KLAUSUR im FACH ***Ingenieur-Informatik*** im Wintersemester 2021/2022

Name, Vorname: …….

Studiengang/Semester: ……

Prüfer: Dipl.-Ing. (FH) Sebastian Balz, Dipl.-Ing. (FH) Sebastian Stelzl, Andreas Behr M.Sc.

Bearbeitungshinweise

1. Tragen Sie auf jeder Seite in der Kopfzeile Ihre Matrikelnummer ein.

2. Der Aufgabensatz (inkl. Deckblatt und Anhang), der aus 19 Seiten besteht (Seite 1 bis 19), ist auf Vollständigkeit zu überprüfen.

3. Der Aufgabensatz ist mit den Lösungsblättern abzugeben.

4. Lösungen auf selbst mitgebrachten Lösungsblättern werden nicht ausgewertet. Verwenden Sie die Ihnen ausgeteilten Lösungsblätter und tragen Sie auch dort Ihre Matrikelnummer ein.

5. Bei Rechenaufgaben muss der Lösungsweg ersichtlich und lesbar sein, sonst erfolgt keine Bewertung der Aufgabe oder des Aufgabenteils.

6. Die Bearbeitungszeit beträgt 90 Minuten.

7. Es wird hiermit darauf hingewiesen, dass vom Prüfungsamt nicht vorher geprüft wurde, ob Sie das Recht bzw. die Pflicht zur Teilnahme an dieser Klausur haben. Die Teilnahme erfolgt auf eigene Gefahr, gleichzeitig bekundet die Teilnahme die Zustimmung zu diesem Passus.

8. Hilfsmittel:
   - C-Coding Styleguide (ist selbst mitzubringen) – nur Markierungen mit einem Marker sind erlaubt (keine eigenen Notizen)
   - ASCII-Tabelle und Kurzreferenz Bibliotheksfunktionen (wird ausgeteilt)
   - Taschenrechner sind **nicht** erlaubt – auch **keine** Smartphones

9. **Die Nichteinhaltung des C-Coding Styleguides führt zu Punktabzug**

10. Bewertung:
    Gesamtpunktzahl: 100 Punkte
    Note 1,0: 90 Punkte Note 4,0: 45 Punkte

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**SUMME**	
Punkte	**10**	**10**	**20**	**20**	**10**	**20**	**10**	**100**	**NOTE**
Erreichte Punkte									

Aufgabe 1: **(10 x 1 Punkt)**

1.1 Wie viele Werte können mit einem 8-Bit Datentyp abgebildet werden? _____

1.2 Wie berechnet sich die maximale Anzahl der Werte, welche mit einem Datentyp mit n Bits abgebildet werden können? (Formel) Anzahl Werte = _____

1.3 Mit welchem Wert wird die Variable `static int i;` initialisiert? _____

1.4 Wo werden lokale Variablen im Speicher abgelegt? _____

1.5 Nennen Sie eine Funktion aus der Utilities.lib? _____

1.6 Wann spricht man von einem Fall-through?

_____

1.7 Nennen Sie ein C-Keyword für eine kopfgesteuerte Schleife: _____

1.8 Definieren Sie ein Ganzzahlarray mit zwei Elementen und initialisieren diese mit den Werten "1":

_____;

1.9 Was wäre eine bessere Alternative für `#define PI 3.1415`?

_____;

1.10 Wozu dient das &-Zeichen beim Aufruf von `scanf_s("%d", &iVal);`?

_____

_____

Aufgabe 2: (10 Punkte)

Schreiben Sie hinter die Kommentare, was ***exakt*** in der Konsole auf einem **32-Bit System (x86)** ausgegeben wird**.** Im Kommentar sind die möglichen Punkte aufgeführt.
Hinweise:
Hexadezimale Ziffer A-F
werden bei printf mit dem Formattierer **%x** als **Kleinbuchstaben** ausgegeben.
Führende Nullen werden durch %x auch **unterdrückt.**

