Universität Augsburg
Institut für Informatik

Probeklausur zur Vorlesung Informatik 1 WS 2019/20
Fakultät für Angewandte Informatik
Lehrprofessur für Informatik
Prof. Dr. Robert Lorenz 18.12.2019

Allgemeine Hinweise:

- Lassen Sie die Klausur **zusammengeheftet**.
- Es sind **keinerlei Hilfsmittel** zugelassen (auch kein Taschenrechner!).
- Die Bearbeitungszeit beträgt **120 Minuten**.
- Benutzen Sie einen **dokumentenechten Stift** (kein roter oder grüner Stift, kein Bleistift).
- Das Verlassen des Raumes ist **nur in Begleitung** und **nur bis 20 Minuten vor Ende** der Klausur erlaubt.
- Jeder muss auf der **Teilnahmeliste unterschreiben** (wir gehen damit von Platz zu Platz). Ohne Unterschrift ist die Abgabe nicht erlaubt.

- Die angegebenen Punkteverteilungen sind zur Orientierung gedacht (**Punkte=Minuten**).
- **Unterschleif** führt zur Bewertung *nicht bestanden* und kann zu weiteren Konsequenzen führen.

- Falls der vorgegebene Platz bei einer Aufgabe nicht ausreicht, dürfen Sie auch die Rückseiten verwenden **mit Verweis** auf die Aufgabe.
- Lesen Sie sich jede Aufgabe zuerst **komplett** durch, bevor Sie loslegen.
- Versteifen Sie sich nicht auf eine Aufgabe, falls Sie nicht weiterkommen. Bearbeiten Sie zunächst die für Sie **leicht machbaren Aufgaben**.
- Halten Sie sich bei den Programmieraufgaben an die Vorgaben der Spezifikation **C89**.

Vorname:_____ **Matrikelnummer:**_____

Nachname:_____ **Unterschrift:**_____

Vom Lehrstuhl auszufüllen:

Aufgabe	1 a) 8 P.	1 b) 6 P.		2 16 P.		3 a) 6 P.	3 b) 5 P.	3 c) 4 P.			Σ 45 Punkte
Punkte											
Aufgabe	4 a) 5 P.	4 b) 5 P.		5 a) 5 P.	5 b) 5 P.		6 a) 12 P.	6 b) 5 P.			Σ 37 Punkte
Punkte											
Aufgabe	7 a) 6 P.	7 b) 4 P.	7 c) 6 P.	7 d) 4 P.		8 a) 6 P.	8 b) 4 P.		9 a) 4 P.	9 b) 4 P.	Σ 38 Punkte
Punkte											

Aufgabe 1 *(Rechnerarchitektur, 14 Punkte)*

a) (Maschinenprogramm erstellen, 8 Punkte)

Nehmen Sie an, an den Stack-Adressen `S1` und `S2` sind nicht-negative ganze Zahlen gespeichert. Erstellen Sie ein Maschinenprogramm, das die an `S1` und `S2` gespeicherten Zahlen multipliziert und das Ergebnis zurückgibt.

Hinweise:

- *Es sind **ausschließlich** die Maschinenbefehle aus der Vorlesung zulässig (siehe Anhang).*
- *Für Adressen sind die Notationen aus der Vorlesung einzuhalten: der erste Befehl des Programms liegt an Adresse `P1`, der Datenteil beginnt mit Adresse `D1` und der Stack Frame mit Adresse `S1`.*
- *Nehmen Sie an, dass alle Zahlen jeweils in einer Speicherzelle gespeichert sind*

b) (Abarbeitung von Maschinenbefehlen durch die CPU, 6 Punkte)
Betrachten Sie das folgende Maschinenprogramm:

```
P1: INIT S2
P2: SPRUNGO P6,S1
P3: ADD S2,S1
P4: DEKREMENT S1
P5: SPRUNG P2
P6: RÜCKGABE S2
```

Beschreiben Sie die Abarbeitung des Maschinenbefehls an Adresse `P2` durch die CPU mittels **Fetch/Decode/Execute/Write-Back**.

