Toetsvoorblad

DE HAAGSE HOGESCHOOL
FACULTEIT TECHNOLOGIE, INNOVATIE & SAMENLEVING
Locatie: Delft

Naam Student: _____

Studentnummer: _____

Opleiding: **Elektrotechniek**		**Toetsnaam:** **GESPRG**	
Opsteller:	J.E.J. op den Brouw	Datum:	1 januari 1970
Tweede lezer:	B. Kuiper	Tijd:	0:00 – 1:30
Groep:	EP1	Aantal bladzijden:	8 (inclusief voorblad)
Cursuscode:	E-GESPRG-co1	Aantal vragen:	6

Bij deze toets worden verstrekt:

☒ Gelinieerd papier
☐ Ruitjes papier
☒ Kladpapier
☐ Omslag voor gemaakt tentamen
☐ Overig: _____
☒ Bijlage(n): C reference card
☐ Opgavenbladen met ruimte om de vragen te beantwoorden
☐ Antwoordformulier ABCDE
☐ Antwoordformulier Ja/Nee
☐ Antwoordformulier Ja/Nee/Vraagteken

Toegestane eigen hulpmiddelen bij het maken van deze toets:

☒ Eenvoudige rekenmachine
☒ Grafische rekenmachine
☐ Computer
☐ Formuleblad(en): _____
☒ Tekenbenodigdheden (liniaal, passer)
☐ Eigen aantekeningen: _____
☐ Boeken/dictaten: _____

Opmerkingen:

Bij deze toets mag geen aanvullende documentatie gebruikt worden

Cesuur (voorlopig):

De cesuur is 45 punten

In te leveren door surveillant bij het faculteitsbureau:

☒ Alle documenten voorzien van naam en studentnummer, per document gesorteerd
☐ Alle documenten voorzien van naam en studentnummer, per student gesorteerd (in omslag)

Belangrijk:

Voor dit tentamen gelden de regels uit de toetsregeling van De Onderwijs- en Examenregeling. Dit document is aanwezig in het toetslokaal;
Je dient zelf te controleren of je alle pagina's en vragen van dit tentamen hebt ontvangen;
Dit tentamen is dubbelzijdig geprint;
Schrijf je naam en studentnummer op alle documenten.

Let op:

- de toets bestaat uit 6 vragen met in totaal 4 deelvragen.
- laat bij het beantwoorden van de vragen de uitwerking of motivatie zien, antwoorden zonder uitwerking of motivatie leveren geen punten op.
- als je een vraag niet (geheel) snapt, geef dan op papier aan hoe je de vraag interpreteert.
- bij de gegeven programmacode zijn regelnummers gezet, deze zijn geen onderdeel van de code.
- er zijn maximaal 90 punten te behalen, eindcijfer $= 1{,}0 + \dfrac{\text{aantal behaalde punten}}{10}$

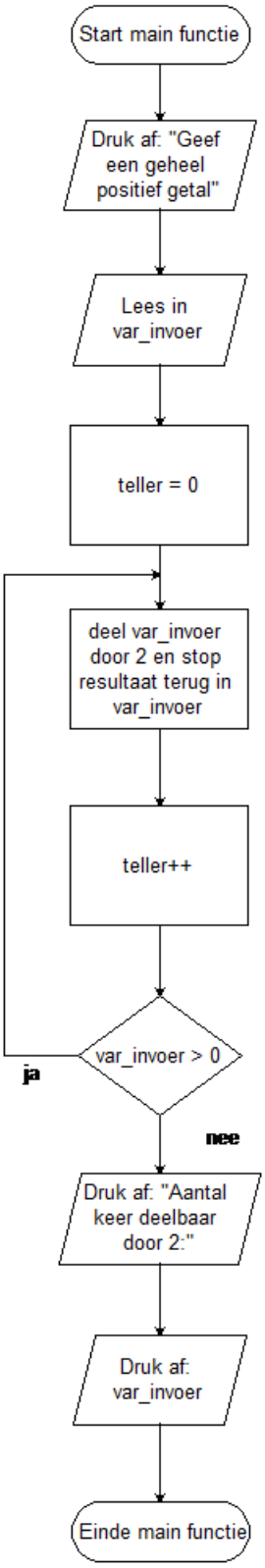
Opgave 1 (8 pt)
Gegeven is de volgende code:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i;
6      printf("%d ", i);
7      for (i = 8; i > -1; i = i - 2){
8          printf("%d ", i);
9      }
10     printf("%d ", i);
11 }
```

Listing 1: *Programma bij opgave 1.*

Geef de uitvoer van het bovenstaande programma. Motiveer daarnaast ook je antwoord.

Opgave 2 (15 pt)
Gegeven is de flowchart in figuur 1.



Figuur 1: *Flowchart bij opgave 2.*

Geef de volledige C-code die hoort bij het gegeven flowchart.

Opgave 3 (13 pt)
Gegeven is de volgende code:

