

**Aufgabe 1:** Finden Sie im folgenden Programm alle Syntaxfehler. ( 10 / 10 )  
 Markieren Sie die Fehler und schreiben Sie hinter bzw. unter die Zeile, wie die Zeile richtig aussehen muss.

```

#include <stdio.h>
#include <stdlib.h>

struct s
{
  ✓ 1 integer int Wert;
  struct s *Next;
};

int main()
{
  ✓ 1 FILE *File = Null NULL;
  ✓ 1 int i = 0;
  struct s *Temp = malloc(struct of sizeof(struct s));
  struct s *Start = Temp;

  // Liste erzeugen (*)
  for ( ; i < 10; )
  {
    ✓ 1 Temp->Wert = ++i;
    if (i < 10)
      Temp->Next = malloc(sizeof(strukt c s));
    else
      Temp->Next = NULL;
    ✓ 1 Temp++ = Temp->Next; Temp = Temp->Next;
  }

  // Liste in Datei schreiben
  Temp = Start;
  ✓ 1 File = fopen("zahlen.txt", "wt");
  ✓ 1 if (File < != NULL)
  {
    while (Temp)
    {
      ✓ 1 fprintf(File, "%i\n", Temp < -> Wert);
      ✓ 1 Temp = Temp -> -> Next;
    }
    ✓ 1 closefile (File);
    } fclose

  // Liste loeschen
  while (Start)
  {
    Temp = Start;
    Start = Start->Next;
    free(Temp);
  }

  return 0;
}

```

**Aufgabe 2:** Multiple-Choice-Fragen.( 15 / 15 )

Kreuzen Sie an, wie der Satz richtig heißen muss. Es gibt immer nur eine richtige Antwort.

Eine make-Datei

- ☐ kann wie eine Header-Datei in ein C-Programm eingebunden werden.
- ✓ ☒ beschreibt Abhängigkeiten.
- ☐ kann vom C-Compiler kompiliert werden.

Mit dem Präprozessorbefehl `#define` wird eine

- ☐ eine Variable definiert.
- ✓ ☐ eine Funktion definiert.
- ☒ eine Konstante definiert.

Wird ein Speicherbereich mit `malloc` reserviert, ist der Inhalt dieses Speicherbereiches

- ☐ mit 0 initialisiert.
- ✓ ☒ undefiniert.
- ☐ mit 255 initialisiert.

Mit dem Schlüsselwort `struct` wird eine

- ✓ ☒ Datenstruktur definiert.
- ☐ Programmstruktur definiert.
- ☐ Computerstruktur definiert.

Der Unterschied zwischen Text- und Binär-Modus bei der Datei-Ein- und Ausgabe liegt u.a.

- ☐ in der Erkennung der deutschen Umlaute.
- ✓ ☒ in der Erkennung des Dateiendes.
- ☐ in der Schreib- und Lesegeschwindigkeit.

Eine Variable namens `FktPointer` vom Typ Zeiger auf eine Funktion, die zwei `int`-Parameter erhält und einen `int`-Zeiger zurückgibt, wird folgendermaßen definiert:

- ☐ `int (*FktPointer* (int, int));`
- ✓ ☐ `int ** (FktPointer(int, int));`
- ☒ `int * (*FktPointer)(int, int);`

Wenn mit der `fopen`-Funktion eine Datei nicht geöffnet werden kann, gibt die Funktion

- ☐ den Wert `EOF` zurück.
- ✓ ☒ einen `NULL`-Zeiger zurück.
- ☐ gar nichts zurück.

Kann ein dynamisch reservierter Speicherbereich nicht mehr freigegeben werden, weil z.B. kein Zeiger mehr auf diesen Speicherbereich verweist, wird dieses

- ✓ ☒ Speicherleck genannt.
- ☐ Computerleck genannt.
- ☐ Programmleck genannt.

Ein mit `#define` definiertes Makro wird wieder entfernt mit dem Befehl

- ☐ `#undefine`
- ✓ ☒ `#undef`
- ☐ `#delete`

Die `fclose`-Funktion

- ☐ gibt nichts zurück.
- ✓ ☐ gibt den Parameter zurück.
- ☒ gibt eine 0 zurück, wenn die Datei geschlossen werden konnte.

Eine Struktur darf

- ✓ ☐ keine Unterstruktur enthalten.
- ☒ beliebig viele Unterstrukturen enthalten.
- ☐ nur eine Unterstruktur enthalten.

