KLAUSUR im FACH ***Ingenieur-Informatik*** im Wintersemester 2022/2023

Name, Vorname: …….

Studiengang/Semester: ……

Prüfer: Dipl.-OStR. Dipl.-Ing. (FH) Rüdiger Ehret, 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 17 Seiten besteht (Seite 1 bis 17), 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**	**SUMME**	
Punkte	**10**	**15**	**20**	**15**	**15**	**25**	**100**	**NOTE**
Erreichte Punkte								

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

1.1 Nennen Sie einen Datentyp, der erst mit C99 im Standard enthalten ist.

1.2 Kleinster Wert eines signed char:

1.3 Beispiel für symbolische Konstante:

1.4 Was ist ein C-Modul?

1.5 Nennen Sie zwei relationale Operatoren.

1.6 Beispiel für einen impliziten Cast, der zu einem falschen Wert führt:
Variablennamen nach dem C-Coding Styleguide verwenden.

1.7 Formatbezeichner zum Einlesen einer double-Variablen mit scanf_s:

1.8 Was bedeutet die Abkürzung (DRY):

1.9 Erklären Sie das DRY-Prinzip. Was soll damit verhindert werden?

1.10 Welche Bibliothek gehört nicht zum Standard, wird aber dennoch vom Compilerhersteller (Microsoft) mit MS Visual Studio installiert?

Aufgabe 2: (15 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 Formatierer **%x** als **Kleinbuchstaben** ausgegeben.
Führende Nullen werden durch %x auch **unterdrückt.**

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

```
//Header.h
#pragma once
struct Vehicle
{
   char cTyp; //'0' = car, '1' = Track , ...
   unsigned int uiLength;
   char acProducer[30];
   char acModel[30];
};
typedef struct Vehicle sVehicle_t;
typedef sVehicle_t* psVehicle_t;

struct VehicleProduct
{
   psVehicle_t psVec;
   union uPower
   {
      unsigned int uiPower;
      float fPower;
   };
};

struct Config
{
   unsigned int uiState : 3;
   unsigned int uiMaxTurns : 7;
};
```

```
void AG2(void)
{
  sVehicle_t sV = { '1', 409U, "Seat","Ibiza" };

  printf("%u\n", sizeof(sV));                      // _____________ 1P
  printf("%u\n", strlen(sV.acModel));              // _____________ 1P
  printf("%u\n", sizeof(struct VehicleProduct));   // _____________ 1P
  printf("%u\n", sizeof(struct Config));           // _____________ 1P
  printf("%u\n", sizeof(&sV));                     // _____________ 1P

  char* pc = (char*)&sV.acProducer;
  pc += 32;
  printf("%u\n", strlen(pc));                      // _____________ 2P
  printf("%i\n", *((char*)(&sV)));                 // _____________ 2P

  char acRec[] = { "Hello-;GoodBye" };
  char* pcContext = NULL;
  char* pcToken = NULL;
  pcToken = strtok_s(acRec, "-", &pcContext);
  printf("%s\n", pcToken);                         // _____________ 1P
  printf("%i\n", *pcContext);                      // _____________ 2P

  unsigned char ucA = 0x5A;
  unsigned char ucB = 0xFF;
  printf("%x\n", ucA ^ ucB);                       // _____________ 1P
  printf("%x\n", (ucA | ucB) >> 7U);               // _____________ 1P
  printf("%i\n", (ucA & 0x0F));                    // _____________ 1P
}
```

Aufgabe 3: **(20 Punkte)**

3.1 Wandeln Sie die folgenden Zahlen in das andere Zahlensystem um. (4 Punkte)

22_{17} = _____$_{10}$ (1 P)

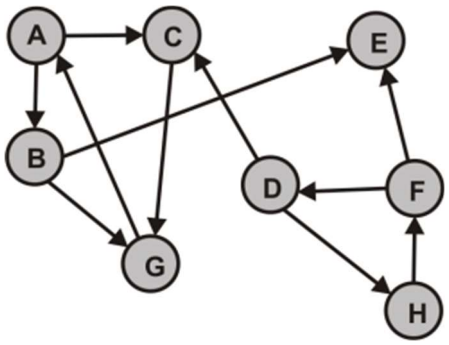
55_{6} = _____$_{5}$ (2 P)

101010_{2} = _____$_{10}$ (1 P)

3.2 Oft findet sich in Bibliotheken ein Makro _countof, welches die Anzahl der Elemente eines Arrays A berechnet. Implementieren Sie ein solches Makro. (3 Punkte)

Hinweis: Nach dem Coding Styleguide müsste der Makroname COUNTOF großgeschrieben werden.

