetzte Bits rechts herausfallen können → Datenverlust.

Ein häufiger Fehler bei der Summation von Punktzahlen ist der Auslöschungseffekt. Dieser tritt auf, wenn zwei annähernd gleich große Zahlenwerte addiert oder subtrahiert werden. Unterscheiden sich zwei Werte fast ausschließlich durch ihre nicht signifikanten Stellen, so fallen bei einer Subtraktion hauptsächlich die signifikanten Stellen weg. Es bleiben nur die nicht aussagekräftigen Stellen übrig. Diese Störungen an den hinteren Stellen der Werte werden zu Störungen an vorderen Stellen des Ergebnisses. Daher ist der relative Fehler sehr hoch.

Auslöschung ist die häufigste Ursache für schlechte Kondition und die numerische Instabilität von Algorithmen.

50. Welche Fehlerarten gibt es? (Fehlerbehandlung)

50.1. Modellfehler:

Modellfehler können nur behandelt werden, wenn sie auch erkannt werden können. Sie entstehen oftmals auch durch bewusste Vernachlässigungen.

50.2. Datenfehler:

Datenfehler sind die häufigste Ursache für eine Fehlerbehandlung. Dazu zählen beispielsweise Messungenauigkeiten, syntaktisch Falsche oder unvollständige Eingabedaten oder fehlerhafte Dateiformate.

50.3. Verletzte Vorbedingung:

Diese Fehlerart zählt eigentlich zu den Datenfehlern. Die Korrektheit der Daten bezieht sich auf einen bestimmten Algorithmus. Die Daten können anhand einer definierten Bedingung (Vorbedingung) auf Korrektheit überprüft werden.

50.4. Anwenderfehler:

Anwenderfehler entstehen durch fehlerhafte Bedienung sowie fehlende oder inkorrekte Eingaben. Sie gehören auch zu den Datenfehlern und erfordern bei der Behandlung oft besondere Richtlinien für die Benutzerführung.

50.5. Fehlerhafter Programmaufruf:

Dies ist ein Fehler, welcher beim Programmaufruf passiert und dadurch eigentlich ein Anwenderfehler ist. Ein fehlerhafter Programmaufruf kann zu einem sofortigen Abbruch des Programms führen.

50.6. Numerische Fehler:

Numerische Methoden liefern nur mit einer Fehlerabschätzung eine sinnvolle Auswertung. Es ist notwendig bei der Implementierung auf die numerische Stabilität des Verfahrens zu achten.

50.7. Speicherprobleme:

Speichermangel ist ein leicht zu erkennendes Problem, welches oftmals nur sehr schwer korrigiert werden kann. Es kann dazu führen das der Algorithmus nicht mehr ausgeführt werden kann.

Abhilfe:

- Neu starten des Programms
- Das Programm wartet bis der Speicher wieder verfügbar ist
- Nicht mehr benötigter Speicher wird freigegeben, wenn möglich.

50.8. Fehlerhafter Speicherzugriff:

Dies ist ein sehr kritischer Fehler, der entweder zum sofortigen Programmabbruch oder zum lesen falscher Daten und überschreiben von Daten, die nicht zum Programm gehören, führt.

Abhilfe: Signalbehandlung

50.9. Fehlerhafte Module:

Bei der Softwareentwicklung wird sehr oft Fremd-Software verwendet. Dazu gehören auch Bibliotheken und Module, aber auch das Betriebssystem. Diese Fremd – Software kann auch Fehler enthalten.

50.10. Dead – Lock:

Bei parallelen Prozessen kann es vorkommen, dass ein Prozess auf Daten des anderen wartet und es daher zu einem Stoppen des Programmes kommt. Dieser Fehler kann während der Programmlaufzeit kaum behoben werden, da die Prozesse gestoppt sind. Am einfachsten ist es eine gewisse Zeit zu warten, dass der Programmablauf von selbst wieder beginnt.

