Informatik für Ingenieure

Wintersemester 2022/23 - Übungsklausur 2 Wintersemester 2022/23

Prof. Dr.-Ing. Görschwin Fey

Auswertungsbericht für Matrikelnummer 552864 erstellt am 17.2.2023 um 09:54:32 Uhr.

Übersicht

Aufgabe	Punkte	Ergebnis
Aufgabe 1	2	2.0
Aufgabe 2	2	2.0
Aufgabe 3	5	2.5
Aufgabe 4	5	0.0
Aufgabe 5	5	5.0
Aufgabe 6	5	5.0
Aufgabe 7	5	5.0
Aufgabe 8	5	5.0
Aufgabe 9	5	0.0
Aufgabe 10	5	5.0
Aufgabe 11	6	6.0
Aufgabe 12	25	24.0
Aufgabe 13	25	25.0
Summe	**100**	**86.5**

Ergebnis

Dies ist die Auswertung eines Testats für die Veranstaltung *Informatik für Ingenieure* im *Wintersemester 2022/23*. Die erreichten Punkte werden anteilig als Bonus auf eine bestandene Klausur angerechnet.

Auswertungsbericht für 552864

Aufgabe 1

2.0 / 2 Punkten

0011000011101001 ist eine Binärzahl. Wählen Sie die gleiche Zahl in hexadezimaler Darstellung aus.

Wählen Sie genau eine Antwortmöglichkeit aus. Sie erhalten nur Punkte für diese Aufgabe, wenn Ihre Auswahl der richtigen Antwort entspricht.

- (A) 30E9 **2 Punkte**
- (B) 2BF7
- (C) 34A0
- (D) A302
- (E) 2451
- (F) 195D

Erstellt am 17.2.2023 um 09:54:32 Uhr

Aufgabe 2

2.0 / 2 Punkten

Addieren Sie folgende vorzeichenlose, binäre Zahlen: 10000 + 11111. Die resultierende Binärzahl kann so viele Ziffern enthalten wie nötig, um das Ergebnis exakt zu repräsentieren.

Wählen Sie genau eine Antwortmöglichkeit aus. Sie erhalten nur Punkte für diese Aufgabe, wenn Ihre Auswahl der richtigen Antwort entspricht.

- (A) 101111 **2 Punkte**
- (B) 110110
- (C) 11010
- (D) 1100100
- (E) 100101
- (F) 00100

Auswertungsbericht für 552864

Aufgabe 3

2.5 / 5 Punkten

Welche der folgenden logischen Ausdrücke sind äquivalent zu *(D ∨ E)*?

In den Booleschen Ausdrücken sind A, D und E boolesche Variablen und diese Symbole repräsentieren folgende logischen Operatoren:
⊕ := XOR
∧ := AND
∨ := OR
¬ := NOT

Wählen Sie für jede Antwortmöglichkeit aus, ob diese richtig oder falsch ist. Für die richige Auswahl erhalten Sie Punkte. Wenn Sie eine falsche Auswahl treffen, dann reduziert dies Ihre Punktzahl. Antwortmöglichkeiten, für die Sie keine Auswahl treffen, werden mit 0 Punkten bewertet.

A ? ✓ ✗ $(D \wedge \neg(\neg(D \vee E) \wedge (\neg D \wedge E))) \vee E$ **1.25 Punkte**

B ? ✓ ✗ $\neg(\neg D \wedge \neg E)$ **1.25 Punkte**

C ? ✓ ✗ $(D \vee D) \wedge (D \vee E) \wedge (E \vee \neg D)$ **1.25 Punkte**

D ? ✓ ✗ $(D \vee E) \oplus (A \vee \neg A)$ **-1.25 Punkte**

Erstellt am 17.2.2023 um 09:54:32 Uhr

Auswertungsbericht für 552864

Aufgabe 4

0.0 / 5 Punkten

Geben Sie an, ob diese Aussagen über die Computerarchitektur korrekt oder nicht korrekt sind.

Wählen Sie für jede Antwortmöglichkeit aus, ob diese richtig oder falsch ist. Für die richige Auswahl erhalten Sie Punkte. Wenn Sie eine falsche Auswahl treffen, dann reduziert dies Ihre Punktzahl. Antwortmöglichkeiten, für die Sie keine Auswahl treffen, werden mit 0 Punkten bewertet.

A ? ✓ ✗ Die C++-Standard-Bibliothek erlaubt es, direkt mit den Peripheriegeräten des Betriebsystems zu interagieren. **-1.25 Punkte**

B ? ✓ ✗ Betriebsysteme erlauben einen gewissen Grad an Hardwareunabhängigkeit. Daher ist es nicht notwendig, Programme für unterschiedliche Hardware-Architekturen neu zu übersetzen. **1.25 Punkte**

C ? ✓ ✗ Bei der Verwendung eines Interpreters ist es möglich, direkt auf Nutzereingaben zu reagieren. **1.25 Punkte**

D ? ✓ ✗ Interpreter übersetzen Programme schrittweise und können daher nicht über längere Code-Abschnitte hinweg optimieren. **-1.25 Punkte**

Erstellt am 17.2.2023 um 09:54:32 Uhr

Auswertungsbericht für 552864

Aufgabe 5

5.0 / 5 Punkten

Nutzen Sie die Codezeilen auf der rechten Seite, um einen kompilierbaren Code zu erstellen, der die Ausgabe '84' erzeugt.

Sie können einige der Bausteine von der rechten Seite auf die linke Seite ziehen und sortieren. Nicht alle Textbausteine müssen benutzt werden. Nutzen Sie nur notwendige Bausteine.

Code

```
#include <iostream>
int main() {
int a = 5, b;
b = 3;
a = a + b;
b = a - 4;
std::cout << a << b << std::endl;
}
```

Nicht genutzt

```
cout << a-4 << b+4 << endl;
b = *a - 4;
```

Ihre Antwort	Lösung	Punkte
		5.00
int a = 5, b;	int a = 5, b;	0
b = 3;	b = 3;	0
a = a + b;	a = a + b;	0
b = a - 4;	b = a - 4;	0
std::cout << a << b << std::endl;	std::cout << a << b << std::endl;	0
}	}	0
		Total: 5.00

Erstellt am 17.2.2023 um 09:54:32 Uhr

Auswertungsbericht für 552864

Aufgabe 6

5.0 / 5 Punkten

Welche Aussagen über die gezeigte Funktion sind korrekt?