Eine andere Schreibweise für `Ptr->Fld` (`Ptr` ist ein Zeiger auf eine Struktur, die das Feld `Fld` beinhaltet) ist

- ✓ ☒ `(*Ptr).Fld`
- ☐ `*(Ptr).Fld`
- ☐ `*(Ptr.Fld)`

Mit `int * const Zeiger;` wird folgendes definiert:

- ☐ Ein unveränderbarer Zeiger auf eine unveränderbare Variable.
- ✓ ☒ Ein unveränderbarer Zeiger auf eine veränderbare Variable.
- ☐ Ein veränderbarer Zeiger auf eine unveränderbare Variable.

Reservierte Speicherbereiche werden freigegeben mit der Funktion

- ☐ `erase`
- ✓ ☐ `remove`
- ☒ `free`

Eine Variable vom Typ `float **` belegt

- ☐ doppelt so viel Speicher wie eine Variable vom Typ `float *`.
- ✓ ☒ genau so viel Speicher wie eine Variable vom Typ `float *`.
- ☐ halb so viel Speicher wie eine Variable vom Typ `float *`.



**Aufgabe 3:** Was gibt das folgende Programm aus? ( 20 / 20 )  
Schreiben Sie die Ergebnisse der vorgegebenen Ausdrücke (Zwischenergebnisse) sowie die Bildschirm-Ausgabe des Programms auf das folgende Blatt!

(1 Punkt je Zwischenergebnis + 4 Punkte für das Endergebnis)

```
#include <stdio.h>

typedef unsigned char Zeichen;

Zeichen Minus1(Zeichen );
Zeichen Plus1(Zeichen );
Zeichen Verdopple(Zeichen );
Zeichen Halbiere(Zeichen );

int main()
{
    Zeichen (*FktArray[4])(Zeichen) = {Minus1,
                                         Plus1,
                                         Verdopple,
                                         Halbiere};

    printf("%c", FktArray[Halbiere(2)](Plus1('G')));
    printf("%c", Minus1(FktArray[Plus1(0)]('k' + 3)));
    printf("for");
    printf("%c", Verdopple(FktArray[Verdopple(1) + 1]('m')));
    printf("%c", Minus1(Minus1(FktArray[Halbiere(0)]('d'))));
    printf("%c", FktArray[1](Minus1(Halbiere(4)) + 'r'));
    printf("%c", FktArray[Plus1(0)](Plus1('g')));
    printf("%c", FktArray[Halbiere(2)](Minus1('k')));
    printf("\n");

    return 0;
}

Zeichen Minus1(Zeichen c)
{ return (c - 1); }

Zeichen Plus1(Zeichen c)
{ return (c + 1); }

Zeichen Verdopple(Zeichen c)
{ return (c * 2); }

Zeichen Halbiere(Zeichen c)
{ return (c / 2); }
```

**Zwischenergebnisse:**

FktArray[Halbiere(2)] zeigt auf Funktion

= Plus1 ✓

FktArray[Halbiere(2)](Plus1('G'))

= 1 ✓

FktArray[Plus1(0)] zeigt auf Funktion

= Plus1 ✓

FktArray[Plus1(0)]('k' + 3)

= 00 ✓

Minus1(FktArray[Plus1(0)]('k' + 3))

= n ✓

FktArray[Verdopple(1) + 1] zeigt auf Funktion

= Halbiere ✓

Verdopple(FktArray[Verdopple(1) + 1]('m'))

= m ✓

FktArray[Halbiere(0)] zeigt auf Funktion

= Minus1 ✓

FktArray[Halbiere(0)]('d')

= c ✓

Minus1(Minus1(FktArray[Halbiere(0)]('d')))

= a ✓

FktArray[1] zeigt auf Funktion

= Plus1 ✓

FktArray[1](Minus1(Halbiere(4)) + 'r'))

= t ✓

FktArray[Plus1(0)] zeigt auf Funktion

= Plus1 ✓

FktArray[Plus1(0)](Plus1('g'))

= i ✓

FktArray[Halbiere(2)] zeigt auf Fkt.

= Plus1 ✓

FktArray[Halbiere(2)](Minus1('k'))

= k ✓**Bildschirm-Ausgabe des oben stehenden Programms:**

(Beachten Sie dabei auch die Formatierungsangaben!)