51. Welche Möglichkeiten bietet C zur Fehlerbehandlung und wie funktionieren diese?

Verschiedene Programmiersprachen bieten verschiedene Möglichkeiten zur Fehlerbehandlung. In C++ gibt es beispielsweise „Ausnahmen“ (exceptions). In C haben sich die sogenannten Fehlercodes durchgesetzt. Es ist empfehlenswert Fehlercodes mit Aufzählungstypen (`enum`) zu realisieren.

Der Präfix `ERR` deutet darauf hin, dass es sich um einen Fehler handelt.

Des Weiteren bietet die Standardbibliothek in C einige Möglichkeiten zur Fehlererkennung und Fehlerbehandlung an.

Die Fehlervariable `errno` ist vom Typ `int` und ist in der Standard-Headerdatei `errno.h` deklariert. Sie gibt viele Fehler von Betriebssystemen und C-Standardbibliotheksfunktionen durch einen Fehlercode bekannt. Beim Programmstart wird diese Variable auf 0 gesetzt und nur im Fehlerfall von Betriebssystem- und Bibliotheksfunktionen auf den Fehlercode gesetzt.

`errno` kann dann wie jede andere Variable abgefragt und ausgewertet werden. Beispielsweise kann `errno` auf `ENOET` gesetzt werden. Dies ist eine Präprozessorkonstante.

52. Was macht die Funktion exit() und wie wird sie verwendet?

Diese Funktion ist Hilfreich für Testphase in der Programmentwicklung. Wird die Funktion `exit()` aufgerufen, wird das Programm sofort beendet, egal wo sich das Programm gerade befindet. Die Funktion ist der Headerdatei `stdlib.h` deklariert und erwartet als einzigen Parameter einen Fehlerwert. Der Funktionsaufruf erfolgt durch:

```
exit(-1);
```

53. Was macht die Funktion atexit() und wie wird sie verwendet?

Mit der Funktion `atexit()` können mehrere Funktionen registriert werden, die beim regulären Programmende aufgerufen werden. Ein Programmende ist regulär, wenn aus der Funktion `main()` zurückgesprungen wird oder wenn `exit(-1)` aufgerufen wird. Diese Funktion ist genau auch wie `exit()` in der Headerdatei `stdlib.h` deklariert. Sind mehrere Funktion registriert, so werden diese in umgekehrter Reihenfolge ihrer Registrierung aufgerufen. Als Parameter wird ein Zeiger auf eine Funktion erwartet.

```
atexit(&endfunction);
exit(-1); //Die Funktion endfunction wird hier aufgerufen.
```

54. Erklärung der Funktion abort():

Die Funktion `abort()` beendet das Programm nicht wie `exit()` regulär, sondern durch einen Abbruch – also ein außergewöhnliches Programmende. Dabei wird das Signal `SIGABRT` an den Prozessor gesendet, wodurch die Signalbehandlung für das Signal ausgelöst wird. Diese voreingestellte Signalbehandlung bricht das Programm ab. `abort()` ist in der Headerdatei `stdlib.h` deklariert und hat keine Argumente.

55. Was macht die Funktion assert() und wie wird sie verwendet?

Assert ist genaugenommen ein Präprozessormakro und ist in der Headerdatei assert.h deklariert. Assert erwartet einen Parameter, ist dieser Parameter gleich 0, so gibt assert() eine Fehlermeldung aus und beendet das Programm durch den Aufruf der Funktion abort().

```
datei =fopen(„bsp.txt“);
assert(datei);
```

Diese Funktion wird gerne in der Entwicklungsphase eines Programms zur Kontrolle des Zustandes einer Variablen verwendet. Der übergebene Ausdruck wird auf 0 abgefragt!