Hinweise:

- *Verzichten Sie auf eine detaillierte Darstellung der Kommunikation zwischen Speicherwerk und Steuerwerk*
- *Geben Sie alle relevanten Register und deren jeweilige Belegung an.*

Aufgabe 2 *(Zahlencodierungen, 16 Punkte)*

In den folgenden Aufgaben ist kein Rechenweg gefordert. Es wird nur das Ergebnis bewertet.

1. (1 Punkt) Bestimmen Sie die 8-adische Darstellung von $(11010010)_2$.
2. (2 Punkte) Berechnen Sie $(A0.B1)_{16} + (55.EE)_{16}$ in der 16-adischen Darstellung.
3. (1 Punkt) Berechnen Sie $(1101)_2/2^2$ in der 2-adischen Darstellung als Festkommazahl.
4. (2 Punkte) Berechnen Sie $((1.01)_2 \cdot 2^1) - ((1.1)_2 \cdot 2^0)$ in der 2-adischen Darstellung als normierte Gleitkommazahl.
5. (1 Punkt) Berechnen Sie $c_{1K,8}(-8)$.
6. (1 Punkt) Berechnen Sie $(01101001)_{2K,8}$.
7. (1 Punkt) Berechnen Sie $0101 \oplus_{1K,4} 0001$
8. (1 Punkt) Berechnen Sie $1000 \ominus_{2K,4} 0101$.
9. (1 Punkt) Berechnen Sie $(10010100)_{EX-127,8}$.
10. (1 Punkt) Berechnen Sie $c_{FK,10,10}(0.1)$.
11. (2 Punkte) Berechnen Sie den absoluten Rundungsfehler bei der Codierung $c_{FK,23,23}(0.25)$.
12. (2 Punkte) Berechnen Sie $(1100111001000000)_{GK,11,16}$.

Aufgabe 3 *(Zeichenfolgen, 15 Punkte)*

a) (6 Punkte)
Implementieren Sie ohne Benutzung von Bibliotheksfunktionen oder Funktionen aus der Vorlesung eine Funktion

```
char * string_ncopy(char *v, char *w, int size)
```

zum sicheren Kopieren der Zeichenkette `w` in die Zeichenkette `v`, die sich genauso wie die Bibliotheksfunktion `strncpy` verhält.

b) (5 Punkte)
Implementieren Sie ohne Benutzung von Bibliotheksfunktionen oder Funktionen aus der Vorlesung eine Funktion

```
char * str_chr(const char *cs, int c)
```

die sich genauso wie die Bibliotheksfunktion `strchr` verhält.

c) (4 Punkte)
Implementieren Sie eine Funktion

```
int string_dd_copy(char **cpy, const char * org)
```

die die Zeichenkette `org` nach `*cpy` kopiert, dabei den Speicherplatz für die Kopie passend dynamisch reserviert, im Fehlerfall `0` liefert und sonst `1` liefert.

Aufgabe 4 *(Benutzereingaben, 10 Punkte)*

Erstellen Sie ein compilierbares und ausführbares C-Programm mit einer `main`-Funktion und einer weiteren Funktion nach folgenden Vorgaben.

a) (Einlesefunktion, 5 Punkte)
Implementieren Sie ohne Berücksichtigung von Pufferfehlern (EOF) eine Einlesefunktion

```
int read_probability(double *x)
```

die vom Benutzer eine Dezimalzahl zwischen einschließlich `0.0` und `1.0` einliest und an der übergebenen Adresse `x` speichert. Bei gültigen Eingaben soll die Funktion `1` zurückgeben. Bei ungültigen Eingaben soll die Funktion bei Bedarf den Eingabepuffer leeren und den Wert `0` zurückgeben.

b) (`main`-Funktion, 5 Punkte)
Lesen Sie vom Benutzer mit `read_probability` Eingaben ein, solange diese gültig sind und berechnen Sie das arithmetische Mittel der eingelesenen Zahlen. Bei der ersten ungültigen Eingabe soll das berechnete arithmetische Mittel ausgegeben werden und danach das Programm beendet werden. Ist schon die erste Eingabe ungültig, soll das Programm `0` ausgeben.

