Semesterendprüfung INF1
Probeprüfung

Dozenten:	E. Bazzi, G. Burkert, K. Rege, M. Thaler	Studiengänge:	ET/ST
Klassen:	ET17a, ET17b, ET17t, ST17a, ST17b	Datum/Zeit:	16.12.2021, 8:00 – 9:30

Name, Vorname	**Klasse**

Maximale Punktzahl:	60 (fünf Aufgaben à 12 Punkte)
Dauer:	90 Minuten
Hilfsmittel:	Persönliche Zusammenfassung max. zwei Blätter A4 beidseitig beschrieben, keine anderen Hilfsmittel erlaubt, insbesondere **keinerlei elektronische Hilfsmittel**
Referenz:	Eine Kurzreferenz (C Reference Card, 2 Seiten) befindet sich am Ende dieser Aufgabenstellung
Vorgehen:	Füllen Sie die Prüfungsmappe und dieses erste Aufgabenblatt aus; schreiben Sie die Lösung soweit möglich direkt auf die Aufgabenblätter
Abgabe:	Beschriften Sie alle separaten Lösungsblätter mit Ihrem Namen, Vornamen und Ihrer Klasse sowie der entsprechenden Aufgabennummer; achten Sie darauf, alle Aufgaben- und Lösungsblätter abzugeben; legen Sie alle Aufgaben- und Lösungsblätter in die Prüfungsmappe und lassen Sie diese auf Ihrem Tisch liegen
Bitte:	Schreiben Sie weder mit Bleistift noch mit roter Farbe

Bewertung:

Aufgabe:	1	2	3	4	5	Total	Note
Punkte:							

Aufgabe 1: Zahlen, Variablen, Ausdrücke (12 Punkte)

a) Gegeben ist ein Programm mit einigen Ausdrücken. Markieren Sie, welche der folgenden Ausdrücke übersetzt werden oder die Kompilation des Programms mit einer Fehlermeldung verhindern. (3 Punkte)

Code	**übersetzt**	**Fehler**
`int main(void) {`		
`  int i;`		
`  char c = 'c';`		
`  int *****ip;`		
`  i = 'a'+ c;`		
`  c = (i==1);`		
`  i++ = c;`		
`  while (i--) i++;`		
`  return 0;`		
`}`		

b) Welche Werte nehmen die Variablen nach den Anweisungen an? (Hinweis: Zuweisung und Vergleichsoperator) (2 Punkte)

```
int a=0, b=0, c=1, d=1, e=1;
b = (a == 1);
if (a = 1) c = 2;
d = e++;
```

a: _____ b: _____ c: _____ d: _____ e: _____

c) Was wird ausgegeben? (1 Punkt)

```
void main(void)  {
    int x=20, y=25, z;
    z = (int)sqrt(y) / (int)sqrt(x);
    printf("%d%d",y /(x/4),z);
}
```

d) Was wird ausgegeben? (1 Punkt)

```
void main(void){
  int a=2;
  switch(a) {
    case 4: printf("A");
    break;
    case 3: printf("B");
    case 2: printf("C");
    case 1: printf("D");
    break;
    default: printf("E");
  }
}
```

e) Wie oft wird die Schleife in untenstehendem Programm durchlaufen? (1 Punkt)

```
void main(void) {
    int i;
    for(i=9;i;i=i/2) {
        printf("\n%d",i);
    }
}
```

f) Was wird ausgegeben? (1 Punkt)

```
void main(void) {
  printf("One");
  if (2>1)
    printf("Two");
  else
    printf("Three");
    printf("Four");
}
```

g) Gegeben ist folgende Programmsequenz; geben Sie an, was jeweils ausgegeben wird (3 Punkte)

```
const int J=4;
const int K=3;
const double E = 3.0;
const double F = 2.0;
```

```
printf ("%d", J/K);          _____
printf ("%d", J&K);          _____
printf ("%d", J&&K);         _____
printf ("%f", E + J/K);      _____
printf ("%f", (E + J)/K);    _____
printf ("%f", J/K*1.0);      _____
printf ("%f", J/(K*1.0));    _____
printf ("%f", (int)E/F);     _____
printf ("%d", J<<K);         _____
printf ("%d", K<<J);         _____
```

