

Systemprogrammierung - AIN/2

Sommersemester 2022

Übungsaufgabe 2: C Arrays

Abgabe bis 21./22.4.2022


Migration von Java nach C: Bubblesort

In Programmiertechnik 1 haben wir ein Java-Programm Bubblesort besprochen, das beliebig viele ganze Zahlen in ein Array einliest und aufsteigend sortiert wieder ausgibt. Das hier vorgegebene Java-Programm **Bubblesort.java** ist so erweitert, dass es die zu sortierenden Zahlen entweder von der Standardeingabe einliest oder mit dem Zufallsgenerator erzeugt. Stellen Sie dieses Java-Programm auf C um:

- Die Quelldatei soll `bubblesort.c` heißen.
- Der Kopf des Hauptprogramms muss `int main(int argc, char *argv[])` lauten. Dabei ist `argc` der C-Ersatz für die Instanzvariable `args.length` beim Java-Array.
- Die Entsprechung zum Java-Ausdruck `Integer.parseInt(args[0])` ist der C-Ausdruck `atoi(argv[1])`.
- Verwenden Sie die Bibliotheksfunktion `malloc` oder `calloc` zum Reservieren des Speichers für das Array.
- Den Zufallsgenerator der C-Standardbibliothek initialisieren Sie mit `srand((unsigned int) time(NULL))` und die nächste Zufallszahl erhalten Sie mit dem Funktionsaufruf `rand()`.

Denken Sie an die Fehlerbehandlung nach der Speicherreservierung und beim Einlesen der ganzen Zahlen sowie an die Freigabe des Speichers am Programmende.

Manueller Test

Speichern Sie die Datei  **Makefile** in Ihr Arbeitsverzeichnis der Aufgabe 2 und verwenden Sie zum Testen die Befehle:

```
javac Bubblesort.java
java Bubblesort 10
make bubblesort
./bubblesort 10
valgrind ./bubblesort 10
make cppcheck
```

Testen Sie Ihre C-Programm mit unterschiedlichen Array-Größen und Eingaben, auch mit extremen und falschen Werten:

- Verhält sich das C-Programm bei allen Eingaben wie das Java-Programm?

- Funktioniert die Fehlerbehandlung bei Speicherreservierung und Einlesen?
- Meldet valgrind Fehler?
- Meldet cppcheck Probleme?

Bessern Sie gegebenenfalls nach.

Automatisierter Test

Verwenden Sie die folgenden Linux-Befehle, um zu prüfen, ob Ihr bubblesort überhaupt richtig sortiert:

```
./bubblesort 1000 < /dev/null | tail -1000 > out.txt  
sort -n out.txt | diff - out.txt
```

Das Umlenken der Standardein-/ausgabe mit `<` und `|` haben Sie schon im ersten Semester kennengelernt. Die Angabe von `/dev/null` als Eingabequelle bewirkt, dass bubblesort bei allen Leseversuchen ein Eingabeende erhält, als hätten Sie `Strg-D` eingegeben. Informationen zu den verwendeten Linux-Befehlen mit ihren Optionen und Argumenten erhalten Sie mit

```
man tail  
man sort  
man diff
```

oder über die auf der Literaturseite der Vorlesung genannten Internetseiten.

- Was gibt die obige Befehlsfolge im Terminal aus, wenn Ihr bubblesort richtig sortiert hat?

Laufzeitmessung

Verwenden Sie die folgenden Linux-Befehle, um die Laufzeit Ihres Programms zu messen:

```
time java Bubblesort 1000 < /dev/null > /dev/null  
time ./bubblesort 1000 < /dev/null > /dev/null
```

Das Schlüsselwort `time` veranlasst die Unix-Shell, die Ausführungszeit des dahinter folgenden Kommandos zu messen. Relevant ist für Sie die Zeile mit der `user-Zeit`. Informationen dazu finden Sie auf den Handbuchseiten des Kommandozeileninterpreters (siehe `man bash` oder entsprechende Internetseiten).

- Wächst die Ausführungszeit tatsächlich quadratisch mit der Array-Größe?
- Ist das Java-Programm oder das C-Programm schneller? Können Sie sich den Unterschied erklären?

Führen Sie zur Beantwortung der beiden Fragen eine Messreihe mit den Array-Größen 1000, 10000 und 100000 durch.

Übersetzen Sie das C-Programm auch mal mit der Optimierungsoption `-O2` (Buchstabe Groß-O, nicht die Ziffer Null):

```
make "CC=gcc -g -O2" clean all
```

- Ist das optimierte Programm erkennbar schneller?

Wiederholen Sie zur Beantwortung der Fragen die obige Messreihe. Verwenden Sie `clean`, um das nicht optimierte und das optimierte Programm nicht durcheinander zu bringen.

`make clean all` löscht das aktuelle Programm und erzeugt die nicht optimierte Version, und `make "CC=gcc -g -O2" clean all` löscht das aktuelle Programm und erzeugt die optimierte Version.

Protokoll

Erstellen Sie ein Protokoll Ihres automatisierten Tests und Ihrer Laufzeitmessungen. Gehen sie dazu so vor wie in [Aufgabe 1](#) beschrieben. Nennen Sie die Protokolldatei `protokoll-aufgabe2.txt` und ergänzen Sie darin Ihre Antworten auf alle obigen Fragen.

Abgabe

Führen Sie Ihr Programm und Ihre Protokolldatei vor.

Hinweis:

Der Compiler gcc darf für Ihr Programm keine Fehler oder Warnungen mehr ausgeben. Ihr Programm muss außerdem ordentlich formatiert sein. Bessern Sie die Formatierung gegebenenfalls mit `astyle` nach:

```
astyle -p -H --style=ansi bubblesort.c
```

Freiwillige Zusatzaufgabe (1 Bonuspunkt pro Spiegelpunkt)

Betrachten Sie die folgenden zwei `for`-Schleifen, die beide `n` mal auf ein Array zugreifen:

```
for (int i = 0; i < n; ++i)
{
    int r = rand() % n;
    a[r] = r;
}

for (int i = 0; i < n; ++i)
{
    int r = rand() % n;
    a[i] = r;
}
```

- Welche der beiden Schleifen ist bei sehr großem `n` schneller?

Testen Sie Ihre Hypothese mit zwei kleinen C-Programmen. Nutzen Sie wie im Pflichtteil das `time`-Kommando.

- Erklären Sie, wie der Laufzeitunterschied zwischen den beiden Schleifen zustande kommt. Sie brauchen für die Antwort Kenntnisse über Rechnerarchitekturen. Der entscheidende Begriff fängt mit dem Buchstaben C an.