Aufgabe 5 *(Matrizen, 10 Punkte)*

Erstellen Sie nach folgenden Vorgaben Teile einer Übersetzungseinheit zur Verwaltung von Matrizen mit Einfachzeigern. Die Einträge der Matrizen sollen Dezimalzahlen sein.

a) (Headerdatei, 5 Punkte)
Erstellen Sie die zugehörige Headerdatei `smatrix.h`. Sie soll allgemein einsetzbar sein und Prototypen für folgende Funktionen enthalten:

1. Eine Funktion, die dynamisch Speicherplatz für eine über einen Einfachzeiger verwaltete Matrix reserviert.
2. Eine Funktion, die die Einträge einer mit einem Einfachzeiger verwalteten Matrix mit Zufallszahlen initialisiert.
3. Eine Funktion, die die Einträge einer mit einem Einfachzeiger verwalteten Matrix auf Kommandozeile ausgibt.
4. Eine Funktion, die den für eine mit einem Einfachzeiger verwaltete Matrix dynamisch reservierten Speicherplatz wieder freigibt.

b) (Ausgabefunktion, 5 Punkte)
Implementieren Sie die Funktion aus Unterpunkt 3 in Teilaufgabe a)..

Aufgabe 6 *(Prädikatenlogische Formeln, 17 Punkte)*

a) (Strukturelle Induktion, 12 Punkte)
Beweisen Sie mit struktureller Induktion: *Für jede aussagenlogische Formel p gibt es eine äquivalente aussagenlogische Formel q, die weder $\wedge$ noch $\vee$ enthält.*

b) (Problemspezifikation, 5 Punkte)
Welches Problem löst die C-Funktion `isdigit` aus `ctype.h`? Geben Sie eine zugehörige formale Problemspezifikation an.

Aufgabe 7 *(Algorithmen und Rekursion, 20 Punkte)*

a) (Algorithmus entwerfen, 6 Punkte)
Geben Sie zu folgender Problemspezifikation einen Algorithmus in Form eines Programmablaufplans an:
Eingabe: $a_1, \ldots, a_n \in \mathbb{R}$
Ausgabe: $s \in \mathbb{R}$
Funktionaler Zusammenhang: $s = \sum_{i=1}^{n} a_i$

b) (Rekursive Funktionen, 4 Punkte)
Betrachten Sie die folgende **induktive Definition** einer Funktion *exp*:

$$exp(n) = \begin{cases} 1 & , \text{ falls } n = 0 \\ 2 \cdot exp(n-1) & , \text{ falls } n > 0 \end{cases}$$

Implementieren Sie eine **rekursive C-Funktion** `int exp(int n)`, die bei Übergabe einer nichtnegativen ganzen Zahl `n` den Wert von `exp(n)` berechnet und zurückgibt.

c) (Partielle Korrektheit, 6 Punkte)
Sei folgende Problemspezifikation gegeben:
Eingabe: $a \in \mathbb{N}, a > 0$
Ausgabe: $z \in \mathbb{N}, z > 0$
Funktionaler Zusammenhang: $z = 6 \cdot a$

Weisen Sie die partielle Korrektheit des nachfolgenden Algorithmus bzgl. dieser Problemspezifikation nach, indem Sie geeignete Zusicherungen einfügen. Sie müssen dabei die Gültigkeit der einzelnen Zusicherungen nicht begründen (und insbesondere keine Regeln für Hoare-Tripel angeben).

