Block 1

Kontrollfragen A

1. Welcher Wert hat w?

```
int w = 034;
```

28 (Dezimal) 034 wird als Octal interpretiert → 34oct = 28dec

2. Wie lauten die Ergebnisse?

```
unsigned int a = 3, b = 5, c = 7;
unsigned int x, y, z;
x = a == 3;
y = (a << 1) & b;
z = (a > 3) ? b : c;
```

1 (a == 3) => true und dies in x speichern -> true = 1

4 (a << 1) a resp. 3 um 1 Bit nach links schieben = 0110bin = 6dec

6dec & (Bitweise AND) b resp. 5dec (0101bin) = 0100bin = 4dec

7 (expr) ? then : else

(a > 3) => false, else -> c = 7

Kontrollfragen B

1. Folgender Code soll x auf 3 setzen, wenn a drei ist (sonst 4), und y auf 3 setzen, wenn a und b drei ist. Was macht dieser Code?

```
y = 4;
if (a == 3)
  x = 3;
  if (b == 3)
    y = 3;
else
  x = 4;
```

Klammern fehlen. Der Compiler ignoriert das Einrücken.

```
y = 4;
if (a == 3) {
  x = 3;
}
if (b == 3) {
  y = 3;
}
Else {
  x = 4;
}
```

Kontrollfragen C

1. Was macht folgender Code?

```
int i = 0;
do
{
  if (i++ % 2)
    continue;
  printf("%d\n", i);
} while (i < 10);
```

Beim ersten Druchgang ist i = 0:

(i++ % 2), zuerst wird (i % 2) überprüft und danach i++ ausgeführt → (0 % 2) = 0, 0 entsprich „false“ somit wird die IF Bedingung nicht erfüllt.

→ Ausgaben: 0\n

Beim zweiten Durchgang ist i = 1:

(1 % 2) != 0 somit If-Bedingung erfüllt-> Continue -> printf() wird übersprungen.

→ Keine Ausgabe

Dritten Durchgang ist i = 2, usw…

➔ Ausgabe: 0, 2, 4, 6, 8

Kontrollfragen D

1. Erklären Sie folgender Code?

```
int i, limit = 100;
char c;
char s[100];
for (i = 0;
     i < limit-1 && (c = getchar()) != EOF && c != '\n'; i++)
{
  s[i] = c;
}
s[i] = '\0';
```

Arraydeklaration für 100 char-Werte → Liest die Eingabe bis maximal 100 Zeichen, ein "ENTER" oder das Ende des Files kommt.

2. Welche Ausgabe erhalten Sie mit folgender Anweisung?

```
float pi = 3.14156f;
printf("%s%f\n", "Pi=", pi);
```

Pi=3.141560 (per Default werden 6 Stellen ausgegeben, das 0 am Ende wird hinzugefügt)

Block 2

Kontrollfragen A

1. Was machen die folgende Zeilen:

```
void xxx(char to[], const char from[])
{
  int i = 0;
  while ((to[i] = from[i]) != '\0')
  {
    i++;
  }
}
```

Kopiert die Zeichenkette (Zeichenvektor) von „from“ nach „to“, solange bis ein \0 kommt.

2. Braucht es zwei Klammerpaare in der while Schleife? Wenn ja, wieso?

Ja, != hat eine höhere Priorität als =.

Kontrollfragen B

1. Definieren Sie einen aufzählenden Typ Color_t, der die drei Werte ROT = 0, BLAU = 5 und GELB = 6 hat.

```
typedef enum Color_
{
 ROT,
 BLAU = 5,
 GELB
} Color_t;
```

2. Wie erzeugen Sie die folgende Ausgabe?

```
Das ist ein Backslash "\".
```

```
printf("Das ist ein Backslash \"\\\"");
```

Kontrollfragen C

```
int y, *ip, *iq;     /* ip und iq sind Zeiger auf int Werte          */
int iWert = 22;
ip = &iWert;

*ip = *ip + 10;      /* Erhöhung der int Variabel um 10              */

y = *ip - 1;         /* Wert von *ip minus 1                         */

*ip += 2;            /* int Variabel inkrementieren                  */

++*ip;               /* Vorrang * vor ++ Operator ++(*ip)  (=35)     */
                     /* Erhöhung des Inhaltes wo ip hinzeigt         */

(*ip)++;             /* Klammern notwendig, sonst *(ip++)  (=36)     */

iq = ip;             /* iq zeigt auf Variable wie ip  (*iq ist 36)   */
```

Kontrollfragen D

1. Wieso können Sie die dynamische Speicherallokation auf kleinen, eingebetteten Systemen selten verwenden?

   Weil diese Systeme zu wenig Speicher / Stack haben, sowie auch weil die Funktionen malloc(), free() etc. nicht vorhanden sind.

2. Kreieren Sie dynamisch einen Vektor für 10 double Werte und initialisieren Sie alle Elemente mit dem Wert 1.0.

```
double* dp;
dp = (double*) malloc(sizeof(double) * 10);
if (dp)
{
  int i;
  for(i=0;i<10;i++) {
    *dp(i) = 1.0d;
  }
}

// do something

free(dp);
```

Block 3

Kontrollfragen A

1. Was macht die folgende Struktur?

```
void addpoint(struct point p1, struct point p2)
{
p1.x += p2.x;
p1.y += p2.y;
}
```

call-by-value Aufruf ohne Rückgabe

Nicht viel….