Aufgabe 2: Kontrollstrukturen (12 Punkte)

a) Gegeben ist folgender Programmausschnitt:

```
char c;
...
if (c == 'm') {
    printf("Montag\n");
} else if ((c == 'd') || (c == 'D')) {
    printf("Dienstag\n");
} else {
    printf("Mi-So\n");
}
```

Schreiben Sie den Programmausschnitt neu mit einer *switch*-Anweisung, wobei er so kurz wie möglich sein muss (fliesst in Bewertung ein). (8 Punkte)

b) Gegeben ist untenstehender Programmcode:

```
int Tiercode;
...
if (Tiercode >= 1000)
    if (Tiercode < 2000)
        printf("Katze\n");
```

Scheiben Sie den Code neu mit nur *einer* if-Anweisung. (2 Punkte)

c) Gegeben ist untenstehender Programmcode:

```
int Tiercode;
...
if (Tiercode < 2000)
    if (Tiercode <= 1000)
        printf("Katze\n");
else
    printf("Hund\n");
printf("Fertig\n");
```

Was gibt der Code aus, wenn `Tiercode` den Wert `2000` hat? (2 Punkte)

Aufgabe 3: Arrays, Zeiger, Strings (12 Punkte)

a) Gegeben ist folgendes Programm:

```
#include <stdio.h>

#define W_SIZE 10
#define LINES  10

int func(int *ptr) {
    return *ptr;
}

int main(void) {
    char fahrzeug[LINES][W_SIZE] = {"Auto", "Toeff", "Velo", "Bus"};
    int  v[5] = {100, 200, 300, 400, 500};
    int  a, b, c, d, r1 = 10, r2 = 20;
    int *iPtr, *kPtr;

    a = func(&v[3]);

    iPtr = &r1;
    kPtr = v;

    b = *iPtr + *(kPtr + 1);
    c = *kPtr + r2;
    d = *iPtr - *(&kPtr[4]);

    // Aufgabe a)
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    printf("c = %d\n", c);
    printf("d = %d\n", d);

    return 0;
}
```

Welche Werte werden für die Variablen a, b, c und d ausgegeben? (8 Punkte)

a = _____

b = _____

c = _____

d = _____

b) Alle der untenstehenden Anweisungen kompilieren (zum Teil mit Warnung), wenn sie in das Programm aus Aufgabenteil a) unten vor „return 0;“ eingesetzt werden. Welche der Anweisungen geben sicher eines der Fahrzeuge auf dem Bildschirm aus (kein Laufzeitfehler)? Kreuzen Sie die die entsprechenden Antworten an. (4 Punkte, 1P pro Anweisung, falls richtig)

Anweisung	Ausgabe
`printf("Das Fahrzeug ist ein %s\n", fahrzeug);`	☐
`printf("Das Fahrzeug ist ein %s\n", &fahrzeug[2]);`	☐
`printf("Das Fahrzeug ist ein %s\n", fahrzeug[2][0]);`	☐
`printf("Das Fahrzeug ist ein %s\n", *fahrzeug[3]);`	☐

Aufgabe 4: Bit-Operationen (12 Punkte)

Zur Ausgabe von bestimmten Statusinformationen soll in dieser Aufgabe eine Reihe von vier Lämpchen dienen, welche in verschiedenen Farben leuchten können. Angenommen, wir haben bereits einfache Funktionen zur Initialisierung und zur Ansteuerung der Lämpchen zur Verfügung:

`void initLights(void)`
Initialisiert die Lämpchenreihe, alle sind erst einmal dunkel.

`void switchLight(int id, char state[])`
Schaltet ein Lämpchen in einen bestimmten Zustand.
id: Nummer des Lämpchens, das ganz rechts hat id=0, das nächste 1, dann 2 und ganz links 3.
state: "off"... ausgeschaltet, "red", "green", "blue", "orange"... es leuchtet in dieser Farbe.

Nach dem Aufruf von *initLights()* sind erst einmal alle vier Lämpchen dunkel:

Hier ist ein kleines Testprogramm:

```
int main (void) {
    initLights();

    switchLight(0, "red");
    switchLight(1, "green");
    switchLight(2, "blue");
    switchLight(3, "orange");

    switchLight(3, "off");
    return 0;
}
```

Dieses Programm stellt den folgenden Zustand her:

 (von links nach rechts: aus – blau – grün – rot)

a. **Blinkendes Lämpchen** (3 Punkte). In einem ersten Beispiel sollen alle Lämpchen dunkel sein, nur das ganz rechts fünfmal kurz rot aufblinken. Ergänzen Sie das Programm. Verwenden Sie einen geeigneten Schleifentyp. Setzen Sie dabei die folgende Funktion als gegeben voraus:

`void delay(double time)`
Unterbricht die Programmausführung für die im Parameter *time* angegebene Anzahl Sekunden.

```
int main (void) {
    int i;
    initLights();









    return 0;
}
```

b. **Funktion ergänzen** (5 Punkte). Eine Funktion *setLights* soll nun alle Lampen aufs Mal ein- oder ausschalten, entsprechend dem Bitmuster, das als Parameter übergeben wird. Alle eingeschalteten Lampen leuchten in der gleichen Farbe (zweiter Parameter).

`void setLights(int data, char color[])`
Setzt die Lämpchen entsprechend dem Bitmuster in *data* auf die in *color* angegebene Farbe.

Beispiele:

`setLights (5, "green");`  (aus – grün – aus – grün)

`setLights (12, "orange");`  (orange – orange – aus – aus)

Ergänzen Sie die Funktionsdefinition:

```
void setLights (int data, char color[]) {
    int i;
    for (i=0; _____________________________________________) {

        if ( _____________________________________________) {
            switchLight(i, color);
        } else {
            switchLight(i, "off");
        }
    }
}
```

c. **Programm analysieren** (2 Punkte). Das folgende Programm verwendet die Funktion *setLights* aus Teil b. und die in Teil a. beschriebene Funktion *delay*:

```
int main (void) {
    initLights();

    int seq = 8;
    while (1) {
        setLights(seq, "red");
        delay(0.3);
        seq = seq >> 1;
        // (Teil d)
    }
    return 0;
}
```

Was wird angezeigt, wenn man das Programm laufen lässt?

d. **Programm anpassen** (2 Punkte). Was muss beim Kommentar in der *while*-Schleife in Aufgabenteil c. ergänzt werden, damit die Anzeigesequenz sich immer wiederholt?

C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$, ...);	function prototype
type name;	variable declaration
`int main(void) {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$, ...) {	function definition
declarations	local variable declarations
statements	
`return` *value*;	
`}`	
`/* */`	comments
`int main(int argc, char *argv[])`	main with args
`exit(`*arg*`);`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*`(`*var*`)` *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
Example. `#define msg(A) printf("%s = %d", #A, (A))`	
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	`\`

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
real number (single, double precision)	`float, double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
double long (64 bit integer)	`long long`
positive or negative	`signed`
non-negative modulo 2^m	`unsigned`
pointer to `int`, `float`,...	`int*, float*`,...
enumeration constant	`enum` *tag* {*name*$_1$`=`*value*$_1$,...};
constant (read-only) value	*type* `const` *name*;
declare external variable	`extern`
internal to source file	`static`
local persistent between calls	`static`
no value	`void`
structure	`struct` *tag* {...};
create new name for data type	`typedef` *type name*;
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type*`)`

Initialization

initialize variable	*type name*`=`*value*;
initialize array	*type name*`[]={`*value*$_1$,...};
initialize char string	`char` *name*`[]="`*string*`"`;

Constants

suffix: long, unsigned, float	`65536L, -1U, 3.0F`
exponential form	`4.2e1`
prefix: octal, hexadecimal	`0, 0x` or `0X`
Example. `031` is 25, `0x31` is 49 decimal	
character constant (char, octal, hex)	`'a', '\`*ooo*`', '\x`*hh*`'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*;
declare function returning pointer to *type*	*type* `*f();`
declare pointer to function returning *type*	*type* `(*pf)();`
generic pointer type	`void *`
null pointer constant	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]`...

Structures

`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	

create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed-to structure	*pointer* `->` *member*
Example. `(*p).x` and `p->x` are the same	
single object, multiple possible types	`union`
bit field with *b* bits	`unsigned` *member*`:` *b*;

Operators (grouped by precedence)

struct member operator	*name*`.`*member*
struct member through pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
relational comparisons	`>, >=, <, <=`
equality comparisons	`==, !=`
and [bit op]	`&`
exclusive or [bit op]	`^`
or (inclusive) [bit op]	`|`
logical and	`&&`
logical or	`||`
conditional expression	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
assignment operators	`+=, -=, *=,` ...
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Flow of Control

statement terminator	`;`
block delimiters	`{ }`
exit from `switch`, `while`, `do`, `for`	`break;`
next iteration of `while`, `do`, `for`	`continue;`
go to	`goto` *label*;
label	*label*`:` `statement`
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*$_1$`)` *statement*$_1$ `else if (`*expr*$_2$`)` *statement*$_2$ `else` *statement*$_3$
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *statement*$_1$ `break;`
	`case` *const*$_2$`:` *statement*$_2$ `break;`
	`default:` *statement*
	`}`