Eingabe : $a \in \mathbb{N}$
$z \leftarrow 0$;
$i \leftarrow 1$;
solange $(i \leq a)$ **tue**
  $z \leftarrow z + 6$;
  $i \leftarrow i + 1$;
Ausgabe : z

d) (O-Notation, 4 Punkte)
Zeigen Sie $h \in O(\log(n))$ für $h(n) := \log(n^2)$.

Aufgabe 8 *(Komplexe Datenstrukturen, 10 Punkte)*

a) (Headerdateien für komplexe Datentrukturen entwerfen, 6 Punkte)
Schreiben Sie nach folgenden Vorgaben eine *allgemein einsetzbare Header-Datei* zur Verwaltung von **Mietfahrzeugen**. Dabei sollen für ein Mietfahrzeug der **Kilometerstand** und das **Kennzeichen** gespeichert werden.

1. Deklarieren Sie einen passenden neuen komplexen Datentyp. Legen Sie dazu insbesondere geeignete Datentypen für die Komponenten fest und verwenden Sie dabei bei Bedarf symbolische Konstanten.
2. Deklarieren Sie für alle Komponenten zugehörige `get-`, `set-` und `check`-Funktionen.
3. Deklarieren Sie eine `new`-Funktion zum Anlegen eines neuen Mietfahrzeugs.

b) (Verwaltungsfunktionen für komplexe Datentrukturen implementieren, 4 Punkte)
Betrachten Sie folgende komplexe Datenstruktur zur Verwaltung von **Aufgaben**.

```
typedef struct _task {
        int priority;
        char *name;
} task;
```

Sie soll die folgenden Datenstrukturinvarianten erfüllen:

- Die Priorität hat den Wertebereich $\{0, 1, 2, 3, 4\}$

Implementieren Sie die folgenden Verwaltungsfunktionen für diese Datenstruktur.

- `int task_check_priority(int priority)`
  Überprüft den übergebenen Wert `priority` auf Gültigkeit als Priorität, gibt im positiven Fall 1, und sonst 0 zurück.
- `int task_set_priority(task *task, int priority)`
  Setzt die Priorität der Aufgabe `*task` neu, gibt im Erfolgsfall 1, und sonst 0 zurück.

Aufgabe 9 *(Dynamische Datenstrukturen, 8 Punkte)*

a) (Dynamische Datentrukturen im Arbeitsspeicher, 4 Punkte)
Skizzieren Sie für die folgende dynamische Datenstruktur **Arraylist**, wie die Zahlenfolge `0,0,5` im Arbeitsspeicher unter Benutzung dieser Datenstruktur abgelegt wird:

```
typedef struct _natlist {
        int *elements;
        int size;
} arraylist;
```

Hinweise zum Anfertigen von Darstellungen von Daten im Arbeitsspeicher:

- Zeichnen Sie eine Speicherzelle pro Datenwert für die Datentypen **`char`**, **`int`**, **`double`** und adresswertige Datentypen, unabhängig vom Speicherbedarf dieser Datentypen.
- In die Speicherzellen schreiben Sie die Datenwerte, falls diese vom Typ **`char`**, **`int`** oder **`double`** sind. Speicherzellen mit undefinierten Werten lassen Sie leer.
- Werte von Zeigern geben Sie entweder mit `NULL` (zeigt nirgendwohin), oder als einen Pfeil, der nicht auf eine andere Speicherzelle zeigt (zeigt irgendwohin), oder als Pfeil zu einer anderen Speicherzelle an. Verwenden Sie keine konkreten Adresswerte.
- Zwischen nicht zwingend zusammenhängenden Speicherbereichen lassen Sie einen Abstand.

b) (Verwaltungsfunktionen für dynamische Datentrukturen implementieren, 4 Punkte)
Betrachten Sie die folgende dynamische Datenstruktur einer **einfach verketteten Liste** zur Verwaltung von Folgen ganzer Zahlen wie in der Vorlesung eingeführt.

```
typedef struct _node {
        int element;
        struct _node *next;
} node;
typedef node * list;
```

Implementieren Sie eine rekursive Funktion **`int`** `list_sum(list m)`, die die Summe aller Zahlen aus `m` zurückgibt.

