Universität Augsburg
Institut für Informatik

WiSe 2020/2021
26. Februar 2021
Dr. Martin Frieb

Informatik 1
Probe Klausur

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).
- Bitte beachten Sie die zusätzlich verteilten Informationen zum **Ablauf der Klausur unter Corona-Bedingungen**.

- Die angegebenen Punkteverteilungen sind zur Orientierung gedacht (**Punkte=Minuten**).
- **Unterschleif/Täuschungsversuch** 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:**_____

Nur vom Lehrstuhl auszufüllen

Aufgabe	1	2	3	4	5	6	Summe
Punkte							
von	17	12	18	31	17	25	120

Prüfungsraum: **Messehalle 5** **Platznummer:** **600**

1. Aufgabe (Rechnerarchitektur) 17 Punkte

a) Schreiben Sie ein Maschinenprogramm, das die folgende C-Funktion realisiert. Geben Sie dazu jeweils an, welche Adressen Sie für Eingabeparameter und lokale Variablen benutzen. *(10 Punkte)*

```
1  int is_odd(int a)
2  {
3          int r = 0;
4          while (a != 0) {
5                  if (r == 0) {
6                          r = r + 1;
7                  } else {
8                          r = r - 1;
9                  }
10                 a = a - 1;
11         }
12         return r;
13 }
```

Beachten Sie:

- Es sind **ausschließlich** die Maschinenbefehle aus der Vorlesung zulässig (siehe Anhang).
- Für Adressen sind die Notationen aus Kapitel 2.7 einzuhalten, insbesondere: Der erste Befehl des Programms liegt an Adresse `P1`, der Datenteil beginnt mit Adresse `D1` und der Stack Frame mit Adresse `S1`.
- Es wird angenommen, dass an Adressen im Datenteil und im Stack, deren Inhalt nicht im Programm mit `0` initialisiert wird, bei Programmstart nur **nicht-negative ganze Zahlen** gespeichert sind (da die in der Vorlesung eingeführten Maschinenbefehle nicht geeignet sind für Berechnungen mit negativen Zahlen).

b) In dieser Aufgabe sollen Sie detailliert die Kommunikation zwischen Steuerwerk und Speicherwerk über Register und Steuersignale bei Schreib- und Lesevorgängen beschreiben.

Geben Sie dabei insbesondere alle relevanten Register und deren jeweilige Belegung an.

Betrachten Sie dazu das folgende Maschinenprogramm:

```
P1: INIT S2
P2: SPRUNG0 P6,S1
P3: ADD S2,S1
P4: DEKREMENT S1
P5: SPRUNG P2
P6: RÜCKGABE S2
```

Beschreiben Sie detailliert die Kommunikation zwischen Steuerwerk und Speicherwerk beim Lesen des ersten Operanden im Decode-Schritt der Abarbeitung des Maschinenbefehls an Adresse `P3`. *(7 Punkte)*

2. Aufgabe (Zahlencodierung) 12 Punkte

In den folgenden Aufgaben ist kein Rechenweg gefordert. Es wird nur das Ergebnis bewertet. Tragen Sie das Ergebnis jeweils an der vorgesehenen Stelle ein. Der Platz unter dem Ergebnisfeld können Sie jeweils für Nebenrechungen nutzen.

Geben Sie die 2-adische Darstellung der Zahl $(0.4)_{16}$ an (1 Punkt)	

Geben Sie die Zahl $(0.000117)_8$ als normierte Gleitkommazahl an (1 Punkt)	

Geben Sie das Ergebnis der Rechnung $((4.0)_8 \cdot 8^{-1}) \cdot ((2.1)_8 \cdot 8^{-2})$ in der 8-adischen Darstellung an (2 Punkte)	

Berechnen Sie $11010010 \oplus_{2K,8} 10110001$. (2 Punkte)	

In den folgenden Aufgaben ist kein Rechenweg gefordert. Es wird nur das Ergebnis bewertet. Tragen Sie das Ergebnis jeweils an der vorgesehenen Stelle ein. Der Platz unter dem Ergebnisfeld können Sie jeweils für Nebenrechungen nutzen.