Beachten Sie auch die **Padding**-Bytes.

```
//myvars.h
#pragma once

enum Language {German, English};

struct Book
{
   char acAuthor[64];
   char acTitle[64];
   char acISBN[32];
   enum Language eLanguage;
};
typedef struct Book sBook_t;
typedef sBook_t* psBook_t;

struct LibraryCard
{
   char acOwner[200];
   unsigned int uiNumber;
   sBook_t asBooks[5];
};
typedef struct LibraryCard sLibraryCard_t;
typedef sLibraryCard_t* psLibraryCard_t;
```

```
void AG2(void)
{
    sBook_t sBook = {"Douglas A.", "Hitchhiker's Guide", "978-0345391803", English};
    psBook_t psBook = &sBook;
    sLibraryCard_t sLibraryCard = {"Lee, Bruce", 123456U};
    psLibraryCard_t psLibraryCard = &sLibraryCard;

    psLibraryCard->asBooks[0] = *psBook;

    printf("%u\n", sizeof(sBook_t));                    // _________________ 1P
    printf("%u\n", sizeof(psBook));                     // _________________ 1P
    printf("%u\n", sizeof(sLibraryCard_t));             // _________________ 1P
    printf("%u\n", sizeof(psLibraryCard));              // _________________ 1P

    printf("%d\n", sLibraryCard.asBooks[0].eLanguage);  // _________________ 1P
    printf("%x\n", sLibraryCard.asBooks[3]);            // _________________ 1P

    printf("%d\n", strlen(psLibraryCard->acOwner));     // _________________ 1P

    char* pc = &sLibraryCard.acOwner[5];
    strncpy_s(sLibraryCard.acOwner, 200, pc, 5);
    printf("%s\n", sLibraryCard.acOwner);               // _________________ 1P

    unsigned char uc1 = 0xF0;
    unsigned char uc2 = 0x0F;
    printf("%x\n", uc1 | uc2);                          // _________________ 1P
    printf("%x\n", (uc1 >> 2U) ^ (uc2 << 2U));          // _________________ 1P
}
```

Aufgabe 3: **(20 Punkte)**

3.1 Wandeln Sie die folgenden Zahlen in das andere Zahlensystem um. (3 Punkte)

123_7 = _____$_{10}$ (1 P)

$0xFACE_{16}$ = _____$_2$ (0,5 P)

33_{15} = _____$_{10}$ = _____$_{40}$ (1 + 0,5 P)

3.2 Wie wird in einem Struktogramm (Nassi-Shneiderman-Diagramm) eine *Mehrfache Selektion* (switch) mit drei Fällen (case) und einem default abgebildet? (2 Punkte)

3.3 Implementieren Sie eine Inline-Funktion, welche die Länge einer Zeichenkette **iterativ** bestimmt und zurückgibt. Bibliotheksfunktionen dürfen nicht verwendet werden. (6 Punkte)

```
__________ unsigned int strlen2(char* pc)
{




}
```

3.4 Implementieren Sie ein Funktionsmakro PRINTINTVAR, welches die Adresse und den Wert einer Ganzzahlvariablen ausgibt. (3 Punkte)

Beispiel:

```
int i = 42;
PRINTINTVAR(i); // Gibt z.B. aus: “AFAD20 | 42”, AFAD20 sei dabei die Adresse
```

3.4 Worin unterscheidet sich ein *Implicit Cast* von einem *Explicit Cast*? (1 Punkt)

3.5 Deklarieren Sie einen komplexen Datentyp mit dessen Hilfe eine Ganzzahl als `int`-Variable **oder** als Zeichenkette mit zehn Elementen abgelegt werden kann. Dieser komplexe Datentyp soll minimalen Speicher belegen. (1 Punkt)

3.5.1 Definieren Sie dynamisch eine Variable des oben deklarierten Datentyps. (2 Punkte)

3.5.2 Schreiben Sie „73“ in die Zeichenkette der Variablen aus 3.5.1. (1 Punkt)

3.5.3 Weisen Sie der Ganzzahl-Variablen aus 3.5.1 den Wert aus der Zeichenkette zu. Verwenden Sie dazu eine passende Konvertierungsfunktion. (1 Punkt)

Aufgabe 4: **(20 Punkte)**

Gegeben sei die einfach verkettete Liste aus der Vorlesung. Implementieren Sie die Funktion **InsertElementFront**.

Fügen Sie auch zwei Sanity-Checks ein.
Wenn alles korrekt war, wird eine 0 zurückgegeben. Berücksichtigen Sie auch die Fallunterscheidung: Liste ist leer oder nicht leer!

```
struct Node
{
   unsigned int uiData;
   struct Node* psNextNode;
};
typedef struct Node sNode_t;
typedef sNode_t* psNode_t;
typedef psNode_t* ppsNode_t;
```

```
int InsertElementFront(ppsNode_t ppsAnchor, psNode_t psNewNodePassed)
{
   int iRet = -1;
   psNode_t psNewNode;

   return iRet;
}
```

Aufgabe 5: **(10 Punkte)**

Implementieren Sie die **rekursive** Funktion GetModuloRecursiv, welche die Operation iA % iB nachbildet. Der Modulooperator % darf nicht verwendet werden.

