Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur

Mikrocontroller, C-Programmierung

Formativer Test

P. Sollberger; Version 1.0

Name: Rohrer Vorname: Felix

Unterschrift:

Rahmenbedingungen:

1. Bearbeitungszeit: **45 Minuten**
2. In der Prüfung können maximal **40 Punkte** erreicht werden. Jeder Aufgabe ist eine maximal erreichbare Punktezahl zugeordnet.
3. Schreiben Sie Ihren Namen auf dieses Blatt und unterschreiben Sie dieses Blatt. Blätter ohne Namensangabe und Unterschrift werden nicht bewertet.
   Mit Ihrer Unterschrift bestätigen Sie, dass Sie die Prüfung persönlich und unter Einhaltung aller Reglemente ausgeführt haben.
4. Es handelt sich um eine schriftliche Prüfung ohne Einsatz des Computers oder elektronischer Hilfsmittel. Sie dürfen keine Unterlagen verwenden.
5. Sollte die Problemstellung Unklarheiten aufweisen, dürfen Sie eigene Annahmen treffen. Führen Sie diese in der Lösung auf.
6. Schreiben Sie möglichst verständlich und gut leserlich. Missverständliche Lösungen werden nicht berücksichtigt.
7. Benutzen Sie den Freiraum unter den Aufgaben für Ihre Lösung.

Für die Korrektur (nicht ausfüllen!)

1	2	3	4	5	6	Punkte
4	5	6	6	8	8	37

Aufgabe 1: Ausdrücke (4 Punkte) 4

Bestimmen Sie die Ausgabe folgender Zeilen.
(1 Punkt pro Zeile → 4 Punkte)

```
int          a = 12;
unsigned int b = 16;

printf("%d\n", ++a);
```

13

```
printf("%d\n", b < 23);
```

1

```
printf("%d\n", b >> 1);
                10000 → 1000
```

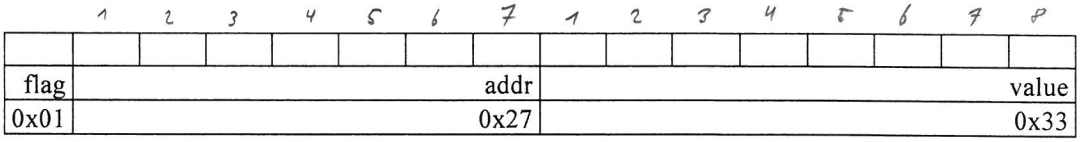
8

```
printf("%s\n", ((a < 20) ^ (b > 0)) ? "JA" : "NEIN");
                  +        +
```

NEIN ✓

Aufgabe 2: Typendefinition und Bitfelder (5 Punkte) 5

a) Definieren Sie einen **Typ** `Header_t`, welcher folgendes Bitmuster aufweist:
(3 Punkte)




```
typedef struct Header_t
  unsigned int flag : 1;
  int addr : 7;
  int value : 8;
} Header_t;
```

✓

b) Deklarieren Sie eine Variable `h` mit Hilfe des neuen Typs und weisen Sie dieser Variable die in der Zeichnung vorgegebenen Werte zu.
(2 Punkte)

```
Header_t h;

h.flag = 1;
h.addr = 0x27;
h.value = 0x33;
```

✓

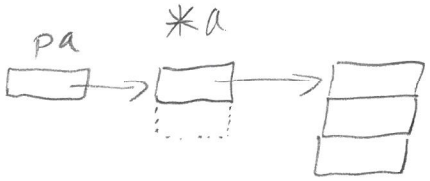
Aufgabe 3: Pointer auf Vektor (6 Punkte) 6

Schreiben Sie die Methode `createAndfillArray(...)`, welche einen Vektor von `n` int-Werten alloziert und ihn mit den Werten `0... n-1` füllt. Die Methode liefert `SUCCESS` zurück, falls die Speicherallokation erfolgreich war, ansonsten `FAILURE`.
Im Code unten sehen Sie, wie die Methode angewendet wird und die dazugehörende Ausgabe.

Konsolenausgabe:

```
0ter Wert: 0
1ter Wert: 1
2ter Wert: 2
```

```
enum FailureCode {SUCCESS, FAILURE};

main()
{
    int i, *a;

    if (createAndfillArray(3, &a) == SUCCESS){
        for (i = 0; i < 3; i++) {
            printf("%dter Wert: %d\n", i, a[i]);
        }
    }
}

/** Creates and fills a vector
 * @param n   Number of elements of the vector
 * @param pa  Reference of the vector to create and fill
 */
enum FailureCode createAndfillArray (int n, int** pa)
{                                          // 1 Punkt ✓

    int  i;

    // Memory allozieren (3 Punkte)
    *pa = (int*) malloc (sizeof(int) * n);   ✓

    if( *pa ) // Fehlerbehandlung (1 Punkt)
    {
        for (i = 0; i < n; i++)
        {
            *(*(pa + i))  // oder (*pa)[i] = i;
            = i; // Wert zuweisen (1 Punkt)
        }
        return SUCCESS;
    }
    else
    {
        return FAIL;
    }
}
```



Aufgabe 4: Header- und Sourcecode-Dateien (8 Punkte) 6

Auf der nachfolgenden Seite finden Sie den Code zu einem Programm, welches einen Stack verwendet. Die Überprüfung des Stacks (Stack bereit, voll oder leer) wird mit Hilfe der Variable `stackState` gemacht.
Teilen Sie nun diesen Code gemäss ‚best practice' in die zwei Dateien `stack.h` und `stack.c` auf, so dass der eingerahmte Code in der Datei `main.c` durch die Anweisung `#include "stack.h"` ersetzt werden kann.

