`struct Vector`: Allgemeines

Die Datenstruktur `struct Vector` (im Folgenden *Vector*) und die zugehörigen Funktionen implementieren ein dynamisch alloziertes Array. `data` enthält das eigentliche Feld von Elementen vom Typ `T`, in `size` steht dessen aktuelle Anzahl an aktiven Elementen und in `cap` dessen momentane Kapazität an Elementen. Die aktuelle Kapazität ist mindestens so groß wie `size`, kann aber auch größer sein: dies ermöglicht es dem Vector, neue Elemente ohne ständige Reallokation einzufügen.

```
typedef double T;

struct Vector
{
  T *data;          // invariant: points to first element
  size_t size;      // invariant: current number of elements
  size_t cap;       // invariant: current capacity
};
```

Zur Benutzung der Datenstruktur Vector stehen folgende Funktionen bereit:

`vector_new`: Dynamisches Vector-Objekt erzeugen

```
struct Vector *vector_new();
```

Funktionsbeschreibung: Alloziert ein Objekt `struct Vector`, initialisert es (leer) und gibt einen Zeiger darauf zurück.

Rückgabewert:

- Zeiger auf einen dynamisch allozierten Vector.
- Der Aufrufende wird alleiniger Besitzer des dynamischen Speichers und übernimmt somit auch die Pflicht des Freigebens (Hinweis: `vector_delete`).

`vector_delete`: Dynamisches Vector-Objekt freigeben

```
void vector_delete(struct Vector **self);
```

Funktionsbeschreibung: Dealloziert ein per doppelter Referenz übergebenes alloziertes `struct Vector`-Objekt, gibt zuvor den vom Vector selbst gehaltenen dynamischen Speicher frei und setzt den Zeiger auf den Vector auf `NULL`.

Parameter:

- `self`: Zeiger auf einen Zeiger auf einen dynamisch allozierten Vector (Doppelzeiger).

`vector_reserve`: Kapazität reservieren

```
void vector_reserve(struct Vector *self, size_t cap);
```

Funktionsbeschreibung: Erweitert die Kapazität eines übergebenen `struct Vector`-Objekts auf einen Wert **mindestens so groß** wie der ebenfalls übergebenen Wert, falls die aktuelle Kapazität kleiner als der übergebene Wert ist. Die aktiven Elemente bleiben erhalten. Falls der übergebene Wert kleiner als die momentane Kapatität ist findet keine Kapazitätsanpassung statt.

Parameter:

- `self`: Zeiger auf den Vector.
- `cap`: angeforderte Kapazität.

`vector_shrink`: Kapazität freigeben

```
void vector_shrink(struct Vector *self);
```

Funktionsbeschreibung: Reduziert die Kapazität eines übergebenen `struct Vector`-Objekts auf die für die aktiven Elemente benötigte Größe. Die Werte der aktiven Elemente bleiben erhalten.

Parameter:

- `self`: Zeiger auf den Vector.

`vector_init`: Initialisierung/Allokation

```
void vector_init(struct Vector *self, size_t n, T value);
```

Funktionsbeschreibung: Alloziert im übergebenen Vector Speicher für die übergebene Anzahl an Elementen und initialisiert alle Elemente mit dem ebenfalls übergebenen Initialwert. Falls der übergebene Vektor nicht leer ist, werden die bislang gehaltenen Ressourcen vorher freigegeben.

Parameter:

- `self`: Zeiger auf den Vector
- `n`: Anzahl an Elementen
- `value`: Initialwert

`vector_free`: Deallokation

```
void vector_free(struct Vector *self);
```

Funktionsbeschreibung: Falls der übergebene Vector nicht leer ist, werden die bislang gehaltenen Ressourcen freigegeben.

Parameter:

- `self`: Zeiger auf den Vector

`vector_push_back`: Anhängen eines Elements am Ende

```
void vector_push_back(struct Vector *self, T value);
```

Funktionsbeschreibung: Fügt ein weiteres Element am Ende hinzu. Die Werte und Reihenfolge aller existierenden Elemente bleibt unberührt.

Parameter:

- `self`: Zeiger auf den Vector
- `value`: Einzufügenden Wert

`vector_pop_back`: Entfernen des letzten Elements

```
void vector_pop_back(struct Vector *self);
```

Funktionsbeschreibung: Entfernt das letzte Element. Die Werte und Reihenfolge aller anderen bereits existierenden Elemente bleibt unberührt.

Parameter:

- `self`: Zeiger auf den Vector

`vector_push_front`: Anhängen eines Elements am Anfang

```
void vector_push_front(struct Vector *self, T value);
```

Funktionsbeschreibung: Fügt ein weiteres Element am Anfang hinzu. Die bereits vorhandenen Werte wandern dadurch um eins nach hinten, die Reihenfolge bleibt allerdings unberührt.

Parameter:

- `self`: Zeiger auf den Vector
- `value`: Einzufügenden Wert

`vector_pop_front`: Entfernen des ersten Elements

```
void vector_pop_front(struct Vector *self);
```

Funktionsbeschreibung: Entfernt das erste Element. Die Werte und Reihenfolge aller anderen bereits existierenden Elemente bleibt unberührt.

Parameter:

- `self`: Zeiger auf den Vector

`vector_insert_before`: Einfügen eines Elements

```
void vector_insert_before(struct Vector *self, size_t n, T value);
```

Funktionsbeschreibung: Fügt ein weiteres Element vor der übergebene Position ein. Die Werte und Reihenfolge aller existierenden Elemente bleibt unberührt.

Parameter:

- `self`: Zeiger auf den Vector
- `n`: Position vor der eingefügt wird
- `value`: Einzufügenden Wert

Annahme: Die übergebene Position `n` liegt im Interval [`0` , `self.size`].

`vector_erase`: Entfernen eines Elements

```
void vector_erase(struct Vector *self, size_t n);
```

Funktionsbeschreibung: Entfernt das Element an der übergebenen Position. Die Werte und Reihenfolge aller anderen bereits existierenden Elemente bleibt unberührt.

Parameter:

- `self`: Zeiger auf den Vector
- `n`: Position die entfernt wird

Annahme: Die übergebene Position `n` liegt im Intervall [`0` , `self.size - 1`].

`vector_print`: Ausgabe aller Elemente

```
void vector_print(const struct Vector *self);
```

Funktionsbeschreibung: Gibt alle Elemente des uebergebenen Vectors in die Konsole aus. Die Werte der Komponente der Struktur werden ebenfalls ausgegeben.

Parameter `self`: Zeiger auf den Vector