3.3 Mit welchem komplexen Datentyp (der struct Node nutzt) könnten Sie den Graph (siehe unten) abbilden? Beachten Sie, dass A auf C und B zeigt (keine verkettete Liste). (3 Punkte)



Graph

```
struct Node
{
   unsigned char;
};
```

Komplexer Datentyp:

3.4 Gegeben sei ein Array von struct Vehicle. Die einzelnen Elemente seien initialisiert.

```
struct Vehicle asVec[100] = { /* … */ };
```

Zum Sortieren soll die Funktion qsort verwendet werden.

```
struct Vehicle
{
  char cTyp; //'0' = car, '1' = Track ,
  unsigned int uiLength;
  char acProducer[30];
  char acModel[30];
};
typedef struct Vehicle sVehicle_t;
typedef sVehicle_t* psVehicle_t;
```

```
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

3.4.1 1 Implementieren Sie eine Vergleichsfunktion compar, die zwei Vehicle nach uiLength vergleicht. Rückgabewert ist 0, wenn deren uiLenght gleich sind. Ein Rückgabewert >0 bedeutet, dass uiLength vom ersten Vehicle größer ist, <0 bedeutet, dass uiLength vom ersten Vehicle kleiner ist. (5 Punkte)

3.4.2 Rufen Sie jetzt qsort auf, um asVec zu sortieren. Verwenden Sie auch das _countof Makro (2 Punkte)

3.4.3 Implementieren Sie eine Funktion, die den mittleren Wert (Average) für uiLength eines übergebenen Arrays von struct Vehicle zurückgibt. Die Anzahl der Elemente im Array soll ebenso übergegeben werden. (3 Punkte)

```
float GetVehicleAverageLength(                                    )
{
   float fret = 0.F;




}
```

Aufgabe 4: **(15 Punkte)**

4.1 Gegeben sei die einfach verkettete Liste aus der Vorlesung. Implementieren Sie die **rekursive** Funktion **GetNumberElements**, welche die Anzahl der Elemente in der Liste zurückgibt.

Fügen Sie auch ein Sanity-Check ein.
Die Funktion soll die Anzahl der Nodes ausgehend von dem übergebenen Knoten zurückgeben.
(12 Punkte)
5 Sonderpunkte falls Sie nur ein if verwenden!

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

```
unsigned int GetNumberElements(psNode_t psNode)
{

}
```

4.2 main ruft GetNumberElements auf. Zeichnen Sie das Structure Chart. (3 Punkte)

Aufgabe 5: **(15 Punkte)**

Implementieren Sie die Funktion strchr (siehe Anhang).

```
char* strchr(const char* pccStr, int iCh)
{

}
```

Aufgabe 6: **(25 Punkte)**

Eine Thread-Funktion soll die Summe eines übergebenen **Float-Arrays** berechnen. Dabei wird ebenso die **Größe des Arrays** übergeben. Die berechnete **Summe** soll dann ebenfalls abgespeichert werden.

6.1. Deklarieren Sie die Struktur ThreadParam mit drei Elementen (siehe oben, bold), die an die Thread-Funktion übergeben wird. (2 Punkte)

```
struct ThreadParam
{



};
```

6.2 Implementieren Sie die Thread-Funktion inklusive Sanity-Check. Diese erhält das struct aus 6.1 und speichert die berechnete Summe auch darin ab. (8 Punkte)

```
DWORD WINAPI ThreadFunction1(void *pParam)
{
  DWORD dwRet = 0;









  return dwRet;
}
```

6.3 Implementieren Sie die Funktion CalcSum auf dem **Lösungsbogen**. (15 Punkte)

```
float CalcSum(float* pfArray, unsigned int uiSize)
```

Die Funktion teilt die Berechnung der Summe eines Arrays auf drei Threads auf. Jeder Thread soll ungefähr die Summe von gleich vielen Elementen berechnen.

Dabei werden die Ergebnisse aus 6.1 und 6.2 verwendet.

Nach der Synchronisation ist die Gesamtsumme aus den ermittelten Summen der einzelnen Threads zu bestimmen. CloseHandle sei **nicht** notwendig.

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 dwRet;
}
```

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;
}
```

strchr

```
char* strchr(const char* str, int character);
```
Locate first occurrence of character in string
Returns a pointer to the first occurrence of *character* in the C string *str*.

The terminating null-character is considered part of the C string. Therefore, it can also be located in order to retrieve a pointer to the end of a string.

Parameters
str
: C string.

character
: Character to be located. It is passed as its `int` promotion, but it is internally converted back to *char* for the comparison.

Return Value
A pointer to the first occurrence of *character* in *str*.
If the *character* is not found, the function returns a null pointer.

Portability
```
In C, this function is only declared as:

char* strchr(const char* str, int character);
```

Example
```
1  /* strchr example */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main ()
6  {
7    char str[] = "This is a sample string";
8    char * pch;
9    printf ("Looking for the 's' character in \"%s\"...\n",str);
10   pch=strchr(str,'s');
11   while (pch!=NULL)
12   {
13     printf ("found at %d\n",pch-str+1);
14     pch=strchr(pch+1,'s');
15   }
16   return 0;
17 }
```
Edit & run on cpp.sh

Output:

```
Looking for the 's' character in "This is a sample string"...
found at 4
found at 7
found at 11
found at 18
```