ANSI Standard Libraries

`<assert.h>`	`<ctype.h>`	`<errno.h>`	`<float.h>`	`<limits.h>`
`<locale.h>`	`<math.h>`	`<setjmp.h>`	`<signal.h>`	`<stdarg.h>`
`<stddef.h>`	`<stdio.h>`	`<stdlib.h>`	`<string.h>`	`<time.h>`

Character Class Tests `<ctype.h>`

alphanumeric?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`
convert to lower case	`tolower(c)`
convert to upper case	`toupper(c)`

String Operations `<string.h>`

`s` is a string; `cs`, `ct` are constant strings

length of `s`	`strlen(s)`
copy `ct` to `s`	`strcpy(s,ct)`
concatenate `ct` after `s`	`strcat(s,ct)`
compare `cs` to `ct`	`strcmp(cs,ct)`
only first `n` chars	`strncmp(cs,ct,n)`
pointer to first `c` in `cs`	`strchr(cs,c)`
pointer to last `c` in `cs`	`strrchr(cs,c)`
copy `n` chars from `ct` to `s`	`memcpy(s,ct,n)`
copy `n` chars from `ct` to `s` (may overlap)	`memmove(s,ct,n)`
compare `n` chars of `cs` with `ct`	`memcmp(cs,ct,n)`
pointer to first `c` in first `n` chars of `cs`	`memchr(cs,c,n)`
put `c` into first `n` chars of `s`	`memset(s,c,n)`

C Reference Card (ANSI)

Input/Output <stdio.h>

Standard I/O

standard input stream	stdin
standard output stream	stdout
standard error stream	stderr
end of file (type is int)	EOF
get a character	getchar()
print a character	putchar(*chr*)
print formatted data	printf("*format*",arg_1,...)
print to string s	sprintf(s,"*format*",arg_1,...)
read formatted data	scanf("*format*",&$name_1$,...)
read from string s	sscanf(s,"*format*",&$name_1$,...)
print string s	puts(s)

File I/O

declare file pointer	FILE *fp;
pointer to named file	fopen("*name*","*mode*")
modes: r (read), w (write), a (append), b (binary)	
get a character	getc(*fp*)
write a character	putc(*chr*,*fp*)
write to file	fprintf(*fp*,"*format*",arg_1,...)
read from file	fscanf(*fp*,"*format*",arg_1,...)
read and store n elts to *ptr	fread(*ptr,eltsize,n,*fp*)
write n elts from *ptr to file	fwrite(*ptr,eltsize,n,*fp*)
close file	fclose(*fp*)
non-zero if error	ferror(*fp*)
non-zero if already reached EOF	feof(*fp*)
read line to string s (< max chars)	fgets(s,max,*fp*)
write string s	fputs(s,*fp*)

Codes for Formatted I/O: "%-+ 0*w.pmc*"

-	left justify
+	print with sign
space	print space if no sign
0	pad with leading zeros
w	min field width
p	precision
m	conversion character: h short, l long, L long double
c	conversion character:

d,i	integer	u	unsigned
c	single char	s	char string
f	double (printf)	e,E	exponential
f	float (scanf)	lf	double (scanf)
o	octal	x,X	hexadecimal
p	pointer	n	number of chars written
g,G	same as f or e,E depending on exponent		

Variable Argument Lists <stdarg.h>

declaration of pointer to arguments	va_list *ap*;
initialization of argument pointer	va_start(*ap*,*lastarg*);
lastarg is last named parameter of the function	
access next unnamed arg, update pointer	va_arg(*ap*,*type*)
call before exiting function	va_end(*ap*);

Standard Utility Functions <stdlib.h>

absolute value of int n	abs(n)
absolute value of long n	labs(n)
quotient and remainder of ints n,d	div(n,d)
returns structure with div_t.quot and div_t.rem	
quotient and remainder of longs n,d	ldiv(n,d)
returns structure with ldiv_t.quot and ldiv_t.rem	
pseudo-random integer [0,RAND_MAX]	rand()
set random seed to n	srand(n)
terminate program execution	exit(status)
pass string s to system for execution	system(s)