Maschinenbefehle aus der Vorlesung

- `INIT A` : Speichere den Wert `0` an Adresse `A`
- `ADD A,B` : Addiere zum Inhalt an Adresse `A` den Inhalt an Adresse `B`
- `SUB A,B` : Subtrahiere vom Inhalt an Adresse `A` den Inhalt an Adresse `B`
- `DEKREMENT A`: Vermindere den Inhalt an Adresse `A` um `1`
- `DEKREMENT0 A,B`: Falls der Inhalt an Adresse `B` gleich `0` ist, vermindere den Inhalt an Adresse `A` um `1`
- `INKREMENT A`: Erhöhe den Inhalt an Adresse `A` um `1`
- `INKREMENT0 A,B`: Falls der Inhalt an Adresse `B` gleich `0` ist, erhöhe den Inhalt an Adresse `A` um `1`
- `SPRUNG A` : Gehe zu Adresse `A`
- `SPRUNG0 A,B` : Falls der Inhalt an Adresse `B` gleich `0` ist, gehe zu Adresse `A`
- `RÜCKGABE A` : Gib den Inhalt an Adresse `A` zurück
- `RÜCKGABE0 A,B` : Falls der Inhalt an Adresse `B` gleich `0` ist, gib den Inhalt an Adresse `A` zurück

Standard-Bibliothek (Auszug)

<string.h>

char *strcat(char *s, const char *ct)
Hängt ct an s an, liedert s.

char *strncat(char *s, const char *ct, site_t n)
Hängt höchstens n Zeichen von ct an s an, liefert s.

char *strcpy(char *s, const char *ct)
Kopiert ct nach s, liefert s.

char *strncpy(char *s, const char *ct, site_t n)
Kopiert höchstens n Zeichen von ct nach s, liefert s.

size_t strlen(const char *cs)
Liefert Länge von cs (ohne '\0').

int strcmp(const char * cs, const char * ct)
Vergleicht cs und ct; liefert <0 wenn cs<ct, 0 wenn cs==ct,oder >0, wenn cs>ct.

int strncmp(const char * cs, const char * ct, size_t n)
Vergleicht höchstens n Zeichen von cs und ct; liefert <0 wenn cs<ct, 0 wenn cs==ct,oder >0, wenn cs>ct.

char *strchr(const char * cs, int c)
Liefert Zeiger auf das erste c in cs oder NULL, falls nicht vorhanden.

char *strrchr(const char * cs, int c)
Liefert Zeiger auf das letzte c in cs oder NULL, falls nicht vorhanden.

size_t strspn(const char * cs, const char * ct)
Liefert Anzahl der Zeichen am Anfang von cs, die sämtlich in ct vorkommen.

size_t strcspn(const char * cs, const char * ct)
Liefert Anzahl der Zeichen am Anfang von cs, die sämtlich nicht in ct vorkommen.

char *strpbrk(const char * cs, const char * ct)
Liefert Zeiger auf die Position in cs, an der irgendein Zeichen aus ct erstmals vorkommt, oder NULL, falls keines vorkommt.

char *strstr(const char * cs, const char * ct)
Liefert Zeiger auf erste Kopie von ct in cs oder NULL, falls nicht vorhanden.

char *strtok(char * s, const char * ct)
Beim ersten Aufruf ist s nicht NULL, er liefert die erste Zeichenfolge in s, die nicht aus Zeichen in ct besteht; bei jedem weiteren Aufruf wird NULL für s übergeben, er liefert die nächste derartige Zeichenfolge, wobei unmittelbar nach dem Ende der vorhergehenden mit der Suche begonnen wird; liefert NULL, wenn keine weitere Zeichenfolge gefunden wird.