Informatik

**Aufgabe 4:** Schreiben Sie die drei fehlenden Funktionen zum vorgegebenen Hauptprogramm.

Die Funktion `createAdresse` soll einen Zeiger auf eine neue Adresse zurückgeben. Als Parameter werden Name (Zeichenkette `char *`) sowie der Geburtstag in Form von Tag, Monat und Jahr (jeweils ganze Zahlen) übergeben. Zuerst muss Speicherplatz für die Adresse und für das Datum reserviert werden. Dann können die Werte in die reservierten Speicherbereiche kopiert werden.

Die Funktion `saveAdressen` soll alle Adressen des gleichnamigen Arrays in einer Textdatei, deren Dateiname als Parameter übergeben wird, speichern (Aufbau und Formatierung der Datei siehe unten stehende Ausgabedatei). Auch wenn es hier im Hauptprogramm nicht genutzt wird, soll diese Funktionen einen Wahrheitswert über Erfolg bzw. Nicht-Erfolg des Speicherns zurückgeben.

Die Funktion `compareBirthday` wird als Vergleichsfunktion beim Bubblesort benötigt. Sie soll zwei Zeiger auf Adressen erhalten und eine ganze Zahl zurückgeben: eine 0, wenn das Geburtsdatum von beiden Adressen identisch ist, eine negative Zahl, wenn das Geburtsdatum von der ersten Adresse kleiner ist als das Geburtsdatum von der zweiten Adresse (chronologisch gesehen) und eine positive Zahl, wenn das Geburtsdatum von der ersten Adresse größer ist als das Geburtsdatum von der zweiten Adresse.

Es soll alles in Zeiger-Schreibweise (also KEINE Array-Schreibweise!) geschrieben werden! **Wichtig:** Es sollen die sicherheitsrelevanten Abfragen mit implementiert werden (Konnte Speicher reserviert werden? Konnte die Datei geöffnet werden? Was ist zu tun, wenn etwas nicht geklappt hat? usw.) ( 25 / 25 )

**Ausgabedatei:**

```
Adressen:
Emma Echt          (* 01.01.1978)
Torsten Taler      (* 01.05.1986)
Adam Apfel         (* 25.12.1986)
Paula Puls         (* 26.12.1986)
Max Muster         (* 06.04.1988)
Norbert Napf       (* 03.10.1989)
```

**Hauptprogramm:**

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

#define MAX 20

typedef struct
{ int Day, Month, Year;
} TDate;

typedef struct
{ char *Name;
  TDate *Birthday;
} TAdresse;

int AnzAdressen = 0;
TAdresse *Adressen[MAX];

void freeAdressen();
void sort(TAdresse **, int, int (*)(TAdresse *, TAdresse *));

// Platz für Ihre Funktionsdeklarationen
// 6 Punkte (2 pro Funktionsdeklaration):

TAdresse * createAdresse(char *, int, int, int);
int saveAdressen(char *);
int compareBirthday(TAdresse *, TAdresse *);
```





```

int main()
{
    Adressen[AnzAdressen++] = createAdresse("Max Muster", 6, 4, 1988);
    Adressen[AnzAdressen++] = createAdresse("Adam Apfel", 25, 12, 1986);
    Adressen[AnzAdressen++] = createAdresse("Norbert Napf", 3, 10, 1989);
    Adressen[AnzAdressen++] = createAdresse("Emma Echt", 1, 1, 1978);
    Adressen[AnzAdressen++] = createAdresse("Paula Puls", 26, 12, 1986);
    Adressen[AnzAdressen++] = createAdresse("Torsten Taler", 1, 5, 1986);

    sort(Adressen, AnzAdressen, compareBirthday);
    saveAdressen("adressen.txt");
    freeAdressen();

    return 0;
}

void freeAdressen()
{
    int i;

    for (i = 0; i < AnzAdressen; i++)
    {
        free(*(Adressen + i) -> Birthday);
        free(*(Adressen + i));
    }
}

void sort(TAdresse **Adr, int Anz, int (*cmpFkt)(TAdresse *, TAdresse *))
{
    int i, j;
    TAdresse *tmp;

    for (i = 1; i < Anz; i++)
        for (j = Anz - 1; j >= i; j--)
            if (cmpFkt(*(Adr + j), *(Adr + j - 1)) < 0)
            {
                tmp = *(Adr + j);
                *(Adr + j) = *(Adr + j - 1);
                *(Adr + j - 1) = tmp;
            }
}