Wählen Sie für jede Antwortmöglichkeit aus, ob diese richtig oder falsch ist. Für die richige Auswahl erhalten Sie Punkte. Wenn Sie eine falsche Auswahl treffen, dann reduziert dies Ihre Punktzahl. Antwortmöglichkeiten, für die Sie keine Auswahl treffen, werden mit 0 Punkten bewertet.

```
void tausche(int x, int y) {
     int hilf = x;
     x = y;
     y = hilf;
}
```

A ? ✓ ✗ Die Variablen `x` und `y` werden über Call-by-Reference übergeben. **1.25 Punkte**

B ? ✓ ✗ Die Variablen `x` und `y` werden über Call-by-Value übergeben. **1.25 Punkte**

C ? ✓ ✗ Die Funktion liefert keinen Returnwert. **1.25 Punkte**

D ? ✓ ✗ Wenn man die Funktion aufruft, sind die Parameterwerte anschließend vertauscht. **1.25 Punkte**

Auswertungsbericht für 552864

Aufgabe 7

5.0 / 5 Punkten

Welche Aussagen treffen bezüglich Testfällen und Testfallgenerierung zu?

Wählen Sie für jede Antwortmöglichkeit aus, ob diese richtig oder falsch ist. Für die richige Auswahl erhalten Sie Punkte. Wenn Sie eine falsche Auswahl treffen, dann reduziert dies Ihre Punktzahl. Antwortmöglichkeiten, für die Sie keine Auswahl treffen, werden mit 0 Punkten bewertet.

A ? ✓ ✗ Um ein Programm auf Richtigkeit zu testen, reichen zufällig definierte Tests aus. **1.25 Punkte**

B ? ✓ ✗ Testfälle sollten alle Code-Zeilen eines Programmes wenigstens einmal ausführen. **1.25 Punkte**

C ? ✓ ✗ Testfälle sollten Randfälle, z.B. größte und kleinste zugelassene Werte eines Parameters, abdecken. **1.25 Punkte**

D ? ✓ ✗ Wenn eine Verzweigung eines z.B. if-else Konstrukts getestet wurde, müssen weitere Verzweigungen nicht berücksichtigt werden. **1.25 Punkte**

Erstellt am 17.2.2023 um 09:54:32 Uhr

Auswertungsbericht für 552864

Aufgabe 8

5.0 / 5 Punkten

In einer Menge *M* mit strenger Totalordnung gilt für drei Elemente *a,b,c* $\in M$ immer das Folgende:

Wählen Sie für jede Antwortmöglichkeit aus, ob diese richtig oder falsch ist. Für die richige Auswahl erhalten Sie Punkte. Wenn Sie eine falsche Auswahl treffen, dann reduziert dies Ihre Punktzahl. Antwortmöglichkeiten, für die Sie keine Auswahl treffen, werden mit 0 Punkten bewertet.

A ? ✓ ✗ Wenn $a < b$, dann $a < c$. **1.00 Punkte**

B ? ✓ ✗ Wenn $a < b$, dann gilt $b < a$ nicht. **1.00 Punkte**

C ? ✓ ✗ Exakt eine der Aussagen $a < b$, $a = b$, oder $b < a$ ist wahr. **1.00 Punkte**

D ? ✓ ✗ $a < b$, $a = b$, und $b < a$ sind wahr. **1.00 Punkte**

E ? ✓ ✗ Aus $a < b$ und $b < c$, folgt $a < c$. **1.00 Punkte**

Auswertungsbericht für 552864

Aufgabe 9

0.0 / 5 Punkten

Beantworten Sie die folgenden Fragen unter Berücksichtigung des gegebenen Quelltexts.

Wählen Sie für jede Antwortmöglichkeit aus, ob diese richtig oder falsch ist. Für die richige Auswahl erhalten Sie Punkte. Wenn Sie eine falsche Auswahl treffen, dann reduziert dies Ihre Punktzahl. Antwortmöglichkeiten, für die Sie keine Auswahl treffen, werden mit 0 Punkten bewertet.

```
1   class Building {
2     private:
3       int floors;
4
5     protected:
6       int w, l;
7
8     public:
9       Building() : floors(3),
10                   w(1), l(1) {
11      }
12
13      int calc_rent() {
14        return -1;
15      }
16  };
17
18  class Shop : public Building {
19    public:
20      int calc_rent() {
21        return w * l * 50;
22      }
23  };
```

A ? ✓ ✗ Die Klasse `Shop` erbt die Member-Variable `floors` von der Klasse `Building`. **1.25 Punkte**

B ? ✓ ✗ Gegeben sei folgender Code-Abschnitt: `Building* a = new Shop ();`, dann ergibt die Anweisung `a->calc_rent()` zur Laufzeit den Wert 50. **-1.25 Punkte**

C ? ✓ ✗ Polymorphismus kommt zum Tragen, wenn es eine Klassenhierarchie gibt, bei denen die Klassen voneinander erben und eine virtuelle Methode aufgerufen wird. **-1.25 Punkte**

D ? ✓ ✗ Für eine Oberklasse kann es mehrere Unterklassen geben. **1.25 Punkte**

Auswertungsbericht für 552864

Aufgabe 10

5.0 / 5 Punkten

Welchem Zweck dient die Definition des Kopfs (Zeiger auf das erste Element) einer Liste?

Wählen Sie für jede Antwortmöglichkeit aus, ob diese richtig oder falsch ist. Für die richige Auswahl erhalten Sie Punkte. Wenn Sie eine falsche Auswahl treffen, dann reduziert dies Ihre Punktzahl. Antwortmöglichkeiten, für die Sie keine Auswahl treffen, werden mit 0 Punkten bewertet.