<stdlib.h>

int atoi(const char *s)
Wandelt den Anfang der Zeichenkette s in **int** um.

`double atof(const char *s)`
Wandelt den Anfang der Zeichenkette `s` in `double` um.

`void *malloc(size_t size)`
Liefert einen Zeiger auf einen Speicherbereich für ein Objekt der Größe `size` oder `NULL`, wenn die Anforderung nicht erfüllt werden kann. Der Bereich ist nicht initialisiert.

`void free(void *p)`
Gibt den Bereich frei, auf den `p` zeigt; die Funktion hat keinen Effekt, wenn `p` den Wert `NULL` hat. `p` muss auf einen Bereich zeigen, der zuvor mit `calloc`, `malloc` oder `realloc` angelegt wurde.

`long strtol(const char *s, char **endp, int base)`
Wandelt den Anfang der Zeichenkette `s` in `long` um. Speichert einen Zeiger auf einen nicht umgewandelten Rest bei `*endp`, falls `endp` nicht `NULL` ist. Hat `base` einen Wert zwischen `2` und `36`, erfolgt die Umwandlung unter der Annahme, dass die Eingabe in dieser Basis repräsentiert ist.

`double strtod(const char *s, char **endp)`
Wandelt den Anfang der Zeichenkette `s` in `double` um. Speichert einen Zeiger auf einen nicht umgewandelten Rest bei `*endp`, falls `endp` nicht `NULL` ist.

`RAND_MAX`
Maximaler Rückgabewert von `rand`

`int rand(void)`
Liefert eine ganzzahlige Pseudo-Zufallszahl im Bereich von `0` bis `RAND_MAX`.

`void srand(unsigned int seed)`
Setzt `seed` als Ausgangswert für eine neue Folge von Pseudo-Zufallszahlen.

`int abs(int n)`
Liefert den absoluten Wert seines `int`-Arguments.

`void *calloc(size_t nobj, size_t size)`
Liefert einen Zeiger auf einen Speicherbereich für einen Vektor von `nobj` Objekten, jedes mit der Größe `size`, oder `NULL`, wenn die Anforderung nicht erfüllt werden kann. Der Bereich wird mit Null-Bytes initialisiert.

`void *realloc(void *p, size_t size)`
Ändert die Größe des Objekts, auf das `p` zeigt, in `size` ab. Bis zur kleineren der alten und neuen Größe bleibt der Inhalt unverändert. Wird der Bereich für das Objekt größer, so ist der zusätzliche Bereich uninitiatisiert. Liefert einen Zeiger auf den neuen Bereich oder `NULL`, wenn die Anforderung nicht erfüllt werden kann; in diesem Fall ist `*p` unverändert.

`EXIT_SUCCESS`
Statuswert für das das erfolgreiche Beenden von `main`.

`EXIT_FAILURE`
Statuswert für das das fehlerhafte Beenden von `main`.

`void exit(int status)`
Beendet das Programm normal. Wie `status` an die Umgebung des Programms geliefert wird, hängt von der Implementierung ab, aber Null gilt als erfolgreiches Ende. Die Werte `EXIT_SUCCESS` und `EXIT_FAILURE` können ebenfalls angegeben werden.

`void *bsearch(const void *key, const void *base, size_t n, size_t size,`

`int (*cmp)(const void *keyval, const void *datum))`
Durchsucht `base[0]`, ... , `base[n-1]` nach einem Eintrag, der gleich `*key` ist. Die Funktion `cmp` muß einen negativen Wert liefern, wenn ihr erstes Argument (der Suchschlüssel) kleiner als ihr zweites Argument (ein Tabelleneintrag) ist, Null, wenn beide gleich sind, und sonst einen positiven Wert. Die Elemente des Vektors `base` müssen aufsteigend sortiert sein. Liefert einen Zeiger auf das gefundene Element oder `NULL`, wenn keines existiert.