Tipp: Überlegen Sie sich zuerst das Abbruchkritierium!

```
unsigned int GetModuloRecursiv(unsigned int uiA, unsigned int uiB)
{
   unsigned int uiRet;









   return uiRet;
}
```

Aufgabe 6: **(20 Punkte)**

Das Programm "ReadFile.exe", zum Einlesen einer Textdatei wird über die Konsole gestartet und bekommet u.a. einen Dateinamen sowie den Pfad beim Aufruf in **einem** Parameter übergeben. Der Aufruf des Programmes sieht z.B. wie folgt aus (mit zwei weiteren Übergabeparametern):

```
C:\Temp>ReadFile.exe -file=C:\Temp\Dokument.txt Hello 42
```

```
int main(int argc, char* argv[])
{
    char acFileName[256] = {'\0'};
    int iRet = GetFileName(argc, argv, acFileName);

    if (iRet == 1)
    {
        ReadFile(acFileName);
    }
    return 0;
}
```

Implementieren Sie die Funktion `GetFileName`, welche anhand des Kenners "`-file=`" den zu verwendenden Pfad + Dateinamen aus **allen** Übergabeparametern ermittelt und in acFileName zurückgibt. Wurde eine Dateiangabe gefunden(`-file=`) so gibt die Funktion eine "1" zurück, ansonsten "-1".

Der Übergabeparameter mit "`-file=`" muss nicht unbedingt der zweite Übergabeparameter sein. Wird ein solcher Übergabeparameter gefunden, so sind die weiteren Übergabeparameter nicht mehr auszuwerten.

Hinweis: argv darf nicht verändert werden.

Lösung bitte auf dem ausgeteilten Lösungsbogen.

Aufgabe 7: **(10 Punkte)**

Ein Programm berechnet in einer Thread-Funktion die **Summe** aller möglichen Vielfachen einer Zahl `uiNumber` für einen beliebigen Bereich von 0 – `uiRangeEnd` und gibt das Ergebnis in `uiSum` zurück.

Beispiel: Vielfache von 5 im Bereich von 0 – 20

- ⇨ `uiNumber`=5, `uiRangeEnd`=20
- ⇨ Vielfache: 5, 10, 15, 20
- ⇨ Summe der Vielfachen: 5 + 10 + 15 + 20 = 50 = `uiSum`

```
struct Conditions
{
    unsigned int uiNumber;
    unsigned int uiRangeEnd;
    unsigned int uiSum;
};
typedef struct Conditions sConditions_t;
typedef sConditions_t* psConditions_t;

int main(int argc, char* argv[])
{
    sConditions_t sCond = { 5U, 20U, 0U };
    HANDLE hThread = CreateThread(NULL, 0, SumOfMultiple, &sCond, 0, NULL);
    if (hThread != 0)
    {
        WaitForSingleObject(hThread, INFINITE);
        printf("The sum of multiples of %u between 0 and %u is: %u\n",
               sCond.uiNumber, sCond.uiRangeEnd, sCond.uiSum);
        CloseHandle(hThread);
    }
    return 0;
}
```

Implementieren Sie die Thread-Funktion `SumOfMultiple` in C.

```
_____________________ SumOfMultiple(_____________________________)
{









}
```

Anhang A: ASCII-Tabelle

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

Anhang B: Codierung

dezimal	hexadezimal	binär
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Anhang C: Standardbibliotheken

ctype.h

```
int isalnum(int iX);
int isalpha(int iX);
int iscntrl(int iX);
int isdigit(int iX);
int isgraph(int iX);
int islower(int iX);
int isprint(int iX);
int ispunct(int iX);
int isspace(int iX);
int isupper(int iX);
int isxdigit(int iX);
int tolower(int iX);
int toupper(int iX);
```

