Wer rastet, der rostet

Ein Rust-Kurs für Anfänger

Hendrik Wolff

Anton Obersteiner

Sommersemester 2023

basic memory layout

Bsp.-Start-Adresse	name	Wofür
0		bleibt frei
2000	Text/Code	Maschinencode des Programms
6000	Data	Globale Variablen
9000	Heap	Dynamische Variablen: `malloc`
999000	Stack	Funktions-Lokale Variablen

basic memory layout

Bsp.-Start-Adresse	name	Wofür
0		bleibt frei
2000	Text/Code	Maschinencode des Programms
6000	Data	Globale Variablen
9000	Heap	Dynamische Variablen: `malloc`
999000	Stack	Funktions-Lokale Variablen

```
int global = 0;
int text (char local) {
  Object local_object; // stirbt mit Ende der Funktion
  Object* on_heap = malloc(sizeof(Object)); // existiert weiter
}
```

simple String-Klasse

```
/* wir wollen uns eine Klasse schreiben,
 * die einen String speichert und seine maximale Länge kennt.
 */
class String {
public:
  char* data;
  unsigned int size;
```

Konstruktor: Geburt

```
    // einen String mit Anfangs-Kapazität erstellen
    String (unsigned int size_request) {
      size = size_request;  // save the capacity/size




      printf("old data: '%s'\n", data);

      data = (char*) malloc (size);




      printf("new data: '%s'\n", data);
    }
```

Konstruktor: Geburt

```
// einen String mit Anfangs-Kapazität erstellen
String (unsigned int size_request) {
  size = size_request;  // save the capacity/size

  // bei was anderem als printf: SegFault, uninitialized pointer
  // hier: printet '(null)' weil printf data == 0 bemerkt
  // Compiler beschwert sich nicht

  printf("old data: '%s'\n", data);


  data = (char*) malloc (size);









  printf("new data: '%s'\n", data);
}
```

Konstruktor: Geburt

```
// einen String mit Anfangs-Kapazität erstellen
String (unsigned int size_request) {
  size = size_request;  // save the capacity/size

  // bei was anderem als printf: SegFault, uninitialized pointer
  // hier: printet '(null)' weil printf data == 0 bemerkt
  // Compiler beschwert sich nicht

  printf("old data: '%s'\n", data);

  // geforderten Platz auf Heap belegen, sehr gut

  data = (char*) malloc (size);









  printf("new data: '%s'\n", data);
}
```

Konstruktor: Geburt

```
// einen String mit Anfangs-Kapazität erstellen
String (unsigned int size_request) {
  size = size_request;  // save the capacity/size

  // bei was anderem als printf: SegFault, uninitialized pointer
  // hier: printet '(null)' weil printf data == 0 bemerkt
  // Compiler beschwert sich nicht

  printf("old data: '%s'\n", data);

  // geforderten Platz auf Heap belegen, sehr gut

  data = (char*) malloc (size);

  // printed bis zum ersten Null-Byte,
  // also einfach random Speicherinhalt
  // Compiler könnte uns warnen, dass in data irgendwas steht
  // in C++ ist das aber ein Risiko, vor dem uns niemand schützt

  printf("new data: '%s'\n", data);
}
```

Destruktor: Tod

```
~String () {

  free(data);
}
```

Destruktor: Tod

```
~String () {
  // Speicher frei-/zurückgeben, sehr gut
  free(data);
}
```

String speichern

```
// neuen text kopieren und dann data darauf zeigen lassen
void write (char* text, const int length) {



  char new_data[length];

  sprintf(new_data, "%s", text);  // schreibt text in den Puffer new_data



  data = &(new_data[0]);  // data zeigt jetzt auf den Anfang von new_data
  size = length; // update size of data
}
```

String speichern

```
// neuen text kopieren und dann data darauf zeigen lassen
void write (char* text, const int length) {
  // grundsätzlicher Fehler: new_data ist lokal, also auf dem Stack
  // außerdem: Platz für Null-Byte fehlt
  char new_data[length];

  sprintf(new_data, "%s", text);  // schreibt text in den Puffer new_data



  data = &(new_data[0]);  // data zeigt jetzt auf den Anfang von new_data
  size = length; // update size of data
}
```