56. Was sind Signale und wo kommen sie her?

Signale sind nicht vorhersehbare Ereignisse, welche zu nicht vorhersehbaren Zeitpunkten auftreten können. Es sind also asynchrone Ereignisse. Ein Signal kann vom Betriebssystem, von externen Interrupts, von anderen Programmen oder Eingabegeräten initiiert werden und an einen Prozess gesendet werden. In Programmen können diese Signale abgefangen und behandelt werden.

57. Wie Funktioniert in C die Signalbehandlung?

In C enthält die Headerdatei `signal.h` eine vollständige Liste aller vorhandenen Signale. Empfängt ein Prozess ein Signal, wird der Programmablauf unterbrochen und eine Funktion zur Signalbehandlung aufgerufen. Diese Funktionen nennt man `singalhandler`.

Die Funktion `signal()` ist in der Headerdatei `signal.h` deklariert und erwartet zwei Parameter. Der erste Parameter ist die Signalnummer und der zweite Parameter ein Zeiger auf die Funktion zur Signalbehandlung. Diese Funktion zur Signalbehandlung hat einen `int` – Parameter, die Signalnummer und hat keinen Rückgabewert.

58. Welche Signale kennen Sie?

Signal	Bedeutung
`SIGINT`	„interrupt“ Wird beim Drücken der Tastenkombination Strg+C an den Prozessor gesandt.
`SIGABRT`	„abort“ Wird von den Funktionen `assert()` und `abort()` verwendet.
`SIGTERM`	„terminate“ Wird beispielsweise beim Herunterfahren des Systems gesandt. Behandlung ist empfohlen.
`SIGILL`	„illegal instruction“ Tritt auf wenn ein illegaler Maschinenbefehl geladen wird.
`SIGFPE`	„floating point exception“ Tritt bei ungültigen arithmetischen Operationen auf. Z.B.: Division durch 0.
`SIGSEGV`	„segmentation violation“ Wird vom Prozessor gesendet, wenn versucht wird auf einen Speicherbereich zuzugreifen, der nicht zum Prozessor gehört oder wenn eine illegale Speicheradresse auftritt.

59. Statische und dynamische Felder:

60. Wie findet man die Nullstelle einer Funktion?

Als erstes setzt man eine obere und unter Grenze. Ist eine Grenze positiv und eine negativ, so befindet sich die Nullstelle noch im Bereich. Dann wird der Bereich halbiert. Ist wiederum eine Grenze positiv und eine negativ, so liegt die Nullstelle immer noch im Bereich. Sind beide Grenzen positiv oder negativ, so liegt der Bereich in dem gesucht wird oberhalb oder unterhalb der Nullstelle. Dieses Verfahren wird solange wiederholt, bis die Annäherung an die Nullstelle ausreichend genau ist.

61. Wie kann man in C numerisch integrieren?

Das Integral einer (stückweise stetigen und beschränkten) Funktion kann als Summe von Teilintegralen berechnet werden. Die einzelnen Summanden entsprechen jeweils der Integration über Teilintervalle. Es gibt verschiedene Methoden das Integral über diese Teilintervalle anzunähern.

Eine relativ einfache und anschauliche Methode ist die Trapezmethode, welche die Fläche unter der Funktionskurve durch Trapeze approximiert.

$$F_i \approx \frac{x_{i+1} - x_i}{2}\left(f(x_i) + f(x_{i+1})\right)$$

Eine Methode um die Fläche unter der Kurve genauer zu approximieren ist die Simpsonmethode:

$$F_i \approx \frac{x_{i+1} - x_i}{6}\left(f(x_i) + 4f\left(\frac{x_{i+1} - x_i}{2}\right) + f(x_{i+1})\right)$$

Außerdem gibt es noch die Newtonmethode:

$$F_i \approx \frac{x_{i+1} - x_i}{8}\left(f(x_i) + 3f\left(x_i + \frac{x_{i+1} - x_i}{3}\right) + 3f\left(x_i + 2\frac{x_{i+1} - x_i}{3}\right) + f(x_{i+1})\right)$$

Referenzbeispiel 4!