| Lösung Klausur 1, infT | max. Punktezahl: 27<br>N = 5 * P / 24 + 1, auf 1/10 gerundet |
|---|---|

**1.** (6 Pkte.)
Erklären Sie folgende Begriffe im Zusammenhang mit der Programmiersprache C in 2, 3 Sätzen:

a) Promotion: Bei Operationen mit Variablen /Werten unterschiedlichen Typs wird implizite in den genauesten *) beteiligten Typ gecastet. Zudem werden nicht in allen Typen Berechnungen vorgenommen. Reine short- und char-Berechnungen werden z.B. nach int promoviert. (1)

*) auch als "mächtigsten", "breitesten", "grössten" Typ bezeichnet.

b) Delimiter: Delimiter bezeichnet allgemein einen "Begrenzer" (Begrenzungsmarke, -code). In C häufig im Zusammenhang mit C-Strings verwendet, wo '\0' als String Terminator verwendet wird. Dieser Delimiter ist dann von Wichtigkeit, wenn printf, scanf-Funktionen und Funktionen aus der string-Library verwendet werden, da auf den Delimiter getestet wird. Bei Angabe von Stringliteralen begrenzt der C-Compiler den String eigenständig mit dem Delimiter '\0'. (1)

c) type checking: C ist eine typisierte Sprache. Variablen, Rückgabewerte, formale Funktionsparameter und z.T Literale müssen mit einem bekannten Typen versehen sein. Damit wird es möglich, noch während Compile-Zeit bei Aufrufen und Zuweisungen Tests vorzunehmen, sei dies, um implizite Typenwandlungen vorzunehmen, Warnungen auszugeben oder Fehler zu melden, wo eine Typ-Inkompatibilität entdeckt wurde. (1)

d) Signatur: Die Signatur kennzeichnet den Namen einer Funktion, die Anzahl verlangter Parameter durch Erwähnung ihres Typs. Damit ist auch die Reihenfolge der Typen dieser Parameter festgelegt. Die Namen der Parameter sind im Prototypen nicht erforderlich. Der Rückgabewert einer Funktion gehört nicht zur Signatur. (1)

e) Stand-alone Block: Innerhalb von Funktionsblöcken können stand-alone oder freestanding Blocks vorgesehen werden. Sie definieren einen eigenen Scope, wobei Symbole einbettender Blöcke sichtbar sind, soweit sie nicht durch "innere" Symbole gleichen Namens verdeckt sind. Mit Verlassen des stand-alone Blocks endet die Lebenszeit aller im Block ins Leben gerufenen (Auto-) Variablen. (1)

f) Auto-Variable: Synomym dazu ist lokale Variable. Diese wird auf dem Stack angelegt, sobald im Scope ihrer Deklaration begegnet wird. Eine Auto-Variable wird nicht nur automatisch auf dem Stack angelegt, sie wird auch automatisch vom Stack entfernt, wenn der Scope der Variablen verlassen wird. (1)

**2.** (4 Pkte.)
Der C-String **`"Hello World"`** (inklusive Anführungszeichen) ist auf zwei unterschiedliche Arten in ein Characterarray zu bringen. Schreiben Sie zwei vollständige C-Programme (ohne Kommentare).

```
/* p1.c */

int main() {
  char carr[] = "\"Hello World\"";

  return 0;
}
```

(2)

```
/* p2.c */

int main() {
  char carr[14] = {
    '"', 'H', 'e', 'l', 'l', 'o',
    ' ', 'W', 'o', 'r', 'l', 'd',
    '"', '\0',
  };

  return 0;
}
```

(2)

Weitere Lösungen mit **`strcpy`** und **`sscanf`**.

**3.** (2 Pkte.)
Schreiben Sie ein C-Programm, mit dem gezeigt werden kann, dass der C-String-Delimiter auch als solcher wirkt.

```
/* delimiterTest.c */

#define DELIMITER '\0'

int main() {
  //            01234567890123456789012345678901
  char str[] = "Vollstaendiger String sichtbar."

  printf("%s\n", str);
  puts("String wird verkuerzt durch Ersetzen des ersten
        Space mit Delimiter");

  str[14] = DELIMITER;
  printf("%s\n", str);                                  (2)

  return 0;
}
```

Weitere ähnliche Programme möglich.

**4.** (5 Pkte.)
Das nachfolgende Programm weist syntaktische und semantische Fehler auf. Korrigieren Sie diese.

Die erwartete Ausgabe sei:

```
    1
  1 2 3
1 2 3 4 5
1 2 3 4 5 6 7
```

Spalte 0

```
/* triangle.c */
#include <stdio.h>

int main() {
  int blanks = 6;
  int i;                                                (1)

  for (int i = 0; i < 4; ++i) {
    int b = blanks;
    blanks -= 2;                                        (1)
    int n;
    for (; b > 0; --b) {
      putchar(' ');
    }
    n = 1;
    for (; n < (i + 1) * 2; n++) {                      (1)
      printf("%i ", n);                  // Space nach %i   (1)
    putchar('\n');
    }                                                   (1)
  }
  return 0;
}
```

Varianten möglich.