A ? ✓ ✗ Der Kopf bietet Zugriff auf das erste Element der Liste. **1.00 Punkte**

B ? ✓ ✗ Der Kopf legt die maximale Länge der Liste fest. **1.00 Punkte**

C ? ✓ ✗ Der Kopf stellt einen festen Einstiegspunkt für die Traversierung der Liste dar. **1.00 Punkte**

D ? ✓ ✗ Der Kopf übernimmt die Allokation von Speicher für die gesamte Liste. **1.00 Punkte**

E ? ✓ ✗ Der Kopf bestimmt das letzte Element einer Liste. **1.00 Punkte**

Auswertungsbericht für 552864

Aufgabe 11

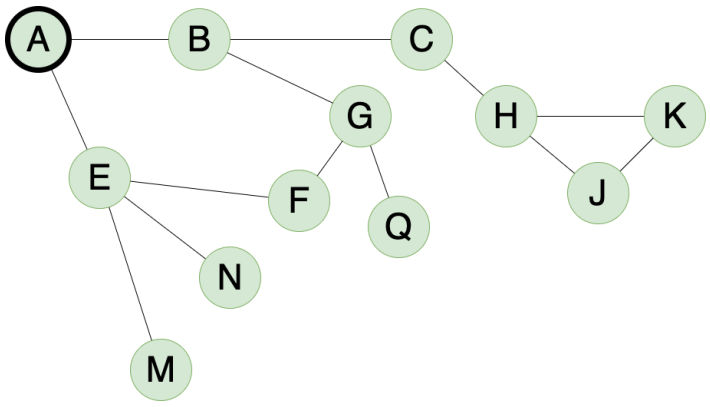
6.0 / 6 Punkten

In welcher Reihenfolge besucht eine Tiefensuche von "A" ausgehend die Knoten des ungerichteten Graphen? An Verzweigungen wird stets der Knoten mit dem kleineren Schlüssel zuerst besucht, z. B. wird "B" vor "E" bevorzugt. Bringen Sie die Elemente in der Liste von links nach rechts via *Drag & Drop* in die richtige Reihenfolge.

Hinweis: **Der Algorithmus besucht alle Knoten.**

A | B | C | H | J | K | G | F | E | M | N | Q

Ihre Antwort	Lösung	Punkte
		6.00
B	B	0
C	C	0
H	H	0
J	J	0
K	K	0
G	G	0
F	F	0
E	E	0
M	M	0
N	N	0
Q	Q	0
		Total: 6.00



Erstellt am 17.2.2023 um 09:54:32 Uhr

Auswertungsbericht für 552864

Erstellt am 17.2.2023 um 09:54:32 Uhr

Aufgabe 12

24.0 / 25 Punkten

Aufgabenstellung

Motivation

Sie haben in der Veranstaltung Algorithmen kennengelernt, um Daten zu sortieren. In dieser Aufgabe geht es um das Suchen von Elementen in einem Array. Hier wird das Array eine besondere Ordnung haben, sodass das Suchen effizient gestaltet werden kann.

Aufgaben

In der Datei `search.hpp` sind Methoden deklariert, die Sie in der Datei `search.cpp` implementieren sollen.

a. `int *create(int size)`

erzeugt ein Array mit mindestens einem Eintrag und initialisiert alle Werte mit `-1`. Diese Funktion soll eine `std::invalid_argument`-Exception werfen, wenn die übergebene Größe des Arrays zu klein ist.

b. `void set(int *&array, int size, int index, int value)`

setzt den Eintrag innerhalb des Arrays an der Stelle "`index`" auf den gegebenen Wert "`value`". Die Funktion soll eine `std::invalid_argument`-Exception werfen, wenn der gegebene Index nicht im Array enthalten ist.

c. `int search(int *&array, int size, int value)`

sucht nach dem Wert in dem Array und gibt zurück, wieviele Einträge besucht wurden. Der Algorithmus ist hier in natürlicher Sprache gegeben:

```
Initialisiere Variable "index" mit 0.
Initialisiere Variable "anzahl" mit 0.

Solange "index" kleiner "size" {
  Erhöhe "anzahl" um 1.
  Wenn der Eintrag des Arrays an der Stelle "index" der gesuchte Wert ist, dann gib "anzahl" zurück.
  Wenn der Eintrag des Arrays an der Stelle "index" größer als "value" ist, dann weise "index" das Ergebnis von ((index + 1)
· 2 - 1) zu.
  Anderenfalls weise "index" das Ergebnis von ((index + 1) · 2) zu.
}

Wirf die Exception "std::logic_error".
```

d. `void multiply(int *&array, int size)`

multipliziert alle Einträge des Arrays mit 2.

e. `std::string to_string(int *&array, int size)`

konvertiert das Array in einen String. Ein Array der Größe 0 soll nur mit eckigen Klammern dargestellt werden: `[]`. Ein Array mit einem Eintrag soll dieses in den Klammern enthalten `[4]`. Bei größeren Arrays werden die Einträge mit einem Komma getrennt: `[3,5]`.

f. `void free(int *&array)`

gibt den Speicher wieder frei und setzt `array` auf `nullptr`.

Hinweise

- Sie können die Kommentare aus der .hpp auch in die .cpp kopieren, damit sie weniger zwischen den Tabs wechseln müssen.
- Sofern nicht anders angegeben, müssen Sie Parameter nicht auf ihre Gültigkeit prüfen.
- Eine Exception wird durch das Schlüsselwort `throw` erzeugt und es kann dem Konstruktor eine Fehlernachricht übergeben werden: `std::range_error("Hallo Welt, ich bin ein Fehler.")`
- Einen `std::string` kann man initialisieren, indem man ihm ein String-Literal zuweist: `std::string var = "Beispiel";`.
- Ein `std::string` oder ein String-Literal kann an einen anderen `std::string` mit Hilfe des +-Operators angehängt werden: `std::string var = var + "e";`
- Die Funktion `std::string std::to_string(int)` wandelt Integer in Strings um.