`void qsort(void *base, size_t n, size_t size,`
`int (*cmp)(const void *, const void *))`
Sortiert den Vektor `base[0]`, ... , `base[n-1]` von Objekten der Größe `size` in aufsteigender Reihenfolge. Für die Vergleichsfunktion `cmp` gilt das gleiche wie bei `bsearch`

<stdio.h>

`int getchar(void)`
Liefert das nächste Zeichen aus dem Standard-Eingabestrom als `unsigned char` (umgewandelt in `int)` oder `EOF` bei Dateiende oder bei einem Fehler.

`int scanf(const char *format, ...)`
Liest vom Standard-Eingabestrom unter Kontrolle von `format` und legt umgewandelte Werte mit Hilfe von nachfolgenden Argumenten ab, die alle Zeiger sein müssen. Die Funktion wird beendet, wenn `format` abgearbeitet ist. Liefert `EOF`, wenn vor der ersten Umwandlung das Dateiende erreicht wird oder ein Fehler passiert; liefert andernfalls die Anzahl der umgewandelten Eingaben.

`int printf(const char *format, ...)`
Wandelt Ausgaben um und schreibt sie in den Standard-Ausgabestrom unter Kontrolle von `format`. Der Resultatwert ist die Anzahl der geschriebenen Zeichen; er ist negativ, wenn ein Fehler passiert ist.

`int putchar(int c)`
Schreibt das `c` in den Standardausgabestrom; Liefert das ausgegebene Zeichen oder `EOF` bei Fehler.

`int sprintf(char *s, const char *format, ...)`
Funktioniert wie `printf`, nur wird die Ausgabe in `s` geschrieben und mit `'\0'` abgeschlossen. Im Resultatwert wird `'\0'` nicht mitgezählt.

<time.h>

`time_t time(time_t *tp)`
Liefert die aktuelle Kalenderzeit oder `-1`, wenn diese nicht zur Verfügung steht.

<ctype.h>

`int isdigit(int c)`
Liefert einen von `0` verschiedenen Wert, wenn `c` eine dezimale Ziffer ist, sonst `0`.

`int islower(int c)`
Liefert einen von `0` verschiedenen Wert, wenn `c` ein lateinischer Kleinbuchstabe ist, sonst `0`.

`int isupper(int c)`
Liefert einen von `0` verschiedenen Wert, wenn `c` ein lateinischer Großbuchstabe ist, sonst `0`.

`int isalpha(int c)`
Liefert einen von `0` verschiedenen Wert, wenn `c` ein lateinischer Buchstabe ist, sonst `0`.

`int isspace(int c)`
Liefert einen von `0` verschiedenen Wert, wenn `c` ein Zwischenraumzeichen ist, sonst `0`.

`int isalnum(int c)`
Liefert einen von `0` verschiedenen Wert, wenn `c` ein lateinischer Buchstabe oder eine dezimale Ziffer ist, sonst `0`.

`int ispunct(int c)`
Liefert einen von `0` verschiedenen Wert, wenn `c` ein sichtbares Zeichen, aber kein Leerzeichen, lateinischer Buchstabe oder dezimale Ziffer ist, sonst `0`.

`int isprint(int c)`
Liefert einen von `0` verschiedenen Wert, wenn `c` ein sichtbares Zeichen ist, sonst `0`.

`int tolower(int c)`
Wandelt `c` in einen lateinischen Kleinbuchstaben um.

`int toupper(int c)`
Wandelt `c` in einen lateinischen Großbuchstaben um.

`<limits.h>`

`CHAR_BIT`
Bits in einem **`char`**.

`CHAR_MAX`
Maximaler Wert für **`char`**.

`CHAR_MIN`
Minimaler Wert für **`char`**.

`INT_MAX`
Maximaler Wert für **`int`**.

`INT_MIN`
Minimaler Wert für **`int`**.

`LONG_MAX`
Maximaler Wert für **`long`**.