// Platz für Ihre Funktionsdefinitionen
// (createAdresse: 6 Pkt, saveAdressen: 9 Pkt, compareBirthday: 4 Pkt):

```

*TAdresse \* createAdresse(char \* Name, int D, int M, int Y)*

```

{
    TDate * NDate = NULL;
    TAdresse * New = NULL;
    New = malloc(sizeof(TAdresse));
    if (New)
    {
        New -> Date Birthday = malloc(sizeof(TDate));
        New -> Name = malloc(strlen(Name) + 1);
        if (New -> Birthday && New -> Name)
        {
            strcpy(New -> Name, Name);
            New -> Birthday -> Day = D;
            New -> Birthday -> Month = M;
            New -> Birthday -> Year = Y;
        }
    }
    return New;
}

```

// Fortsetzung Funktionsdefinitionen für Aufgabe 4

```

int saveAdressen(char * Ad Name)
{
    int i=0;
    FILE * D=NULL;
    D=fopen(Name, "w");
    if (!D)
    {
        printf("Die Datei konnte leider nicht geöffnet werden.");
        return 0;
    }
    fprintf(D, "Adressen: \n");
    for (i=0; i<AnzAdressen; i++)
    {
        fprintf(D, "Adressen: \n"); (*Adressen[i])
        fprintf(D, "%s", Adressen[i] -> Name);
        fprintf(D, "[%02i-%02i-%04i]", Adressen[i] -> Birthday -> Day,
            Adressen[i] -> Birthday -> Month,
            (*Adressen[i]) -> Birthday -> Year);
    }
    fclose(D);
    return 1;
}

```

```

int CompareBirthday(TAdresse * A1, TAdresse * A2)
{

```

```

    int i=0;
    if (A1 && A2)
    {
        i=(A1 -> Birthday -> Year - A2 -> Birthday -> Year);
        if (i==0)
        {
            i=(A1 -> Birthday -> Month - A2 -> Birthday -> Month);
            if (i==0)
            {
                i=(A1 -> Birthday -> Day - A2 -> Birthday -> Day);
            }
        }
    }
    return i;
}

```

if (A1 -> Birthday && A2 -> Birthday)



- Aufgabe 5:** Schreiben Sie die drei fehlenden Funktionen zum vorgegebenen Hauptprogramm.  
Die Funktion printMesswerte soll alle Elemente der verketteten Liste entsprechend der unten stehenden Beispielausgabe auf dem Bildschirm ausgeben. Beachten Sie dabei auch die Formatierungen!  
Die Funktion appendMesswert soll ein Element vom Typ TMesswert in der einfach verketteten Liste anhängen. Als Parameter erhält die Funktion einen Zeiger auf einen Messwert (der auf NULL geprüft werden muss!), zurückgeben soll die Funktion einen Wahrheitswert, ob der Messwert in der Liste angehängen werden konnte (auch wenn das Ergebnis hier im Hauptprogramm nicht genutzt wird!).  
Die Funktion removeMesswert soll den Messwert von der Stunde, die als Parameter übergeben wird, aus der verketteten Liste herausnehmen (ohne den Speicher freizugeben!). Zurückgegeben wird ein Zeiger auf diesen Messwert bzw. einen NULL-Zeiger, wenn es zu der angegebenen Stunde keinen Messwert gibt oder die Liste leer ist. (30 / 30)

**Beispielausgabe:**