Folgende Dateien wurden Ihnen während der Prüfung zur Verfügung gestellt:

main.cpp

```
#include "search.hpp"

#include <iostream>
```

```
void print(int *array, int size) {
   if(array == nullptr) {
       std::cout << "null array" << std::endl;
       return;
   }

    std::cout << "[";
    for(int i = 0; i < size;++i) {
        if(i> 0) std::cout << ",";
        std::cout << array[i];
    }
    std::cout << "]";

    std::cout << std::endl;
}

int main() {
    int size = 8;
    int * array = create(size);

    std::cout << "Das Array wurde angelegt (erwartet [-1,-1,-1,-1,-1,-1,-1,-1]): "; print(array, size);
    std::cout << "Created array (expected [-1,-1,-1,-1,-1,-1,-1,-1]): "; print(array, size);
    std::cout << std::endl;

    set(array, size, 0, 23);
    set(array, size, 1, 10);
    set(array, size, 2, 43);
    set(array, size, 3, 7);
    set(array, size, 4, 12);
    set(array, size, 5, 42);
    set(array, size, 6, 50);
    set(array, size, 7, 3);

    std::cout << "Das Array wurde gefüllt (erwartet [23,10,43,7,12,42,50,3]): "; print(array, size);
    std::cout << "Initialized array (expected [23,10,43,7,12,42,50,3]): "; print(array, size);
    std::cout << std::endl;

    int steps1 = search(array, size, 42);

    std::cout << "Die Zahl 42 wurde nach  " << steps1 << " Schritten gefunden (3 erwartet)." << std::endl;
    std::cout << "Found 42 after  " << steps1 << " steps (expected 3)." << std::endl;
    std::cout << std::endl;

    multiply(array, size);

    std::cout << "Die Zahlen des Arrays wurden verdoppelt (erwartet [46,20,86,14,24,84,100,6]): "; print(array, size);
    std::cout << "Doubled array (expected [46,20,86,14,24,84,100,6]): "; print(array, size);
    std::cout << std::endl;

    int steps2 = search(array, size, 46);

    std::cout << "Die Zahl 100 wurde nach  " << steps2 << " Schritten gefunden (erwartet 1)." << std::endl;
    std::cout << "Found 100 after  " << steps2 << " steps (expected 1)." << std::endl;
    std::cout << std::endl;

    std::cout << "Konvertiere Array in ein String (erwartet [46,20,86,14,24,84,100,6]): " << to_string(array, size) << std::endl;
    std::cout << "Converted array to string (expected [46,20,86,14,24,84,100,6]): " << to_string(array, size) << std::endl;
    std::cout << std::endl;

    free(array);

    std::cout << "Das Array wurde freigegeben (erwartet 0x0): " << array << std::endl;
    std::cout << "The array was freed (expected 0x0): " << array << std::endl;
    std::cout << std::endl;

    return 0;
}
```

search.cpp

```
#include "search.hpp"

/*
 * HINWEIS:
 *
 * Details zu den Funktionen gibt es in der Header-Datei. Der zu implementierende
 * Algorithmus der Suche ist in der Aufgabenbeschreibung gegeben.
 *
 * REMARK:
 *
 * The function details are given in the header file and the task description.
 *   In the task description, you will find the description of the search algorithm
 *   as well.
 */

int *create(int size) {
    return nullptr;
}

void set(int *&array, int size, int index, int value) {

}

int search(int *&array, int size, int value) {
    return -1;
}
```

```
void multiply(int *&array, int size) {

}

std::string to_string(int *&array, int size) {
    return "";
}

void free(int *&array) {

}
```

search.hpp

```
#ifndef SEARCH_HPP
#define SEARCH_HPP

#include <stdexcept>
#include <string>

/**
 * @brief Erzeugt ein Array der uebergebenen Groesse und initialisiert die Werte mit -1.
 *
 * @param size Die Groesse des zu erzeugenden Arrays.
 * @return int* Das erzeugte Array
 * @throws std::invalid_argument Wenn die angegebene Größe kleiner oder gleich 0 ist
 *
 *
 * @brief Creates an array of the given size and initializes the values with -1.
 *
 * @param size The size of the array.
 * @return int* The newly created array.
 * @throws std::invalid_argument If the given size is equal or less to 0.
 *
 */
int *create(int size);

/**
 * @brief Speichert "value" an der Stelle "index" des Arrays.
 *
 * @param array Das Array, in dem der Wert gesetzt werden soll
 * @param size  Die Größe des Arrays.
 * @param index Index, an dem der Wert gesetzt werden soll.
 * @param value Der Wert, der gesetzt werden soll.
 * @throws std::invalid_argument Wenn "index" außerhalb der Arraygrenzen liegt.
 */
/**
 * @brief Stores "value" in the "array" at index "index".
 *
 * @param array The array where the value should be stored.
 * @param size  The size of the array.
 * @param index The index where the value should be set.
 * @param value The value to set.
 * @throws std::invalid_argument If "index" is out of the array boundaries.
 */
void set(int *&array, int size, int index, int value);

/**
 * @brief Sucht den Wert "value" im "array". Für einen detaillierten Algorithmus
 *   schauen Sie bitte in die textuelle Aufgabenbeschreibung.
 *
 * @param array Das Array, in dem gesucht werden soll.
 * @param size Die Größe des Arrays.
 * @param value Der Wert, der gesucht werden soll.
 * @return int Anzahl an besuchten Elementen.
 * @throws std::logic_error Wenn das Element nicht gefunden werden kann.
 */
/**
 * @brief Searches "value" in the array "array". Please have a look at the
 *  textual task description for the exact algorithm to implement.
 *
 * @param array The array to search in.
 * @param size The size of the array.
 * @param value The value that should be searched.
 * @return int Number of visited array elements.
 * @throws std::logic_error If the value could not be found.
 */
int search(int *&array, int size, int value);

/**
 * @brief Multipliziert alle Werte des eingegebenen Arrays mit zwei.
 *
 * @param array Das zu bearbeitende Array.
 * @param size Größe des Arrays.
 */
/**
 * @brief Multiplies all values of the given array by two.
 *
 * @param array The array to use.
 * @param size Size of the array
 */
void multiply(int *&array, int size);

/**
 * @brief Erzeugt einen String aus dem übergebenen Array. Ein Array der
 *   Größe 0 soll nur mit eckigen Klammern dargestellt werden: "[]" Ein
```

```
 *   Array mit einem Eintrag soll dieses in den Klammern enthalten "[4]".
 *   Bei größeren Arrays werden die Einträge mit einem Komma getrennt: "[3,5]".
 *
 * Hinweise:
 *   Die Anführungszeichen oben sind nur als Veranschaulichung gedacht und sollen
 *      nicht im String enthalten seien.
 *   std::string std::to_string(int): Diese Funktion konvertiert einen Integer in
 *      einen String.
 *   std::string lassen sich mit dem +-Operator aneinander hängen.
 *   An einen std::string lässt sich ein einfaches String-Literal, wie zum Beispiel
 *      "Hallo Welt", durch den +-Operator anhängen.
 *
 * @param array Das zu konvertierende Array.
 * @param size Die Größe des Arrays.
 * @return std::string Der erzeugte String.
 */
/**
 * @brief Converts the array to a string. An empty array should be represented
 *   by brackets: "[]". An array with a single element should print the element
 *   between the brackets: "[4]". For larger arrays the elements should be separated
 *   by a comma: "[3,5]".
 *
 * Hints:
 *   The quotation marks in the description are only to show the beginning and end of the example
 *     and should not be part of the string.
 *   std::string std::to_string(int): This function converts an integer to a string.
 *   You can concatenate std::string by using the +-operator.
 *   You can concatenate a std::string and a simple string literal, such as "hello world",
 *     by using the +-operator.
 *
 * @param array The array to convert.
 * @param size Size of the array.
 * @return std::string The created string.
 */
std::string to_string(int *&array, int size);

/**
 * @brief Gibt das übergebene Array frei und setzt den übergebenen Zeiger auf "nullptr".
 *
 * @param array Das freizugebende Array.
 */
/**
 * @brief Frees the given array and stores "nullptr" in the parameter.
 *
 * @param array The Array to free.
 */
void free(int *&array);

#endif //SEARCH_HPP
```