Berechnen Sie $c_{GK,5,8}(14.4)$ und geben Sie den absoluten Rundungsfehler an, der dabei auftritt (4 Punkte)	

Welche $EX-q$-Codierung wird zur Codierung des Exponenten in der Gleitkomma-Codierung $c_{GK,53,64}$ verwendet? (2 Punkte)	

3. Aufgabe (C-Funktionen) 18 Punkte

a) Implementieren Sie ohne Berücksichtigung von Pufferfehlern (`EOF`) eine Einlesefunktion
`int read_age(int *age)`
die vom Benutzer eine ganze Zahl zwischen einschließlich `0` und `150` einliest und an der übergebenen Adresse `age` 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. *(6 Punkte)*

b) Implementieren Sie eine Funktion
`int *array_d_copy(int v[], int n)`
die die ersten `n` Komponenten von `v` kopiert und den Speicherplatz für die Kopie dabei dynamisch reserviert. Die Funktion soll im Erfolgsfall einen Zeiger auf die Kopie und sonst `NULL` liefern. *(6 Punkte)*

c) Implementieren Sie eine Funktion
`unsigned long int fibonacci(void)`
die beim n-ten Aufruf die n-te **Fibonacci-Zahl** f_n zurückgibt. *(6 Punkte)*

Die Fibonacci-Zahlen sind dabei wie folgt induktiv definiert:

- $f_1 := 0$
- $f_2 := 1$
- $f_{n+1} := f_n + f_{n-1}$ für $n \geq 2$

4. Aufgabe (C-Programme) 31 Punkte

a) Erstellen Sie ein kompilierbares und ausführbares C-Programm mit einer `main`-Funktion und weiteren Funktionen nach folgenden Vorgaben: *(17 Punkte)*

- Legen Sie eine systemunabhängige symbolische Konstante `VMAX` für die maximale Länge von `int`-Feldern in Ihrem Programm an.
- Das Programm soll genau einen Kommandozeilenparameter erwarten. Übergibt der Benutzer weniger oder mehr Kommandozeilenparameter, so soll das Programm eine passende Fehlermeldung auf der Kommandozeile ausgeben und mit einem Fehlerwert ungleich Null abbrechen.
- Entspricht der übergebene Parameter einer nicht-leeren Zeichenkette, die nur aus Ziffern besteht und nicht mit `0` beginnt, so soll der Parameter in eine ganze Zahl `n` umgewandelt werden:
  - Ist `n` größer als `VMAX`, so soll das Programm eine passende Fehlermeldung auf der Kommandozeile ausgeben und mit einem Fehlerwert ungleich Null abbrechen.
  - Ansonsten sollen `n` ganze Zufallszahlen zwischen einschließlich `0` und `999` in einem `int`-Feld der Länge `VMAX` gespeichert werden.
- Hat der übergebene Parameter ein anderes als die beschriebenen Formate, so soll das Programm eine passende Fehlermeldung auf der Kommandozeile ausgeben und mit einem Fehlerwert ungleich Null abbrechen.
- Verwenden Sie wo möglich passende Bibliotheksfunktionen, und legen Sie sinnvolle eigene Funktionen für verschiedene Teilaufgaben an.

b) Erstellen Sie eine allgemein einsetzbare Übersetzungseinheit, bestehend aus einer Header- und einer C-Datei, mit folgenden Bestandteilen:

- Ein Makro, um auf der Kommandozeile die Nachricht auszugeben, dass eine an das Makro übergebene Zahl eine Primzahl ist (mit Ausgabe der Zahl).
  Hinweis: Das Makro soll **nicht** *testen, ob die Zahl eine Primzahl ist.*
  Beispielausgabe eines Aufrufs des Makros: `17 ist prim`
- Ein Makro zur Berechnung des Abstands zweier Zahlen.
- Eine Funktion, um zu testen, ob eine positive ganze Zahl eine Primzahl ist.
- Eine Funktion, um zu testen, ob eine positive ganze Zahl eine Quadratzahl ist.

Legen Sie dabei selbst den Namen von Header- und C-Datei, die Namen von symbolischen Konstanten und Makros sowie die Prototypen der Funktionen geeignet fest. Verwenden Sie zur Implementierung der Makros und Funktionen keine Bibliotheksfunktionen (Ausnahme: `printf`). *(14 Punkte)*

5. Aufgabe (Datenstrukturen) 17 Punkte

In den ersten Teilaufgaben fertigen Sie jeweils eine Arbeitsspeicher-Skizze nach folgenden Vorgaben an:

- 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.

a) Betrachten Sie die folgenden Anweisungen:

```
int (*p)[4] = malloc(2 * 4 * sizeof(int));
p[1][2] = 3;
```

