Fakultät IV Elektrotechnik und Informatik
Einführung in die Programmierung WS 2021/2022
Manfred Hauswirth / Damien Foucard / Christoph Seifert / Uwe Kuehn

Programmierblatt 08

Ausgabe: *13.01.2022 12:00*
Abgabe: *25.01.2022 08:00*

Thema: *Heaps*

Abgabemodalitäten

1. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf Deinem Rechner mit dem Befehl `clang -std=c11 -Wall -g` kompilieren.
2. Die Abgabe für den Quellcode erfolgt ausschließlich über unser Git im entsprechenden Branch. Nur wenn ein Ergebnis im ISIS-Kurs angezeigt wird, ist sichergestellt, dass die Abgabe erfolgt ist. Die Abgabe ist bestanden, wenn Du an Deinem Test einen grünen Haken siehst.
3. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben. Lies Dir die Hinweise der Tests genau durch, denn diese helfen Dir Deine Abgabe zu korrigieren.
Bitte beachte, dass ausschließlich die letzte Abgabe gewertet wird.
4. Die Abgabe erfolgt, sofern nicht anders angegeben, in folgendem Branch: `iprg-b<xx>-a<yy>`, wobei `<xx>` durch die zweistellige Nummer des Aufgabenblattes und `<yy>` durch die entsprechende Nummer der Aufgabe zu ersetzen sind.
5. Gib für jede Aufgabe die Quellcodedatei(en) gemäß der Vorgabe ab. Im ISIS-Kurs werden zum Teil Vorgabedateien bereitgestellt. Nutze diese zur Lösung der Aufgaben.
6. Die Abgabefristen werden vom Server überwacht. Versuche Deine Abgabe so früh wie möglich zu bearbeiten. Du minimierst so auch das Risiko, die Abgabefrist auf Grund von „technischen Schwierigkeiten“ zu versäumen. Eine Programmieraufgabe gilt als bestanden, wenn alle bewerteten Teilaufgaben bestanden sind.

Aufgabe 1 Implementierung eines binären Max-Heaps (bewertet)

Eine Fluggesellschaft möchte das Einsteigen der Passagiere in ihre Flugzeuge beschleunigen. (Es wird angenommen, dass die Flugzeuge nur über den Eingang hinter dem Cockpit betreten werden.) Dazu sollen möglichst immer die Sitzreihen, welche sich am weitesten hinten im Flugzeug befinden, zuerst besetzt werden. (Die Sitzreihen werden vom Cockpit an hochgezählt.) Sobald ein Passagier am Gate eintrifft, wird er mit seiner Sitzreihe registriert. Neue Sitzreihen werden jeweils erst aufgerufen, wenn im Kabinengang wieder Platz vorhanden ist. Für diese Aufgabe soll ein System entwickelt werden.

Das System soll nach dem folgenden Muster funktionieren:

Eingabe <Zahl> Wenn ein Passagier beim Gate erscheint, soll dessen Sitzreihe (eine positive ganze Zahl) in die Warteliste eingetragen werden.

Eingabe ’n’ Falls im Kabinengang des Flugzeugs wieder ausreichend Platz ist, soll das Bodenpersonal die Sitzreihe mit der größten Nummer angezeigt bekommen, für die ein Passagier am Gate wartet. Eine leere Warteliste wird durch die Ausgabe von “Leer” angezeigt.

Eingabe ’q’ Das Programm wird durch Eingabe von “q” (für “quit”) beendet.

Andere Eingaben Auf alle anderen Eingaben wird eine Fehlernachricht angezeigt.

Um die Möglichkeit von Betriebsfehlern zu reduzieren, müssen die folgenden Bedingungen erfüllt sein:

- Die Implementierung verwendet einen binären Max-Heap. Der Heap ist in `introprog_heap.h` als `struct heap` vorgegeben.
- Die Implementierung basiert auf dem Pseudocode der Vorlesung.
- Eventueller dynamischer Speicher muss freigegeben werden.

Die Ein-/Ausgabe sowie das Beenden des Programmes sind bereits vorgegeben.

