1 Pointer als Funktionsparameter

```c
struct point_t {
    double x;
    double y;
};
```

Gegeben sei die Struktur `point_t`. Schreiben Sie eine Funktion, die zwei Pointer zu Punkten als Parameter akzeptiert und den Abstand zwischen den Punkten berechnet.

```c
double distance (point_t *p1, point_t *p2);
```

2 Pointer- und Variablenadressen

2.1 Mehrfache Indirektion

Wird ein Pointer als Funktionsargument übergeben, dann wird der Pointerinhalt (gespeicherte Adresse) an die Funktion übermittelt und dort in einer lokaler Variable gespeichert (pass-by-value). Die Information über die Pointer-Adresse, wo der Pointerinhalt gespeichert ist, wird an die Funktion nicht weitergeleitet. In manchen Fällen wollen wir aber diese Information erhalten.
Als erstes schreiben Sie die Funktion `ptr_info` so, die folgende Auskunft über den Pointer gibt. Die Besonderheit bei dieser Funktion ist, dass die Funktion Informationen über die Variable, die an die Funktion übergeben wurde liefert und nicht über die lokalen Variablen. Der Aufruf von `ptr_info` mit dem `pi` als Parameter soll beispielsweise Informationen über `pi` liefern.

Ausgabe von `ptr_info`:

1. Pointeradresse
2. Pointerinhalt
3. **`int`**`-Wert` des Objektes, auf den der Pointer zeigt

Folgender Programmabschnitt soll vervollständigt und ausgeführt werden.

```c
int main (int argc, char ** argv) {
    int i = 5;
    int *pi = &i;
    ptr_info(/* pointer */);
    return 0;
}
```

2.2 Einfache vs. mehrfache Indirektion

In dieser Aufgabe geht es darum die Funktionen `set_to_1()` und `set_to_2()` zu implementieren. Die beiden Funktionen machen tun prinzipiell das Gleiche. Sie geben Informationen über den Pointer aus und setzen das Objekt, auf das die Pointer zeigen auf ein bestimmtes Werte.

- Die Funktion `set_to_1(/*pointer */)`
  - Setzt das Objekt auf 1
  - Ruft `ptr_info` auf
- Die Funktion `set_to_2(/*pointer */)`
  - Setzt das Objekt auf 2
  - Ruft `ptr_info` auf

Implementieren Sie die fehlenden Funktionen und führen Sie das Programm aus.

```
1 int main (int argc, char ** argv) {
2     int i = 5;
3     int *pi = &i;
4     ptr_info(/* pointer */);
5     set_to_1(pi);  // i = 1
6     set_to_2(&pi); // i = 2
7     return 0;
8 }
```

3 Implementation der Summary-Funktion

3.1 Leichtgewichtige Version der Summary-Funktion

Schreiben Sie eine Funktion `summary`, die Minimum, Maximum und Mittelwert von einem **`int`**-Array berechnet.

```
1 void summary (int *arr, size_t size, int *max, int *min, int *mean);
```

Berechnen Sie die Statistiken für die folgenden Arrays:

```
1 int cars_speed[50] = {4, 4, 7, 7, 8, 9, 10, 10, 10, 11, 11, 12, 12, 12,
     ↪ 12, 13, 13, 13, 13, 14, 14, 14, 14, 15, 15, 15, 16, 16, 17, 17,
     ↪ 17, 18, 18, 18, 18, 19, 19, 19, 20, 20, 20, 20, 20, 22, 23, 24,
     ↪ 24, 24, 24, 25};
2
3 int car_dist[50] = {2, 10, 4, 22, 16, 10, 18, 26, 34, 17, 28, 14, 20,
     ↪ 24, 28, 26, 34, 34, 46, 26, 36, 60, 80, 20, 26, 54, 32, 40, 32,
     ↪ 40, 50, 42, 56, 76, 84, 36, 46, 68, 32, 48, 52, 56, 64, 66, 54,
     ↪ 70, 92, 93, 120, 85};
```

Hier sind die richtigen Werte zur Kontrolle:

```
1 > summary(cars)
2        speed           dist
3 Min.   : 4.0   Min.   :  2.00
4 1st Qu.:12.0   1st Qu.: 26.00
5 Median :15.0   Median : 36.00
6 Mean   :15.4   Mean   : 42.98
7 3rd Qu.:19.0   3rd Qu.: 56.00
8 Max.   :25.0   Max.   :120.00
```

3.2 Bonus: Vollständige Implementierung

Berechnen Sie den Median und die Quartile.