Fakultät IV Elektrotechnik und Informatik
Einführung in die Programmierung WS 2019/2020
Manfred Hauswirth / Damien Foucard / Christoph Seifert

Aufgabenblatt 9

Ausgabe: *17.01.2020*
Abgabe: *28.01.2020 09:59*

Thema: *Heaps*

Abgabemodalitäten

1. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des ZE-CM/eecsIT mittels `clang -std=c11 -Wall -g` kompilieren.
2. Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Nur wenn ein Test in Osiris angezeigt wird ist sichergestellt, dass die Abgabe erfolgt ist.
3. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben.
4. Die Abgabe erfolgt in folgendem Unterordner:
   `introprog-wise1920/Studierende/<L>/<TU-Login>/Abgaben/Blatt<XX>`
   wobei `<L>` durch den ersten Buchstabe des TU-Logins und `<XX>` durch die Nummer des Aufgabenblattes zu ersetzen sind. Die Ordner werden automatisch angelegt sobald die Abgabe freigeschaltet wird.
5. Du darfst den Abgabeordner für das Blatt nicht selbst erstellen. Die Ordner erstellen wir kurz nach der Ausgabe der Aufgabe.
6. Um die Änderungen (z. B. neue Abgabeordner) vom Server abzuholen, musst Du den Befehl `svn update` im Wurzelverzeichnis (also im Verzeichnis `introprog-wise1920`) des Repositories ausführen.
7. Im Abgabeordner gelten einige restriktive Regeln. Dort ist nur das Einchecken von Dateien mit den in der Aufgabe vorgegebenen Namen erlaubt, ausserdem werden die Abgabefristen vom Server überwacht. Beachte eventuelle Fehlermeldungen beim SVN-Commit. Lade nur Dateien hoch, die Du selbst bearbeiten sollst, insbesondere also keine Vorgaben.
8. Der Dateiname bei Abgaben mit C-Code soll dem der Vorgaben entsprechen. Entferne ausschließlich `_vorgabe` aus dem Name. Bei Textaufgaben sind, wenn nicht anders angegeben, `.txt` Dateien zugelassen, die als Plaintext-Datei zu speichern sind (keine Word-Dateien umbenennen).
9. Es gibt einen Ordner `Arbeitsverzeichnis`, in dem Du Dateien für Dich ablegen kannst.
10. Die Ergebnisse der automatischen Tests kannst Du auf OSIRIS einsehen:
    https://osiris.ods.tu-berlin.de/

Aufgabe 1 Binäre Heaps (2 Punkte)

Aufgabe 1.1 Heap-Kriterien (1 Punkt)

Betrachte die Bäume in der Abbildung 1 und gib jeweils an, ob es sich um einen Max-Heap, einen Min-Heap oder gar keinen Heap handelt.

Gibt weiterhin für die gültigen Heaps die Anordnung der Daten in der Arrayrepräsentation an.

Die Abgabe erfolgt über die Beantwortung der Fragen des Formulars `introprog_heap_identification_vorgabe.txt`, welches Du im SVN findet.

Aufgabe 1.2 Heap erstellen (1 Punkt)

Wende die in der Vorlesung vorgestellte Funktion `build_heap` auf die angegebenen Arrays an und überführe diese in einen Max- bzw. Min-Heap.

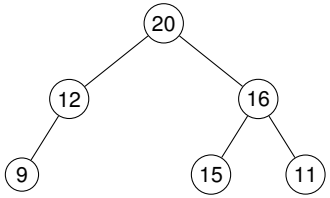
a) Max-Heap: `[8,1,2,5,3,1]`
b) Min-Heap: `[42,23,15,16,8,4]`
c) Min-Heap: `[10,100,1,10,100,10]`

Die Abgabe erfolgt über die Beantwortung der Fragen des Formulars `introprog_build_heap_vorgabe.txt`, welches Du im SVN findet.

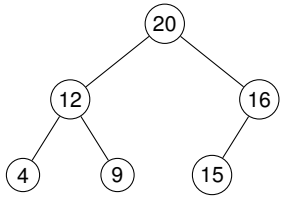
Dabei wird der Zustand des Heaps wie folgt beschrieben:

- Jeder Knoten im Baum wird nummeriert.
- Die Nummerierung weist der Wurzel den Index 1, seinen Kindern die Werte 2 (links) und 3 (rechts) zu usw. (siehe Abbildung 3).
- Um zu beschreiben, dass der Knoten mit dem Index 5 den Wert 11 enthält, verwenden wir die Notation “5:11”.
- Die Angabe mehrerer Elemente erfolgt jeweils auf einer eigenen Zeile.
- Im Unterschied zu den binären Suchbäumen aus Aufgabenblatt 6 verwenden wir an dieser Stelle sowohl für die Indizes als auch für die Werte der Knoten Zahlen. Beachtet also, dass der erste Wert den Index repräsentiert und der zweite den Wert.
- Der Zustand des Baums aus Abbildung 2 wird somit wie folgt beschrieben:

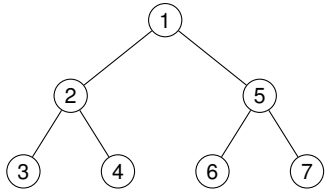
```
1: 7
2: 8
3: 11
4: 12
5: 9
```



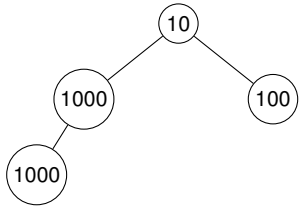
(a) Baum 1



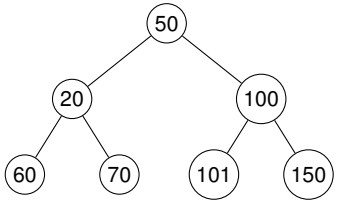
(b) Baum 2



(c) Baum 3



(d) Baum 4



(e) Baum 5

Abbildung 1: Heaps oder nicht?

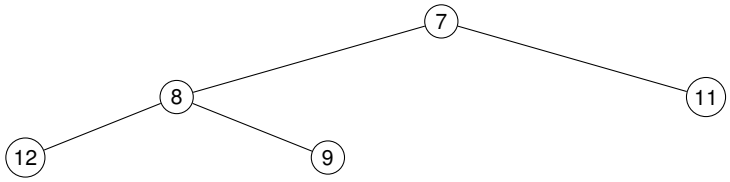


Abbildung 2: Min-Heap

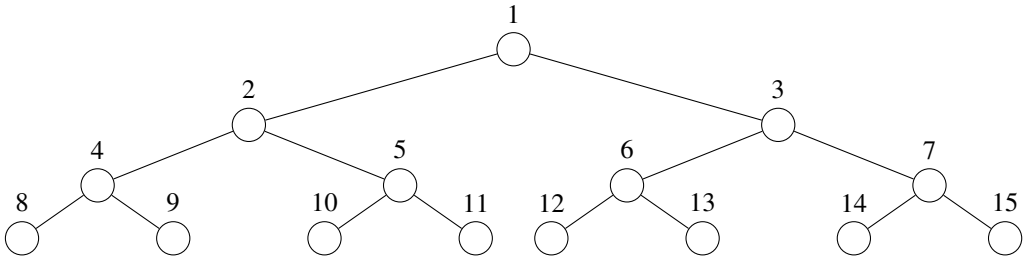


Abbildung 3: Nummerierung der Knoten im Baum

Aufgabe 2 Implementierung eines binären Max-Heaps (8 Punkte)

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
```

Die Codevorgabe im SVN besteht aus drei Dateien: `introprog_heap.h`, `introprog_main_maxheap.c` und `introprog_maxheap_vorgabe.c` (siehe Listing 2). Die letzte Datei soll angepasst werden und als `introprog_maxheap.c` eingereicht werden.

Folgende Funktionen müssen implementiert werden:

Aufgabe 2.1 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 2.2 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 2.3 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 2.4 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.

Listing 2: Codevorgabe `introprog_maxheap_vorgabe.c`

```
/* Reserviere Speicher für einen Heap
 *
 * Der Heap soll dabei Platz für capacity Elemente bieten.
 *
```

```
12  * Um konsistent mit den Parametern für malloc zu sein, ist
13  * capacity als size_t (entspricht unsigned long auf x86/64)
14  * deklariert.
15  * Da es sauberer ist, sind auch alle Indizes in dieser Aufgabe
16  * als size_t deklariert. Ob man size_t oder int als Index für
17  * ein Array verwendet, macht in der Praxis (auf x86/64) nur
18  * bei Arrays mit mehr als 2.147.483.647 Elementen einen
19  * Unterschied.
20  */
21 heap* heap_create(size_t capacity) {
22 }
23
24 /* Stellt die Heap Bedingung an Index i wieder her
25  *
26  * Vorraussetzung: Der linke und rechte Unterbaum von i
27  * erfüllen die Heap-Bedingung bereits.
28  */
29 void heapify(heap* h, size_t i) {
30 }
31
32 /* Holt einen Wert aus dem Heap
33  *
34  * Wenn der Heap nicht leer ist, wird die größte Zahl
35  * zurückgegeben. Wenn der Heap leer ist, wird -1 zurückgegeben.
36  */
37 int heap_extract_max(heap* h) {
38 }
39
40 /* Fügt einen Wert in den Heap ein
41  *
42  * Wenn der Heap bereits voll ist, wird -1 zurückgegeben.
43  */
44 int heap_insert(heap* h, int key) {
45 }
46
47 /* Gibt den Speicher von einem Heap wieder frei
48  */
49 void heap_free(heap* h) {
50 }
```