Ein beispielhafter Aufruf des Programmes ist in Listing 1 dargestellt.

Listing 1: Programmbeispiel

```
> clang -std=c11 -Wall introprog_maxheap.c \
    introprog_main_maxheap.c -o introprog_maxheap
> ./introprog_maxheap
[...]
> n
Leer!
> 12
> 33
> 3
> n
33
```

```
> n
12
> n
3
> n
Leer!
> foo
Fehler: Eingabe nicht erkannt!
> q
```

Folgende Funktionen müssen implementiert werden:

Aufgabe 1.1 Speicherverwaltung

Implementiere anhand der Funktionssignaturen in der Vorgabe die Funktion `heap_create(size_t capacity)` und `heap_free(heap* h)`. Achte dabei darauf, dass du nicht mehr Speicher als nötig reservierst, den Speicher korrekt initialisierst und am Ende alles wieder freigibst.

Aufgabe 1.2 Funktion heapify

Schreibe die Funktion `void heapify(heap* h, int i)`, die sicherstellt, dass der Knoten i und seine Kinderknoten die Heap-Eigenschaft erfüllen, falls die Unterbäume der Kinderknoten die Heap-Eigenschaft erfüllen.

Aufgabe 1.3 Funktion heap_insert

Schreibe die Funktion `int heap_insert(heap* h, int val)`, die ein neues Element in den Heap einfügt. Achte dabei darauf, dass die Datenstruktur auch nach dem Einfügen ein gültiger Heap ist. Orientiere dich an dem Pseudocode der Vorlesung.

Sollte der Funktion ein Heap übergeben werden, der schon vollständig aufgefüllt worden ist (d.h. ein Heap mit genau `capacity` Elementen), soll -1 zurückgeben werden.

Aufgabe 1.4 Funktion heap_extract_max

Schreibe die Funktion `int heap_extract_max(heap* h)`, die das größte Element des Heaps zurückgibt und gleichzeitig vom Heap entfernt. Achte dabei darauf, dass auch nach dem Entfernen des Elements der Heap die Heapeigenschaft erfüllt. Orientiere dich an dem Pseudocode der Vorlesung.

Sollte der Funktion einen Heap übergeben werden, der leer ist (d.h. ein Heap mit 0 Elementen), soll -1 zurückgeben werden.

Hinweis: In bisherigen Aufgaben wurde der Einfachheit halber den Funktionen `malloc` und `calloc` ein `int` übergeben. Schaut man jedoch in die Dokumentation zu diesen Funktionen, wird dort die folgende Funktionssignatur gezeigt: `void *malloc(size_t size);`.

Der Datentyp `size_t` ist immer so groß wie ein Pointer (der verwendeten Architektur) und ist speziell für Pointerarithmetik gedacht. Auf der aktuell verbreitetsten Architektur x86/64 entspricht `size_t` einem `unsigned long long` (8 Byte). Damit macht es in der Praxis auf dieser Architektur keinen Unterschied, ob man (bei kleineren Datenmengen als 2GB) ein `int` oder `size_t` für den Aufruf und malloc/calloc oder zum Indizieren von Arrays verwendet. Ein Cast ist für die Konvertierung `int -> size_t` auch nicht erforderlich, da hierbei keine Informationen verloren gehen. Da C jedoch eine Sprache ist, die explizit dafür konzipiert wurde auf unterschiedlichen Plattformen programmiert werden zu können, wollen wir an dieser Stelle die Methode wählen, die auf allen Architekturen zuverlässig funktioniert.

Für die zu bearbeitende Aufgabe hat das zur Folge, dass wir den Funktionen `heap* heap_create(size_t capacity)` und `void heapify(heap* h, size_t i)` nun kein `int`, sondern eine Variable vom Typ `size_t` übergeben. Im Umgang mit diesen Variablen ändert sich für euch nichts.

Nutze zur Lösung der Aufgabe die Vorgaben aus unserem ISIS-Kurs. Füge Deine Lösung als Datei `introprog_maxheap.c` im entsprechenden Abgabebranch in Dein persönliches Repository ein und übertrage die Lösung an die Abgabeplattform.