Folgende Dateien haben Sie in der Prüfung bearbeitet bzw. erstellt:

main.cpp

```
#include "search.hpp"

#include <iostream>

void print(int *array, int size) {
   if(array == nullptr) {
       std::cout << "null array" << std::endl;
       return;
   }

    std::cout << "[";
    for(int i = 0; i < size;++i) {
        if(i> 0) std::cout << ",";
        std::cout << array[i];
    }
    std::cout << "]";

    std::cout << std::endl;
}

int main() {
    int size = 8;
    int * array = create(size);

    std::cout << "Das Array wurde angelegt (erwartet [-1,-1,-1,-1,-1,-1,-1,-1]): "; print(array, size);
    std::cout << "Created array (expected [-1,-1,-1,-1,-1,-1,-1,-1]): "; print(array, size);
    std::cout << std::endl;

    set(array, size, 0, 23);
    set(array, size, 1, 10);
    set(array, size, 2, 43);
    set(array, size, 3, 7);
    set(array, size, 4, 12);
    set(array, size, 5, 42);
    set(array, size, 6, 50);
    set(array, size, 7, 3);

    std::cout << "Das Array wurde gefüllt (erwartet [23,10,43,7,12,42,50,3]): "; print(array, size);
    std::cout << "Initialized array (expected [23,10,43,7,12,42,50,3]): "; print(array, size);
    std::cout << std::endl;
```

```
    int steps1 = search(array, size, 42);

    std::cout << "Die Zahl 42 wurde nach  " << steps1 << " Schritten gefunden (3 erwartet)." << std::endl;
    std::cout << "Found 42 after  " << steps1 << " steps (expected 3)." << std::endl;
    std::cout << std::endl;

    multiply(array, size);

    std::cout << "Die Zahlen des Arrays wurden verdoppelt (erwartet [46,20,86,14,24,84,100,6]): "; print(array, size);
    std::cout << "Doubled array (expected [46,20,86,14,24,84,100,6]): "; print(array, size);
    std::cout << std::endl;

    int steps2 = search(array, size, 46);

    std::cout << "Die Zahl 100 wurde nach  " << steps2 << " Schritten gefunden (erwartet 1)." << std::endl;
    std::cout << "Found 100 after  " << steps2 << " steps (expected 1)." << std::endl;
    std::cout << std::endl;

    std::cout << "Konvertiere Array in ein String (erwartet [46,20,86,14,24,84,100,6]): " << to_string(array, size) << std::endl;
    std::cout << "Converted array to string (expected [46,20,86,14,24,84,100,6]): " << to_string(array, size) << std::endl;
    std::cout << std::endl;

    free(array);

    std::cout << "Das Array wurde freigegeben (erwartet 0x0): " << array << std::endl;
    std::cout << "The array was freed (expected 0x0): " << array << std::endl;
    std::cout << std::endl;

    return 0;
}
```