Skizzieren Sie die Belegung des Arbeitsspeichers nach Abarbeitung dieser Anweisungen, falls es dabei zu keinem Fehler gekommen ist. *(2 Punkte)*

b) Wie wird die Matrix

$$\begin{pmatrix} 7 & 1 & 4 & 2 \\ 0 & 4 & 3 & 6 \end{pmatrix}$$

im Arbeitsspeicher abgelegt, wenn sie durch einen Doppelzeiger `int **m` repräsentiert wird, der notwendige Speicherbereich dynamisch genau passend reserviert wurde und die Werte in einer Zeile jeweils in aufeinanderfolgenden Speicherzellen abgelegt werden? *(4 Punkte)*

c) Wie wird die Liste von Zeichenketten
`"a"`
`"bc"`
`""`
im Arbeitsspeicher abgelegt, wenn sie durch ein Feld von Zeigern `char *m[4]` repräsentiert wird und der notwendige Speicherbereich dynamisch genau passend reserviert wurde? *(3 Punkte)*

d) Betrachten Sie die dynamische Datenstruktur eines **dynamischen Felds** (Arraylist) zur Verwaltung von Folgen ganzer Zahlen:

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

Implementieren Sie eine Funktion `int arraylist_get(arraylist *m, int index)`, die das Element an Position `index` zurückgibt, falls es existiert, und sonst `INT_MIN` zurückgibt. Die Indizierung soll bei 0 starten. *(3 Punkte)*

e) Implementieren Sie eine Funktion, die die Einträge einer mit einem Einfachzeiger verwalteten Matrix übersichtlich auf Kommandozeile ausgibt. Finden Sie selbst einen passenden Prototyp für die Funktion. Die Matrix soll `int`-Werte speichern können. *(5 Punkte)*

6. Aufgabe (Induktion, Problemspez., Algorithmen) 25 Punkte

a) Beweisen Sie mit vollständiger Induktion: $\forall n \in \mathbb{N}\backslash\{1,2\}\,(2 \cdot n + 1 < 2^n)$
Machen Sie dabei deutlich, was Induktionsanfang, Induktionsvoraussetzung und Induktionsschritt sind und wo Sie die Induktionsvoraussetzung jeweils einsetzen. *(5 Punkte)*

b) Geben Sie für die folgende natürlichsprachliche Problemstellung eine formale Problemspezifikation an: Sei A ein geordnetes Alphabet und $<$ die lexikographische Ordnung auf A^*. Bestimme für Wörter $u, v \in A^*$, ob $u < v$, $u = v$ oder $u > v$. *(5 Punkte)*

c) Drücken Sie die folgende natürlichsprachliche Aussage als prädikatenlogische Formel aus: Das Produkt von zwei ungeraden natürlichen Zahlen ist eine ungerade Zahl. *(2 Punkte)*

d) Finden Sie für die folgende prädikatenlogische Formel eine möglichst prägnante und intuitive natürlichsprachliche Aussage: $\forall x \in \mathbb{R}(((x > 0) \vee (x < 0)) \Rightarrow \exists z \in \mathbb{R}(x \cdot z = 1))$ *(2 Punkte)*

e) Beweisen oder widerlegen Sie: $h \in O(n^2)$ für $h(n) := 3 \cdot n^2 - 2 \cdot n + 10$ *(4 Punkte)*

f) Geben Sie den folgenden als Struktogramm gegebenen Algorithmus als Pseudocode an. *(7 Punkte)*

Algorithmus: prefix

Eingabe: $w_1 \dots w_n, v_1 \dots v_m \in A^*$ $(n, m \in \mathbb{N})$		
$i \leftarrow 1$		
Solange $i \leq m$		
tue	ja — $i > n \vee w_i \neq v_i$ — nein	
	Ausgabe: 0	
	$i \leftarrow i + 1$	
Ausgabe: 1		

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`
- `DEKREMENTO 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`
- `INKREMENTO 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`
- `SPRUNGO A,B` : Falls der Inhalt an Adresse `B` gleich `0` ist, gehe zu Adresse `A`
- `RÜCKGABE A` : Gib den Inhalt an Adresse `A` zurück
- `RÜCKGABEO A,B` : Falls der Inhalt an Adresse `B` gleich `0` ist, gib den Inhalt an Adresse `A` zurück

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

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, size_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, size_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 in ct vorkommen.

`size_t strcspn(const char * cs, const char * ct)`
Liefert Anzahl der Zeichen am Anfang von cs, die 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>

Vereinbart Funktionen zum Testen von Zeichen. Jede Funktion hat ein `int`-Argument, dessen Wert entweder `EOF` ist oder als `unsigned char` dargestellt werden kann, und der Resultatwert hat den Typ `int`. Die Funktionen liefern einen von Null verschiedenen Wert (wahr), wenn das Argument `c` die beschriebene Bedingung erfüllt; andernfalls liefern sie Null.

`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.