String speichern

```
// neuen text kopieren und dann data darauf zeigen lassen
void write (char* text, const int length) {

  // grundsätzlicher Fehler: new_data ist lokal, also auf dem Stack
  // außerdem: Platz für Null-Byte fehlt

  char new_data[length];


  sprintf(new_data, "%s", text);  // schreibt text in den Puffer new_data

  // data nicht freed :/
  // rest ist an sich ~ok~

  data = &(new_data[0]);  // data zeigt jetzt auf den Anfang von new_data
  size = length; // update size of data
}
```

printing itself

```
  void print () const {


    printf("%s", data);




  }
};
```

printing itself

```
  void print () const {
    // printf muss hier nicht viel machen, legt deswegen wenig stack an
    // und überschreibt new_data nicht, weswegen es kein Symptom gibt
    printf("%s", data);




  }
};
```

printing itself

```
  void print () const {
    // printf muss hier nicht viel machen, legt deswegen wenig stack an
    // und überschreibt new_data nicht, weswegen es kein Symptom gibt
    printf("%s", data);
    // printf("%s\n", data); verwendet tatsächlich stack und schreibt
    // @`J'J'
    // '@@oЮ'
    // und so ähnliches
  }
};
```

main(), erste Hälfte

```
int main(void) {
  // String mit 64 Bytes Platz, aber Länge 0 erstellen
  String my_string (64);

  char* some_data;
  if (my_string.size ≥ 64) {


    some_data = my_string.data;
  } else {


    some_data = (char*) malloc (64);
  }

  printf("Please write your favorite color: ");
  scanf("%s", some_data); // Zeile vom Nutzer in den Puffer einlesen
  printf("Thank you.\n");
```

main(), erste Hälfte

```
int main(void) {
  // String mit 64 Bytes Platz, aber Länge 0 erstellen
  String my_string (64);

  char* some_data;
  if (my_string.size ≥ 64) {
    // borrow data from my_string
    some_data = my_string.data;
  } else {


    some_data = (char*) malloc (64);
  }

  printf("Please write your favorite color: ");
  scanf("%s", some_data); // Zeile vom Nutzer in den Puffer einlesen
  printf("Thank you.\n");
```

main(), erste Hälfte

```
int main(void) {
  // String mit 64 Bytes Platz, aber Länge 0 erstellen
  String my_string (64);

  char* some_data;
  if (my_string.size ≥ 64) {
    // borrow data from my_string
    some_data = my_string.data;
  } else {
    // neu allozieren
    some_data = (char*) malloc (64);
  }

  printf("Please write your favorite color: ");
  scanf("%s", some_data); // Zeile vom Nutzer in den Puffer einlesen
  printf("Thank you.\n");
```

main(): zweite Hälfte

```
    // String some_data in my_string hineinschreiben, dazu Länge bestimmen
    my_string.write(some_data, strlen(some_data));
    printf("You said your favorite color was: ");
    my_string.print();
    printf("\n");



    free(some_data); // möglicher 'double free'
}
```

main(): zweite Hälfte

```
    // String some_data in my_string hineinschreiben, dazu Länge bestimmen
    my_string.write(some_data, strlen(some_data));
    printf("You said your favorite color was: ");
    my_string.print();
    printf("\n");

    // ob some_data borrowed oder alloziert ist,
    // müsste man eigentlich extra checken, z.B. mit
    // if (my_string.size < 64)

    free(some_data); // möglicher 'double free'
}
```

typische Ausführung

```
/* Funktioniert nur deswegen halbwegs, weil Speicher
 * mit nullen initialisiert ist und printf bisschen was abfängt.
$ g++ kaputt.cpp -o kaputt && ./kaputt
old data: '(null)'
new data: ''
Please write your favorite color: null_ptr
Thank you.
You said your favorite color was: 'null_ptr'
free(): invalid size
Aborted (core dumped)
 */
```