search.cpp

```
#include "search.hpp"

/*
 * HINWEIS:
 *
 * Details zu den Funktionen gibt es in der Header-Datei. Der zu implementierende
 * Algorithmus der Suche ist in der Aufgabenbeschreibung gegeben.
 *
 * REMARK:
 *
 * The function details are given in the header file and the task description.
 *   In the task description, you will find the description of the search algorithm
 *   as well.
 */

/**
 * @brief Erzeugt ein Array der uebergebenen Groesse und initialisiert die Werte mit -1.
 *
 * @param size Die Groesse des zu erzeugenden Arrays.
 * @return int* Das erzeugte Array
 * @throws std::invalid_argument Wenn die angegebene Größe kleiner oder gleich 0 ist
 *
 *
 * @brief Creates an array of the given size and initializes the values with -1.
 *
 * @param size The size of the array.
 * @return int* The newly created array.
 * @throws std::invalid_argument If the given size is equal or less to 0.
 *
 */
int *create(int size) {
    if (size <= 0) throw std::invalid_argument("Zu kleine size");

    int * arr = new int[size];
    for (int i = 0; i < size; i++) arr[i] = -1;
    return arr;
}

void set(int *&array, int size, int index, int value) {
    if (index >= size || index < 0) throw std::invalid_argument("Index out of bounds");
    array[index] = value;
}
/*
Initialisiere Variable "index" mit 0.
Initialisiere Variable "anzahl" mit 0.

Solange "index" kleiner "size" {
  Erhöhe "anzahl" um 1.
  Wenn der Eintrag des Arrays an der Stelle "index" der gesuchte Wert ist, dann gib "anzahl" zurück.
  Wenn der Eintrag des Arrays an der Stelle "index" größer als "value" ist, dann weise "index" das Ergebnis von ((index + 1) · 2 - 1) zu.
  Anderenfalls weise "index" das Ergebnis von ((index + 1) · 2) zu.
}

Wirf die Exception "std::logic_error".
*/
int search(int *&array, int size, int value) {
    int index =0, anzahl = 0;
    while (index < size) {
        anzahl++;
        if (array[index] == value) return anzahl;
        if (array[index] > value) index = (index +1)*2 -1;
        else index = (index +1)*2;
    }
```

```
    throw std::logic_error("Not found");
}

/**
 * @brief Multipliziert alle Werte des eingegebenen Arrays mit zwei.
 *
 * @param array Das zu bearbeitende Array.
 * @param size Größe des Arrays.
 */
void multiply(int *&array, int size) {
    for (int i =0; i < size; i++) array[i] *= 2;
}

std::string to_string(int *&array, int size) {
    std::string str = "[";
    for (int i =0; i <size; i++) {
        str += std::to_string(array[i]);
        if (i != size -1) str += ",";
    }

    return str + "]";
}

void free(int *&array) {
    delete array;
    array = 0x0;
}
```

Bewertung mit Unittests

Testname	Assertions	Bestanden	Punkte
createArrayWithInvalidSize	2	2	1.00
createArrayWithPositiveSize	6	6	1.00
freeAndCheckPointer	1	0	0.00
multiplyOnArrayWithSingleElement	2	2	1.00
multiplyOnEmptyArray	2	2	1.00
multiplyOnFilledArray	5	5	1.00
multiplyOnFilledArray2	7	7	1.00
multiplyOnFilledArray3	4	4	1.00
multiplyOnFilledArray4	7	7	1.00
searchOnEmptyArray	1	1	1.00
searchOnFilledArrayLeftOut	1	1	1.00
searchOnFilledArrayRightOut	1	1	1.00
searchOnFilledArraySearch1	1	1	1.00
searchOnFilledArraySearch2	1	1	1.00
searchOnFilledArraySearch3	1	1	1.00
searchOnFilledArraySearch4	1	1	1.00
searchOnFilledArraySearch5	1	1	1.00
setAndCheckValues	4	4	1.00
setWithInvalidIndex	2	2	1.00
setWithMaxIndex	2	2	1.00
setWithMidIndex	2	2	1.00
setWithZeroIndex	2	2	1.00
toStringOnEmptyArray	1	1	1.00
toStringWithArrayOfLengthOne	1	1	1.00
toStringWithArrayOfLengthTwoAndMore	2	2	1.00

Auswertungsbericht für 552864

Erstellt am 17.2.2023 um 09:54:32 Uhr

Auswertungsbericht für 552864

Aufgabe 13

25.0 / 25 Punkten

Aufgabenstellung

Motivation

Eine Prioritätswarteschlange (Priority Queue) dient dazu, Einträge zu verwalten, die entsprechend einer zuvor vorgegebenen Priorität wieder abgerufen werden können. Dies wird zum Beispiel mit einer `std::priority_queue` aus der Standard-Bibliothek realisiert. Eine ähnliche Funktionalität soll im Folgenden realisiert werden.

Konzept

Die Einträge (Prioriäten) in der Priority Queue werden als `int`-Werte dargestellt. Die Klasse `PriorityQueue` speichert alle Einträge in einem speziellen Graphen, einem binären Baum (siehe Abb. 1). Ein binärer Baum besteht, ähnlich einer Liste, aus einzelnen Elementen - sogenannten Knoten - welche die Einträge als Daten (Variable `data` speichern. Jeder Knoten besitzt in den Variablen `leftChild` und `rightChild` Zeiger auf die Nachfolgerknoten im Baum. Leere Nachfolgerknoten werden durch `nullptr` gekennzeichnet. Knoten ohne Nachfolger werden auch als `Blätter` des Baums bezeichnet. Für jeden Knoten gilt, dass beide Nachfolgerknoten immer einen kleineren Wert enthalten. Der oberste Knoten - der sogenannte `root`- oder `Wurzel`-Knoten - enthält somit immer den aktuell größten Eintrag in der Prioritätswarteschlange. Jeder Knoten enthält zudem einen Zeiger auf seinen Vorgänger in der Variablen `parent`. Im Wurzelknoten gilt `parent = nullptr`. Die Struktur des Baums wird beim Einfügen oder Entfernen von Elementen durch die Hilfsfunktionen `get_leaf`, `sift_up` und `sift_down` aufrecht erhalten.

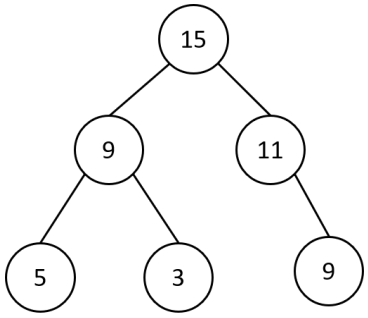


Abb. 1: Beispiel eines binären Baums zum Speichern einer Priority Queue

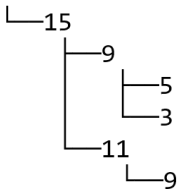


Abb. 2: Darstellung des Baums aus Abb.1 in der Programmausgabe

Hinweise zur Implementierung

Der Zeiger `root` auf die Wurzel des Baums entspricht `nullptr` im Falle einer leeren Priority Queue. Für einige der zu implementierenden Funktionen empfiehlt sich eine rekursive Implementierung. Korrekte, iterative Implementierungen werden aber ebenfalls mit voller Punktzahl bewertet. Die Aufgaben sind in etwa nach aufsteigender Schwierigkeit sortiert.