string.h

```
void* memcpy(void* pvS1, const void* pvS2, size_t uiN);
void* memmove(void* pvS1, const void* pvS2, size_t uiN);
char* strcpy(char* pcS1, char* pcS2);
errno_t strcpy_s (char* pcDest, rsize_t uSize, const char* pcSrc); // C11
char* strncpy(char* pcS1, char* pcS2, size_t uiN);
errno_t strncpy_s(char* dest, rsize_t destsz, const char* src, rsize_t count); // C11
char* strcat(char* pcS1, const char* pcS2);
char* strncat(char* pcS1, const char* pcS2, size_t uiN);
int memcmp(const void* pvS1, const void* pvS2, size_t uiN);
int strcmp(const char* pcS1, const char* pcS2);
int strcnmp(const char* pcS1, const char* pcS2, size_t uiN);
void* memchr(const void* pvS, int iC, size_t uiN);
char* strchr(const char* pcS, int c);
size_t strcspn(const char* pcS1, char* pcS2);
char* strpbrk(const char* pcS1, const char* pcS2);
char* strrchr(const char* pcS, int iX);
size_t strspn(const char* pcS1, const char* pcS2);
const char* strstr(const char* pcS1, const char* pcS2);
void* memset(void* pvM, int iC, size_t uiN);
size_t strlen(const char* pcS);

char* strtok(char* pcStr, const char* pccDelimiters);
char* strtok_s(char* pcStr, const char* pccDelimiters, char** ppcContext); // C11
char* strlwr(char* pcStr); // converts string to lowercase – no C Standard
char* strupr(char* pcStr); // converts string to uppercase – no C Standard
```

stdio.h

```
int fflush(FILE* pfFile);
size_t fread(void* pvData, size_t uiSize, size_t uiNumber, FILE* pfFile);
size_t fwrite(void* pvData, size_t uiSize, size_t uiNumber, FILE* pfFile);
int fseek(FILE* pfFile, long lOffset, int iPos);
long ftell(FILE* pfFile);
void rewind(FILE* pfFile);
int feof(FILE* pfFile); //return not 0 if EOF is reached
int fputs(const char* pccStr, FILE* pfFile); //return EOF if error, otherwise non-zero
char* fgets(char* pcStr, int num, FILE* pfFile); //CRLF is part of pcStr
int rename(char* pcFilenameOld, char* pcFilenameNew); // return 0 if success
int remove(const char* pccFileName); // return 0 if success

FILE* fopen(const char* pccFilename, const char* pccModus);
errno_t fopen_s (FILE** ppfFile, const char* pccFilename,  const char* pccModus); // C11
int fclose(FILE* pfFile); //0 if okay, EOF if Error
int printf (const char * pccFormat, ... );
int sprintf (char* pcStr, const char * pccFormat, ... );
int fprintf (FILE* pfFile, const char * pccFormat, ... );
int scanf (const char*pccFormat, ... );
int scanf_s(const char * pccFormat, ...); // Zusätzlicher Wert bei %c und %s notwendig -
                                          Anzahl Zeichen  (C11)
int sscanf (char* pcStr, const char* pccFormat, ... )
```

math.h

```
double acos(double dX);
double asin(double dX);
double atan(double dX);
double atan2(double dX, double dY);
double cos(double dX);
double sin(double dX);
double tan(double dX);
double cosh(double dX);
double sinh(double dX);
double tanh(double dX);
double exp(double dX);
double log(double dX);
double log10(double dX);
double pow(double dX, double dY); //x^y
double sqrt(double dX);
double ceil(double dX); // Ganzzahliger Wert durch Aufrunden
double floor(double dX); // Ganzzahliger Wert durch Abrunden
double fmod(double dX, double dY); // Rest der (ganzzahligen) Division der beiden Param.

double fabs (double dX);              // C99
float fabsf (float dX);               // C99
long double fabsl (long double dX); // C99
```

stdlib.h

```
double atof(const char* pccValue);
int atoi(const char* pccValue);
long atol(const char* pccValue);
double strtod(const char* pccValue, char** ppcEndConversion);
long int strtol(const char* pccValue, char** ppcEndConversion, int iBase);
unsigned long int strtoul(const char* pccValue, char** ppcEndConversion, int iBase);
int rand(void);
void srand(unsigned int uiStartValue);
int abs(int iValue);
int labs(long int liValue);
char* itoa(int iValue, char* pcStr, int iBase);
void* malloc(size_t uiSize);
void free (void* pvMem);
```

time.h

Datenstrukturen

```
struct tm
{
   int tm_sec;         /* seconds,  range 0 to 59          */
   int tm_min;         /* minutes, range 0 to 59           */
   int tm_hour;        /* hours, range 0 to 23             */
   int tm_mday;        /* day of the month, range 1 to 31  */
   int tm_mon;         /* month, range 0 to 11             */
   int tm_year;        /* The number of years since 1900   */
   int tm_wday;        /* day of the week, range 0 to 6    */
   int tm_yday;        /* day in the year, range 0 to 365  */
   int tm_isdst;       /* daylight saving time             */
};
typedef long int clock_t;
typedef long int time_t;
```

```
char* asctime(const struct tm* pcsTime);
time_t mktime(struct tm* psTime);
struct tm* localtime(const time_t* pcxTime);
clock_t clock(void);
time_t time(time_t* pxTime);
char* ctime(const time_t* pcxTime);

double difftime(time_t xEndtime, time_t Begintime);
```