Conversions

convert string s to double	atof(s)
convert string s to integer	atoi(s)
convert string s to long	atol(s)
convert prefix of s to double	strtod(s,&endp)
convert prefix of s (base b) to long	strtol(s,&endp,b)
same, but unsigned long	strtoul(s,&endp,b)

Storage Allocation

allocate storage	malloc(size), calloc(nobj,size)
change size of storage	newptr = realloc(ptr,size);
deallocate storage	free(ptr);

Array Functions

search array for key	bsearch(key,array,n,size,cmpf)
sort array ascending order	qsort(array,n,size,cmpf)

Time and Date Functions <time.h>

processor time used by program	clock()
Example. clock()/CLOCKS_PER_SEC is time in seconds	
current calendar time	time()
$time_2$-$time_1$ in seconds (double)	difftime($time_2$,$time_1$)
arithmetic types representing times	clock_t,time_t
structure type for calendar time comps	struct tm

tm_sec	seconds after minute
tm_min	minutes after hour
tm_hour	hours since midnight
tm_mday	day of month
tm_mon	months since January
tm_year	years since 1900
tm_wday	days since Sunday
tm_yday	days since January 1
tm_isdst	Daylight Savings Time flag

convert local time to calendar time	mktime(tp)
convert time in tp to string	asctime(tp)
convert calendar time in tp to local time	ctime(tp)
convert calendar time to GMT	gmtime(tp)
convert calendar time to local time	localtime(tp)
format date and time info	strftime(s,smax,"*format*",tp)
tp is a pointer to a structure of type tm	

Mathematical Functions <math.h>

Arguments and returned values are double

trig functions	sin(x), cos(x), tan(x)
inverse trig functions	asin(x), acos(x), atan(x)
arctan(y/x)	atan2(y,x)
hyperbolic trig functions	sinh(x), cosh(x), tanh(x)
exponentials & logs	exp(x), log(x), log10(x)
exponentials & logs (2 power)	ldexp(x,n), frexp(x,&e)
division & remainder	modf(x,ip), fmod(x,y)
powers	pow(x,y), sqrt(x)
rounding	ceil(x), floor(x), fabs(x)

Integer Type Limits <limits.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system, followed by minimum required values (if significantly different).

CHAR_BIT	bits in char	(8)
CHAR_MAX	max value of char	(SCHAR_MAX or UCHAR_MAX)
CHAR_MIN	min value of char	(SCHAR_MIN or 0)
SCHAR_MAX	max signed char	(+127)
SCHAR_MIN	min signed char	(−128)
SHRT_MAX	max value of short	(+32,767)
SHRT_MIN	min value of short	(−32,768)
INT_MAX	max value of int	(+2,147,483,647) (+32,767)
INT_MIN	min value of int	(−2,147,483,648) (−32,767)
LONG_MAX	max value of long	(+2,147,483,647)
LONG_MIN	min value of long	(−2,147,483,648)
UCHAR_MAX	max unsigned char	(255)
USHRT_MAX	max unsigned short	(65,535)
UINT_MAX	max unsigned int	(4,294,967,295) (65,535)
ULONG_MAX	max unsigned long	(4,294,967,295)

Float Type Limits <float.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system.

FLT_RADIX	radix of exponent rep	(2)
FLT_ROUNDS	floating point rounding mode	
FLT_DIG	decimal digits of precision	(6)
FLT_EPSILON	smallest x so $1.0\text{f} + x \neq 1.0\text{f}$	($1.1\text{E}-7$)
FLT_MANT_DIG	number of digits in mantissa	
FLT_MAX	maximum float number	($3.4\text{E}38$)
FLT_MAX_EXP	maximum exponent	
FLT_MIN	minimum float number	($1.2\text{E}-38$)
FLT_MIN_EXP	minimum exponent	
DBL_DIG	decimal digits of precision	(15)
DBL_EPSILON	smallest x so $1.0 + x \neq 1.0$	($2.2\text{E}-16$)
DBL_MANT_DIG	number of digits in mantissa	
DBL_MAX	max double number	($1.8\text{E}308$)
DBL_MAX_EXP	maximum exponent	
DBL_MIN	min double number	($2.2\text{E}-308$)
DBL_MIN_EXP	minimum exponent	

Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. ⟨jhs@math.brown.edu⟩