**5.** (1 Pkt.)
Ergänzen Sie Code, der die Information des unten gegebenen C-Strings in eine char-Variable bringt.

```
char str[] = "";
char c = str[0];                                        (1)
```

**6.** (4 Pkte.)
Erstellen Sie einen Datentypen für Einträge in ein Schweizer Telefonbuch. Pro Eintrag ist der Name, der Vorname, die PLZ, der Ort und eine 10-stellige Telefonnummer vorzusehen. Stellen Sie dann eine "Instanz" bereit, die 30 Einträge aufnehmen kann. Tragen Sie einen Teilnehmer an der höchsten Stelle ein.

```
/* telbuch.c */
#include <stdio.h>

#define SIZE 30

struct Telbuch {
  char name[31];
  char vorname[31];
  unsigned int plz;
  char ort[21];
  char tel[11];
};                                                    2

struct Telbuch tb[SIZE];     // grosse Daten besser nicht auf Stack

int main() {
  char n1[4], n2[4], n3[3], n4[3];
  int i = SIZE - 1;

  strcpy(tb[i].name, "Aeberhard-Moser");
  strcpy(tb[i].vorname, "Hildegard Josephine");
  tb[i].plz = 5210;                                                2
  strcpy(tb[i].ort, "Windisch");
  strcpy(tb[i].tel, "0569999999");


  // NICHT GEGENSTAND DER LOESUNG

  printf("Name:    %s\n", tb[i].name);
  printf("Vorname: %s\n", tb[i].vorname);
  printf("PLZ:     %i\n", tb[i].plz);
  printf("Ort:     %s\n", tb[i].ort);
  sscanf(tb[i].tel, "%3s%3s%2s%2s", n1, n2, n3, n4);
  printf("Tel:     %s %s %s %s\n", n1, n2, n3, n4);


  /* Nicht moeglich!
  tb[0] = {
    "Familienname", "Vorname/n", 9999, "Ort", "7777777777"};
  */


  // Die muehsame Art:
  {
    static struct Telbuch telbuch[] = {
      {},                                                                  // Slot 0
          {}, {}, {}, {}, {}, {}, {}, {}, {},
      {}, {}, {}, {}, {}, {}, {}, {}, {}, {},
      {}, {}, {}, {}, {}, {}, {}, {}, {},
      {"FamiliennameX", "Vorname/nX", 1010, "OrtX", "6666666666"}  // Slot 29
    };
    int i = 29;

    printf("\n\n");
    printf("Name:    %s\n", telbuch[i].name);
    printf("Vorname: %s\n", telbuch[i].vorname);
    printf("PLZ:     %i\n", telbuch[i].plz);
    printf("Ort:     %s\n", telbuch[i].ort);
    sscanf(telbuch[i].tel, "%3s%3s%2s%2s", n1, n2, n3, n4);
    printf("Tel:     %s %s %s %s\n", n1, n2, n3, n4);
  }
```

```
  // Neuere C-Compiler schaffen hingegen dies:
  {
    int i = 0;
    struct Telbuch eintrag = {
      "Familienname", "Vorname/n", 9999, "Ort", "7777777777"};

    tb[i] = eintrag;         // Alle Feldwerte werden kopiert!

    printf("\n\n");
    printf("Name:    %s\n", tb[i].name);
    printf("Vorname: %s\n", tb[i].vorname);
    printf("PLZ:     %i\n", tb[i].plz);
    printf("Ort:     %s\n", tb[i].ort);
    sscanf(tb[i].tel, "%3s%3s%2s%2s", n1, n2, n3, n4);
    printf("Tel:     %s %s %s %s\n", n1, n2, n3, n4);
  }

  return 0;
}
```

**7.** (5 Pkte.)
Wie sieht die Ausgabe an der Konsole nach Ausführung des folgenden Programms aus?

```
/* switch.c */
#include <stdio.h>

int main() {
  int arr[] = {24, 5, 23, 6, 1};
  char a, b, c, d, str[] = "a l o e";
  int i;

  sscanf(str, "%c %c %c %c", &a, &b, &c, &d);
  for (i = 4; i >= 0; --i) {
    switch (arr[i] % 5) {
      case 0: putchar('0');
      case 1: putchar(c);
      case 2: putchar('2');
      case 3: putchar(d);
      default: putchar('d');
    }
    putchar('\n');
  }
  return 0;
}
```

Ausgabe:

```
o2ed
o2ed
ed
0o2ed
```

(5)

Abzüge:
-2P Falling through nicht erkannt (keine break)
-1P Keine newlines
-1/2 Falsche Zeichen in Zeile