Aufgaben

In der Datei `priority_queue.hpp` sind Klassen für die Knoten des Baums und für die Priority Queue definiert. Die folgenden Memberfunktionen der Klasse `PriorityQueue` sind dort bereits implementiert:

a. `PriorityQueue()` ist der Konstruktor.

b. `PriorityQueue(const PriorityQueue &p)` ist der Copy-Constructor.

c. `bool get_rand()` gibt zufällig `false` oder `true` zurück. Nutzen Sie die Funktionen nur an notwendigen Stellen, damit die erwarteten Ergebnisse erzielt werden.

d. `void print_tree()` gibt eine graphische Darstellung des Baums in die Konsole aus (siehe Abb. 2).

e. `void pop()` entfernt mit Hilfe der `get_leaf`- und `sift_down`-Funktion das Element mit der höchsten Priorität aus der Priority Queue. Falls keine Elemente enthalten sind, wird eine `std::runtime_error`-Exception ausgelöst.

f. `void push(int item)` fügt mit Hilfe der `get_leaf`- und `sift_up`-Funktion ein Element in die Priority Queue ein.

g. `void sift_down(Node* node)` vertauscht den Eintrag des übergebenen Knotens `node` so lange mit dem größeren Nachfolger, bis beide Nachfolger kleiner sind oder keine weiteren Nachfolger existieren. Gilt `node=nullptr`, tut die Funktion nichts.

h. `~PriorityQueue()` gibt den gesamten allokierten Speicher frei.

Die Funktionsrümpfe weiterer Memberfunktionen sind in der Datei `priority_queue.cpp` verfügbar. Implementieren Sie dort die Funktionen:

a. `bool empty()`

liefert `true`, falls die Priority Queue leer ist. Sonst liefert die Funktion `false`.

b. `int top()`

liefert den höchsten Prioritätswert aus der Warteschlange zurück. Falls keine Elemente enthalten sind, wird eine `std::runtime_error`-Exception ausgelöst.

c. `void clear()`

leert den gesamten Baum. Anschließend soll `root=nullptr` gelten. Beachten Sie den Fall, dass die Liste bereits leer war.

d. `int size()`

liefert die Anzahl der enthaltenen Elemente, indem alle Elemente des Baums durchlaufen werden. Tipp: Diese Aufgabe lässt sich durch eine rekursive Funktion lösen. Die Struktur hierfür ist bereits vorgegeben. Sie brauchen nur die rekursive Funktion `int tree_size(Node* node)` zu implementieren.

e. `Node* get_leaf()`

gibt einen Zeiger auf ein beliebiges Blatt des Baums zurück. Dafür soll ausgehend vom Wurzelknoten so lange zufällig zu einem der Nachfolgerknoten gewechselt werden, bis ein Blatt erreicht wird. Für die zufällige Auswahl des Nachfolgerknotens soll die Funktion `get_rand` genutzt werden. Gibt diese Funktion `false` zurück, wird `leftChild` gewählt, andernfalls `rightChild`. Hat der Knoten nur einen Nachfolger, wird immer dieser als nächstes besucht und die `get_rand`-Funktion nicht aufgerufen. Falls der Baum leer ist, wird eine `std::runtime_error`-Exception ausgelöst.

f. `void sift_up(Node* node)`

vertauscht den Eintrag des übergebenen Knotens `node` so lange mit seinem Elternknoten, bis dieser einen größeren Eintrag enthält oder der Wurzelknoten erreicht ist. Der Wurzelknoten ist daran zu erkennen, dass `parent == nullptr` ist. Gilt `node == nullptr`, tut die Funktion nichts.

Folgende Dateien wurden Ihnen während der Prüfung zur Verfügung gestellt:

main.cpp

```
        #include "priority_queue.hpp"

int main()
{
    PriorityQueue q(0);
    PriorityQueue q0;
    std::cout << std::endl;

    std::cout << "Aktueller Baum:"  << std::endl;
    std::cout << "Current tree:"  << std::endl;
    q.print_tree();

    std::cout << std::endl;
    try {
        std::cout << "Aufruf von empty() (erwartet 0): " << q.empty() << std::endl;
    }
    catch(std::logic_error error) {
        std::cout << "  Funktion noch nicht implementiert." << std::endl;
    }
    try {
    std::cout << "Calling empty() (expected 0): " << q.empty() << std::endl;
    }
    catch(std::logic_error error) {
        std::cout << "  Function not yet implemented." << std::endl;
    }

    std::cout << std::endl;
    std::cout << "Aufruf von top() (erwartet 15):: " << q.top() << std::endl;
    std::cout << "Calling top() (expected 15): " << q.top() << std::endl;

    try{
        std::cout << std::endl;
        std::cout << "Rufe top() auf leerem Baum auf (erwarte runtime_error): " << std::endl;
        std::cout << "Calling top() on empty tree (expected runtime_error): " << std::endl;
        q0.top();
        std::cout << "  runtime_error wurde nicht geworfen." << std::endl;
        std::cout << "  runtime_error was not thrown." << std::endl;
     }
    catch(std::runtime_error error){
        std::cout << "  runtime_error wurde geworfen. " << std::endl;
        std::cout << "  runtime_error was thrown." << std::endl;
    }

    std::cout << std::endl;
    std::cout << "Aufruf von size() (erwartet 6): " << q.size() << std::endl;
    std::cout << "Calling size() (expected 6): " << q.size() << std::endl;

    q.clear();
    std::cout << std::endl;
    std::cout << "Aufruf von clear() (erwartet nullptr):" << (q.root == nullptr ? "nullptr" : "not nullptr")  << std::endl;
    std::cout << "Calling clear() (expected nullptr): " << (q.root == nullptr ? "nullptr" : "not nullptr") << std::endl;

    std::cout << std::endl;
    std::cout << "Rufe size() auf leerem Baum auf (erwartet 0): " << q.size() << std::endl;
    std::cout << "Calling size() on empty tree (expected 0):" << q.size() << std::endl;

    try{
        std::cout << std::endl;
        std::cout << "Rufe get_leaf() auf leerem Baum auf (erwartet runtime_error): "  << std::endl;
        std::cout << "Calling get_leaf() on empty tree (expedted runtime_error): " << std::endl;
        q.get_leaf();
        std::cout << "  runtime_error wurde nicht geworfen." << std::endl;
        std::cout << "  runtime_error was not thrown." << std::endl;    }
    catch(std::runtime_error error){
        std::cout << "  runtime_error wurde geworfen. " << std::endl;
        std::cout << "  runtime_error was thrown." << std::endl;
    }

    PriorityQueue q2(0);
    std::cout << std::endl;
    std::cout << "Aktueller (neuer) Baum:"  << std::endl;
    std::cout << "Current (new) tree:"  << std::endl;
    q2.print_tree();

    std::cout << std::endl;
    std::cout << "Rufe get_leaf() auf (erwartet 9): " << q2.get_leaf()->data << std::endl;
    std::cout << "Calling get_leaf() (expected 9): " << q2.get_leaf()->data << std::endl;

    std::cout << std::endl;
    std::cout << "Füge 20 durch den Aufruf von push(20) ein. Resultierender Baum:" << std::endl;
    std::cout << "Adding 20 to the tree, by calling push(20). Resulting tree::" << std::endl;

    try {
        q2.push(20);
        q2.print_tree();
    }
    catch(std::logic_error error) {
        std::cout << "  Funktion empty() noch nicht implementiert." << std::endl;
        std::cout << "  Function empty() not yet implemented." << std::endl;
    }

    std::cout << std::endl;
    std::cout << "Erwarteter Baum:" << std::endl;
    std::cout << "Expected tree:" << std::endl <<
        R"(└──20
    ├──9
    │   ├──5
```

```
    │   └──3
    └──15
        └──11
            └──9
    )";

    std::cout << std::endl;
    std::cout << "Entferne Wurzel durch Aufruf von pop(). Resultierender Baum:" << std::endl;
    std::cout << "Removing root by calling pop(). Resulting tree:" << std::endl;
    q2.pop();
    q2.print_tree();

    std::cout << std::endl;
    std::cout << "Erwarteter Baum:" << std::endl;
    std::cout << "Expected tree:" << std::endl <<
        R"(└──15
    ├──9
    │   ├──5
    │   └──3
    └──11
        └──9
    )";

    return 0;
}
```