```
Tagestemperaturen:
06:00 Uhr: 7.3°C
07:00 Uhr: 8.7°C
08:00 Uhr: 9.4°C
09:00 Uhr: 11.2°C
10:00 Uhr: 14.9°C
11:00 Uhr: 17.5°C
12:00 Uhr: 20.1°C
13:00 Uhr: 23.6°C
14:00 Uhr: 22.9°C
15:00 Uhr: 22.7°C
16:00 Uhr: 21.3°C
17:00 Uhr: 19.1°C
18:00 Uhr: 15.8°C
Tagestemperaturen:
Es liegen keine Messwerte vor!
```

**Hauptprogramm:**

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


typedef struct sMesswert
{
    int Stunde;
    double Temperatur;
    struct sMesswert *Next;
} TMesswert;

TMesswert *First = NULL;
TMesswert *Last = NULL;

TMesswert *createMesswert(int, double);

// Platz für Ihre Funktionsdeklarationen
// 6 Punkte (2 Punkte pro Deklaration):
```

```
void printMesswerte();
int appendMesswert(TMesswert *);
TMesswert * removeMesswert(int);
```



```

int main()
{
    TMesswert *tmp;
    int i;

    appendMesswert( createMesswert( 6, 7.3 ) );
    appendMesswert( createMesswert( 7, 8.7 ) );
    appendMesswert( createMesswert( 8, 9.4 ) );
    appendMesswert( createMesswert( 9, 11.2 ) );
    appendMesswert( createMesswert( 10, 14.9 ) );
    appendMesswert( createMesswert( 11, 17.5 ) );
    appendMesswert( createMesswert( 12, 20.1 ) );
    appendMesswert( createMesswert( 13, 23.6 ) );
    appendMesswert( createMesswert( 14, 22.9 ) );
    appendMesswert( createMesswert( 15, 22.7 ) );
    appendMesswert( createMesswert( 16, 21.3 ) );
    appendMesswert( createMesswert( 17, 19.1 ) );
    appendMesswert( createMesswert( 18, 15.8 ) );

    printMesswerte();

    for (i = 5; i < 20; i += 2)    // Messwerte der ungeraden Stdunden
        free( removeMesswert( i ) );
    for (i = 20; i > 3; i -= 2)    // Messwerte der geraden Stdunden
        free( removeMesswert( i ) );

    printMesswerte();

    return 0;
}

TMesswert *createMesswert(int Std, double M)
{
    TMesswert *Neu = malloc(sizeof(TMesswert));

    if (Neu)
    {
        Neu->Stunde = Std;
        Neu->Temperatur = M;
    }
    return Neu;
}

// Platz für Ihre Funktionsdefinitionen
// (printMesswerte: 5 Pkt, appendMesswert: 7 Pkt, removeMesswert: 12 Pkt):

```

void printMesswerte()

```

{
    TMesswert * temp = xxx First;
    while (temp)
    {
        printf( "Tagestemperaturen\n");
        printf( "%02i:00 Uhr: %.1f °C", temp->Stunde,
            temp->Temperatur);
        temp = temp->Next;
    }
    if (temp == NULL)
        printf( "Es liegen keine Messwerte vor!\n");
}

```

*Handwritten notes and corrections:*

- Under `TMesswert * temp`: ~~xxx~~ First; (with a blue arrow pointing to `First`)
- Under `while (temp)`: `while (temp)` (with a blue arrow pointing to `temp`)
- Under `printf` for temperature: `%.1f °C` (with a red `4` under the `1` and a blue `ln` above the `°C`)
- Under `temp->Temperatur`: `temp->Temperatur` (with a blue `ln` above it)
- Under `if (temp == NULL)`: `if (temp == NULL)` (with a blue `ln` above it)

A large red checkmark is drawn at the bottom right of the handwritten code block.

// Fortsetzung Funktionsdefinitionen für Aufgabe 5

```

int appendMesswert (TMesswert * M)
{
    if (M)
    {
        if (First == NULL)
        {
            First = Last = M;
            M->Next = NULL;
        }
        else
        {
            Last = Last->Next = M;
            M->Next = NULL;
        }
        M->Next = NULL;
        return 1;
    }
    return 0;
}

```

~~TMesswert \* removeMesswert (int S)~~

```

{
    TMesswert * prev = First;
    TMesswert * temp = First;
    while (temp)
    {
        if (temp->Stunde == S)
        {
            temp return temp;
            temp = temp->Next;
        }
    }
}

```

Rückseite!

```


    if (Last->Stunde == S)
    {
        if (First == 0)
            return NULL;
        else if (temp->Stunde == S)
        {
            First = First->Next;
        }
        else
        {
            while (temp->Next)
            {
                if (temp->Next->Stunde == S)
                {
                    temp->Next = temp->Next->Next;
                    temp = temp->Next;
                }
            }
            if (temp == Last)
        }
    }


```



TMesswert \*  
~~int~~ removeMesswert (int ~~int~~ S)

{

TMesswert \*temp = First;

TMesswert \*prev = First;

if (First == NULL)

return ~~int~~ NULL;

if (First -> Shunde == S)

{

temp = First;

if (First == Last)

Last = NULL;

First = First -> Next

return temp;

}

temp = First -> Next;

while (temp != NULL)

{

if (temp -> Shunde == S)

{

prev -> Next = temp -> Next

if (temp == Last)

Last = prev;

~~int~~

return temp;

}

prev = temp; ~~Next~~

temp = temp -> Next;

}

return NULL;

}