Es wird eine Kopie von p1 und p2 erstellt und diese innerhalb der Funktion verändert.

Kontrollfragen B

1. Was macht die folgende Struktur?

```
struct point addpoint(struct point p1, struct point p2)
{
p1.x += p2.x;
p1.y += p2.y;
return p1;
}
```

call-by-value: Aufruf mit struct Rückgabe

Funktioniert, jedoch ineffizient!

Kontrollfragen C

1. Was macht die folgende Struktur?

```
void addpoint(struct point* p1, const struct point* p2)
{
p1->x += p2->x;
p1->y += p2->y;
}
```

call-by-reference Aufruf ohne Rückgabe

Verändert Inhalt von p1 (rechnet x und y zusammen)

Const: Inhalt von p2 ist „statisch“, d.h. die Werte von p2 können nicht geändert werden.

Block 4

Kontrollfragen A

1. Taschenrechner mit umgek. polnischer Notation ("HP")

   Infix: (1-2) * (4+5)

   umg.poln: 1 2 - 4 5 + *

   ➔ *Stack*

   push() / pop() / …

Taschenrechner mit umgek. polnischer Notation ("HP")

umg.poln: 1 2 - 4 5 + *

mögliche Lösung mit C-Funktionen:

while (*nächster Operator oder Operand bedeutet nicht Dateiende*) **getop();**

  if (*Zahl*)

    auf den Stack **push();**

  else if (*Operator*)

    Operandenvom Stack holen **pop();**

    Operation ausführen

    Resultat auf den Stack **push();**

  else if (*Zeilenende*)

    Wert vom Stack holen und ausgeben **pop();**

  else

    Fehler

Siehe Buch Beispiel S. 74

Kontrollfragen B

1 Was macht die folgende Funktion?

```
int getop(char s[])
{
int i, c;
while((s[0] = c = getchar()) == ' ' || c == '\t');
if (!isdigit(c) && c != '.'){
s[1] = '\0';
return(c); /*keine Zahl */
}
i = 0;
if (isdigit(c)) /* ganzzahligen Teil sammeln */
while (isdigit(s[++i] = c = getchar()));
s[i] = '\0';
return (NUMBER);
}
```

```
int getop(char s[])
{
        int i, c;

        while((s[0] = c = getchar()) == ' ' || c == '\t')
            ;
        if (!isdigit(c) && c != '.'){
            s[1] = '\0';
            return(c);        /*keine Zahl */
        }
        i = 0;
        if (isdigit(c))    /* ganzzahligen Teil sammeln */
            while (isdigit(s[++i] = c = getchar()))
                ;
        s[i] = '\0';

        return (NUMBER);
    }
```

stdin z.B. File

18 2 - 4 5 + * EOF

s[0] s[1] s[2] s[3] s[4] ... s[n-1]

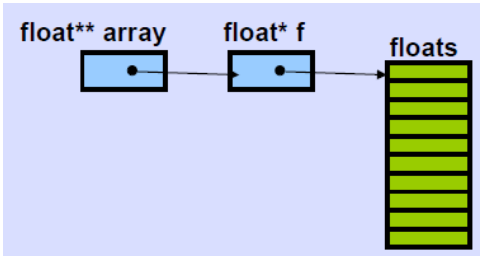
Block 5

Kontrollfragen A

```
main()
{
  float* f; int i;

  if(getFloatArray(10, #1)))
    for(i=0;i<10;i++)
      printf("%dter Wert: %f\n, i, *(f+i));
  free(f);
}

int getFloatArray(int size, #2 array)
{
  int i, rc = 0;
  #3 = (float *)malloc(size*sizeof(float));
  if (*array != NULL) {
    for(i=0; i<size; i++)
      #4 = 0.0f;
  rc = 1;
  }
  return rc ;
}
```



1. Was fehlt bei #1, #2, #3 und #4?

#1 &f

*#2 float ***

*#3 *array*

*#4 (*array)[i] oder *(*array+i)*

Kontrollfragen B - mehrdimensionale Vektoren

1 Was macht diese Funktion hier?

```
static char daytab[2][13] = {
  {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
  {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

int day_of_year(int year, int month, int day)
{
  int i, leap;
  leap = (((year%4 == 0) && (year%100 !=0)) || (year%400 == 0));
  for (i = 1; i < month; i++)
    day += daytab[leap][i];
  return day;
}
```

Liefert die Anzahl Tage bis zu dem übergebenen Datum.

Static => „private“, daytab ist nur innerhalb dieses Modul sichtbar

Kontrollfragen C

Welche Bedeutung hat f?

1. int *f(int wert);

Funktion f (mit Rückgabewert Zeiger auf Int)

2. int* f(int wert);

Funktion f (mit Rückgabewert Zeiger auf Int)

3. int (*f)(int wert);

f ist eine Variable, die einen Zeiger auf eine Funktion beinhaltet f ist ein Zeiger auf eine Funktion

4. int *(*f)(int wert);

f ist ein Zeiger auf eine Funktion (mit Rückgabewert Zeiger auf int)