```
1  #include <stdio.h>
2
3  int f(int i) {
4      i = i + 1;
5      return i;
6  }
7  int g(int *p) {
8      *p = *p + 1;
9      return *p;
10 }
11 int main(void)
12 {
13     int j = 0;
14     printf("%d, ", g(&j));
15     printf("%d, ", f(j));
16     printf("%d\n", g(&j));
17 }
```

Listing 2: *Programma bij opgave 3.*

Geef de uitvoer van het bovenstaande programma. Motiveer daarnaast ook je antwoord.

Opgave 4 (20 pt)
In deze opgave moet een functie `is_spiegelwoord` worden geschreven die bepaalt of een C-string een spiegelwoord is.

Wij definiëren een spiegelwoord als een woord dat alleen uit kleine letters bestaat en dat van links naar rechts gelezen hetzelfde is als van rechts naar links gelezen. Voorbeelden van spiegelwoorden in de Nederlandse taal zijn lepel, droomoord, negen, lol, soos en meetsysteem. Voorbeelden van woorden die geen spiegelwoorden zijn: Lol (omdat er een hoofdletter in voorkomt) en 73neen37 (omdat er cijfers in voorkomen). Het prototype van de functie is als volgt:

```
int is_spiegelwoord(char woord[]);
```

De functie moet de waarde 1 teruggeven als de meegegeven C-string een spiegelwoord is en moet de waarde 0 teruggeven als de meegegeven C-string geen spiegelwoord is.

Enkele tips:

- Je kunt de functie `strlen` die gedeclareerd is in `string.h` gebruiken om het aantal karakters van het als argument meegegeven woord te vinden.
- Je kunt de functie `islower` die gedeclareerd is in `ctype.h` gebruiken om te controleren of een karakter een kleine letter is.
- Je hoeft niet te controleren of het woord voorkomt in de Nederlandse taal. Het woord acjsheehsjca is dus een spiegelwoord.
- De functie `strrev` mag NIET gebruikt worden.

Geef de volledige code van de functie `is_spiegelwoord`.

Opgave 5 (20 pt)
Gegeven is de onderstaande C code. Deze code gaat over een gedeelte van een treinregistratiesysteem.

```
1  #include <stdio.h>
2
3  #define AANTAL_WAGONS 4
4
5  typedef struct{
6      int wagonNr;
7      int lengte;
8  } WagonGegevens;
9
10     /* Op deze plaats moet code komen te staan*/
11
12 int main()
13 {
14     WagonGegevens trein[AANTAL_WAGONS];
15     int i;
16
17     for(i=0; i<AANTAL_WAGONS; i++){
18         printf("Geef wagon nr: ");
19         scanf("%d",&trein[i].wagonNr);
20         printf("Geef lengte: ");
21         scanf("%d",&trein[i].lengte);
22     }
23
24     /* Op deze plaats moet code komen te staan*/
25 }
```

Listing 3: *Programma bij opgave 5.*

a) Hoeveel variabelen van het type **int** zitten in de array `trein`? Licht je antwoord toe (6 pt).

b) Op regel 24 moet code komen te staan die de totale lengte print van de trein die is ingevoerd. Geef de benodigde code. Zorg dat er gebruik wordt gemaakt van een functie die de totale lengte berekent en terug geeft. Deze functie moet op regel 10 komen te staan (14 pt).

Tip: een trein bestaat uit een aantal wagons en de totale lengte van de trein is dus de lengte van alle wagons bij elkaar opgeteld.

Opgave 6 (14 pt)
Gegeven is de onderstaande C code.

```
1  int main()
2  {
3      double *pt1;
4      double *pt2;
5
6      double vara = 4.0;  /* variabele vara staat op adres 00100010 */
7      /* begin-adres van onderstaande array is 00100020 */
8      double arrayA[] = { 1.0,2.0,3.0,4.0 };
9
10     pt1 = &vara;
11     *pt1 = 5.0;
12     pt2 = arrayA;
13     *pt2 = 6.0;
14     ++pt2;
15     *pt2 = 7.0;
16
17     return 0;
18 }
```

Listing 4: *Programma bij opgave 6.*

a) Geef aan wat de inhoud is van de variabelen `pt1`, `pt2` en `vara` ná het uitvoeren van coderegels 1 tot en met 15 (7 pt).

b) Geef aan wat de inhoud is van alle variabelen die in `arrayA` zitten ná het uitvoeren van coderegels 1 tot en met 15 (7 pt).

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 1.0f + $x \neq$ 1.0f	(1.1E − 7)
FLT_MANT_DIG	number of digits in mantissa	
FLT_MAX	maximum float number	(3.4E38)
FLT_MAX_EXP	maximum exponent	
FLT_MIN	minimum float number	(1.2E − 38)
FLT_MIN_EXP	minimum exponent	
DBL_DIG	decimal digits of precision	(15)
DBL_EPSILON	smallest x so 1.0 + $x \neq$ 1.0	(2.2E − 16)
DBL_MANT_DIG	number of digits in mantissa	
DBL_MAX	max double number	(1.8E308)
DBL_MAX_EXP	maximum exponent	
DBL_MIN	min double number	(2.2E − 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⟩