`stack.h` (inklusive Makros zum Verhindern von mehrfachem include): 3

```
#ifndef STACK_H
#define STACK_H
#define SIZE 3 (1)
enum ErrorCode { READY, EMPTY, FULL };
enum ErrorCode stackState = READY;
extern enum ErrorCode stackState;
void push(int);
int pop(void);

#endif
```

`stack.c` (Inhalt der Funktionen können durch … abgekürzt werden): 3

```
#include "stack.h"
int stack[SIZE];
int pos = 0;
void push(int c)
{
  //...
}

int pop(void)
{
  //...
}
```

static
static

* enum ErrorCode stackState = READY;
#define SIZE 3

(1) wird nur innerhalb von stack verwendet → gehört nicht in die Header-Datei

```
#include <stdio.h>
```

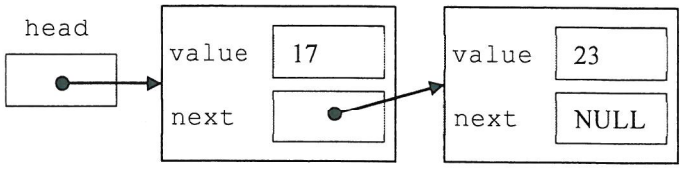
stack.h:

```
#define SIZE 3

enum ErrorCode { READY, EMPTY, FULL };

enum ErrorCode stackState = READY;

int stack[SIZE];
int pos = 0;

void push(int e)
{
    if (pos < SIZE)
    {
        stack[pos++] = e;
        stackState = READY;
    }
    else
    {
        stackState = FULL;
    }
}

int pop(void)
{
    int r = -1;

    if (pos > 0)
    {
        r = stack[pos--];
        stackState = READY;
    }
    else
    {
        stackState = EMPTY;
    }
    return r;
}
```

```
#include "stack.h"

int main(int argc, char** argv)
{
    ...
    r = pop() + pop();
    if (stackState == READY)
    {
        printf("Resultat = %d\n", r);
    }
    else
    {
        printf("Fehler: Zu wenig Operanden auf Stack\n");
    }
    ...
}
```

Aufgabe 5: Strukturen (9 Punkte) 8

Zur Speicherung beliebig vieler Werte setzen Sie eine einfach-verkettete Liste ein.
Ein Knoten in dieser Liste hat neben der Referenz auf das nächste Element (`next`) ein `int`-Attribut `value` für den Wert.



1. Definieren Sie einen neuen Datentyp `NodePtr_t` für einen Zeiger auf einen Knoten.
   (2 Punkte)

```
typedef struct Node_t** NodePtr_t;
```

1

-1p

2. Definieren Sie einen neuen Datentyp `Node_t` für den Knoten. Die Struktur beinhaltet die beiden oben beschriebenen Felder `value` und `next`. Verwenden Sie den unter a.) definierten neuen Datentyp `NodePtr_t`.
   (2 Punkte)

```
typedef struct Node_t {
    int value;
    NodePtr_t next;
} Node_t;
```

2

3. Kreieren (allozieren) Sie dynamisch einen neuen Knoten und weisen Sie dessen Referenz der Wurzel `head` zu. Verwenden Sie die unter b.) definierten Datentypen.
   (2 Punkte)

```
NodePtr_t head = (NodePtr_t) malloc (sizeof(Node_t));
```

✓ 2

4. Schreiben Sie eine Methode `printList(...)`, welche alle Werte in der Liste ausgibt (mit `printf(...)`).
   (3 Punkte)

```
void printList(NodePtr_t p)
{
    if ( p != NULL)
    {
        printf("%d\n", p -> value);
        printList(p -> next);
    }
}
```

✓ 3

Aufgabe 6: Zeiger auf Funktion (8 Punkte) 8

a) Deklarieren Sie eine Variable `f` als Zeiger auf Funktion. Die Funktion erwartet zwei Parameter vom Typ Zeichenkette (Zeiger auf `char`) und gibt einen `float` Wert zurück.
(3 Punkte)

```
float (*f) (char*, char*);
```

✓

b) Implementieren Sie die setter-Funktion `void setF(...)` mit einem Parameter, der die Variable f setzt.
(3 Punkte)

```
void setF( float (*myFnPtr) (char*, char*))
{
  f = myFnPtr;
}
```

✓

c) Rufen Sie die unter b) gesetzte Funktion in der `update()` Funktion auf. Als aktuelle Parameter übergeben Sie die Werte von s1 und s2.
(2 Punkt)

```
void update()
{
    char* s1 = "Hello";
    char* s2 = "World";
    float result;
    setF(...);
    result = f(s1,s2);
}
```

✓