Anhang D: Beispielcode zu Threads

Starten eines Threads per Windows Thread API

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  SIZE_T                  dwStackSize,
  LPTHREAD_START_ROUTINE  lpStartAddress,
  __drv_aliasesMem LPVOID lpParameter,
  DWORD                   dwCreationFlags,
  LPDWORD                 lpThreadId
);
```

Wichtige Parameter:

lpStartAddress: Funktionszeiger auf eine Thread-Funktion.
Deklaration einer Thread-Funktion: `DWORD WINAPI ThreadFunction(void *pParam);`

lpParameter: void-Zeiger auf die Parameter

Beispiel: Starten eines Threads ohne Parameterübergabe:

```
int main(void)
{
   HANDLE hThread1 = 0;

   // Beispiel ohne Parameterübergabe an Threadfunktion
   hThread1 = CreateThread(NULL, 0, ThreadFunction, NULL, 0, NULL);

   if (hThread1 != 0)
   {
      // Code
      CloseHandle(hThread1);
   }
   return 0;
}
```

Beispiel: Starten eines Threads mit Parameterübergabe:

```
int main(void)
{
   struct Data myData = {4711, "Hello"};

   // “&myData”: Impliziter Cast zu void*(Typenloser Zeiger)
   CreateThread(NULL, 0, ThreadFunction1, &myData, 0, NULL);
   return 0;
}

DWORD WINAPI ThreadFunction1(void *pParam)
{
   DWORD dwRet = 0;
   // Auf typenlosen Zeiger kann nicht zugegriffen werden -> „Rückcast“
   struct Data* psData = (struct Data*) pParam;
   // Code

   return 0;
}
```

Synchronisation von Threads

Warten auf einen Thread:

```
DWORD WaitForSingleObject(HANDLE hHandle,
                          DWORD dwMilliseconds );
```

Warten auf mehrere Threads:

```
DWORD WaitForMultipleObjects(DWORD nCount,
                             const HANDLE *lpHandles,
                             BOOL bWaitAll,
                             DWORD dwMilliseconds );
```

Wichtiger Parameter:
dwMilliseconds: Timeout in Millisekunden. INFINITE bedeutet keine Zeitbeschränkung.

Beispiel: Synchronisation von Threads – Hauptthread und zwei Threads

```
hThread1 = CreateThread(NULL, 0, ThreadFunction, &sData1, 0, NULL);
hThread2 = CreateThread(NULL, 0, ThreadFunction, &sData2, 0, NULL);
if ((hThread1 != 0) && (hThread2 != 0))
{
   ahThread[0] = hThread1;
   ahThread[1] = hThread2;

   // Hauptthread wartet auf hThread1 und hThread2 – ohne Timeout
   WaitForMultipleObjects(2, (const HANDLE*)&ahThread, TRUE, INFINITE);

   // Code
}
```

Anhang E: Beispielcode zu DLL

DLL Implementierung

Alle zu exportierenden Funktionen müssen mit __declspec(dllexport) gekennzeichnet sein.

Headerdatei (z.B. DLL.h):

```
#pragma once
__declspec(dllexport) void PrintFromDLL(char* pcText);
```

Sourcedatei:

```
#include "DLL.h"

__declspec(dllexport) void PrintFromDLL(char* pcText)
{
   //
}
```

Aufruf der DLL von einer Anwendung über Implicit Linking

```
#include <windows.h>
int main(void)
{
   HMODULE hModule;
   void (*fpDLLFunc)(char*);
   hModule = LoadLibrary(TEXT("DLL.dll"));
   if (hModule != 0)
   {
      fpDLLFunc = (void(*)(char*))GetProcAddress(hModule, "PrintFromDLL");
      if (fpDLLFunc != NULL)
      {
         fpDLLFunc("My First Call to a DLL");
      }
      FreeLibrary(hModule);
   }
   return 0;
}
```