`LONG_MIN`
Minimaler Wert für **`long`**.

`UCHAR_MAX`
Maximaler Wert für **`unsigned char`**.

`UINT_MAX`
Maximaler Wert für **`unsigned int`**.

`ULONG_MAX`
Maximaler Wert für **`unsigned long`**.

<float.h>

`DBL_MAX`
Maximaler Wert für **`double`**.

`DBL_MIN`
Minimaler normalisierter Wert für **`double`**.

<math.h>

`double log(double x)`
Liefert $\ln(x)$, $x > 0$.

`double exp(double x)`
Liefert e^x.

`double log10(double x)`
Liefert $\log_{10}(x)$, $x > 0$.

`double pow(double x, double y)`
Liefert x^y, Argumentfehler bei $x = 0$ und $y < 0$, oder bei $x < 0$ und y nicht ganzzahlig.

`double sqrt(double x)`
Liefert $\sqrt{x}$, $x \geq 0$.

`double ceil(double x)`
Liefert kleinsten ganzzahligen Wert, der nicht kleiner als x ist.

`double floor (double x)`
Liefert größten ganzzahligen Wert, der nicht größer als x ist.

`double fabs(double x)`
Liefert $|x|$.

`double modf(double x, double *ip)`
Zerlegt x in einen ganzzahligen Teil und einen Rest, die beide das gleiche Vorzeichen wie x besitzen. Der ganzzahlige Teil wird bei `*ip` abgelegt, der Rest ist das Resultat.

`double frexp(double x, int *exp)`
Zerlegt x in eine normalisierte Mantisse im Bereich $[\frac{1}{2}, 1]$, die als Resultat geliefert wird, und eine Potenz von 2, die bei `*exp` abgelegt wird. Ist x null, sind beide Teile des Resultats null.

ASCII-Tabelle

0	NUL	**16**	DLE	**32**	SPC	**48**	0	**64**	@	**80**	P	**96**	`	**112**	p
1	SOH	**17**	DC1	**33**	!	**49**	1	**65**	A	**81**	Q	**97**	a	**113**	q
2	STX	**18**	DC2	**34**	"	**50**	2	**66**	B	**82**	R	**98**	b	**114**	r
3	ETX	**19**	DC3	**35**	#	**51**	3	**67**	C	**83**	S	**99**	c	**115**	s
4	EOT	**20**	DC4	**36**	$	**52**	4	**68**	D	**84**	T	**100**	d	**116**	t
5	ENQ	**21**	NAK	**37**	%	**53**	5	**69**	E	**85**	U	**101**	e	**117**	u
6	ACK	**22**	SYN	**38**	&	**54**	6	**70**	F	**86**	V	**102**	f	**118**	v
7	BEL	**23**	ETB	**39**	'	**55**	7	**71**	G	**87**	W	**103**	g	**119**	w
8	BS	**24**	CAN	**40**	(	**56**	8	**72**	H	**88**	X	**104**	h	**120**	x
9	HT	**25**	EM	**41**	)	**57**	9	**73**	I	**89**	Y	**105**	i	**121**	y
10	LF	**26**	SUB	**42**	*	**58**	:	**74**	J	**90**	Z	**106**	j	**122**	z
11	VT	**27**	ESC	**43**	+	**59**	;	**75**	K	**91**	[	**107**	k	**123**	{
12	FF	**28**	FS	**44**	,	**60**	<	**76**	L	**92**	\	**108**	l	**124**	|
13	CR	**29**	GS	**45**	-	**61**	=	**77**	M	**93**	]	**109**	m	**125**	}
14	SO	**30**	RS	**46**	.	**62**	>	**78**	N	**94**	^	**110**	n	**126**	~
15	SI	**31**	US	**47**	/	**63**	?	**79**	O	**95**	_	**111**	o	**127**	DEL

Abbildung 1: ASCII-Tabelle von 0 bis 127