priority_queue.cpp

```
#include <priority_queue.hpp>
#include <stdexcept>

bool PriorityQueue::empty() {
    throw std::logic_error("Not yet implemented.");
}

int PriorityQueue::top() {
    return -100;
}

void PriorityQueue::clear(){

}

int tree_size(Node* node){
    return -1;
}

int PriorityQueue::size(){
    return tree_size(root);
}

Node* PriorityQueue::get_leaf(){
    return new Node(-1);
}

void PriorityQueue::sift_up(Node* node){

}
```

priority_queue.hpp

```
#ifndef PRIORITYQUEUE_HPP
#define PRIORITYQUEUE_HPP

#include <stdexcept>
#include <iostream>

/**
  * @brief Knoten / Element eines Baums
  * @brief Node / element of the tree
  *
  */
class Node {
public:
    /**
     * @brief gespeicherter Prioritätswert
     * @brief stored priority
     *
     */
    int data;
    /**
     * @brief Zeiger auf Vorgänger (parent), linken Nachfolger (leftChild) und rechten Nachfolger (rightChild)
     * @brief Pointer to predecessor (parent), left successor (leftChild) and right successor (rightChild)
     *
     */
    Node *parent;
    Node *leftChild;
    Node *rightChild;

    /**
     * @brief Konstruktor
     * @brief Constructor
     *
     */
    Node(int data) {
        this->data = data;
        leftChild = nullptr;
        rightChild = nullptr;
        parent = nullptr;
    }
    /**
```

```
     * @brief Konstruktor mit Nachfolgern
     * @brief Constructor with children
     *
     */
    Node(int data, Node* leftChild, Node* rightChild){
        this->data = data;
        this->leftChild = leftChild;
        this->rightChild = rightChild;
        this->parent = nullptr;
        if(leftChild != nullptr)
            this->leftChild->parent = this;
        if(rightChild != nullptr)
            this->rightChild->parent = this;

    }
    /**
     * @brief Copy-Konstruktor
     * @brief Copy-Constructor
     *
     */
    Node(const Node& other){
        parent = nullptr;
        this->data = other.data;
        if(other.leftChild != nullptr){
            this->leftChild = new Node(*other.leftChild);
            this->leftChild->parent = this;
        }
        else{
            this->leftChild = nullptr;
        }
        if(other.rightChild != nullptr){
            this->rightChild = new Node(*other.rightChild);
            this->rightChild->parent = this;
        }
        else{
            this->rightChild = nullptr;
        }
    }
    /**
     * @brief Destruktor
     * @brief Destructor
     *
     */
    ~Node(){
        if(leftChild != nullptr)
            delete leftChild;
        if(rightChild != nullptr)
            delete rightChild;
    }


};

/**
     * @brief Prioritätswarteschlange
     * @brief Priority Queue
     *
     */
class PriorityQueue {
private:
    /**
     * @brief Einige nicht so zufällige Werte, damit der Algorithmus sich immer gleich verhält.
     * @brief Some not very random values, to allow the algorithm to be deterministic.
     */
    const int randoms[15] = {23,235,93,39,57,7,83,5,77,42,24,38,37,44,132};
public:
    /**
     * @brief Index für die Random-Werte.
     * @brief Index for the random values.
     */
    int index = 0;

public:
    /**
     * @brief Konstruktor
     * @brief Constructor
     *
     */
    PriorityQueue(){
        root = nullptr;
    }
    /**
     * @brief Konstruktor für Beispiel-Baum
     * @brief Constructor with exemplary tree
     *
     */
    PriorityQueue(int i) {
        Node* sixthNode = new Node(i+9);
        Node* fourthNode = new Node(i+5);
        Node* fifthNode = new Node(i+3);

        Node* secondNode = new Node(i+9,fourthNode,fifthNode);
        Node* thirdNode = new Node(i+11,nullptr,sixthNode);

        root = new Node(i+15,secondNode,thirdNode);
    }
    /**
     * @brief Copy-Konstruktor
     * @brief Copy-Constructor
     *
     */
    PriorityQueue(const PriorityQueue& other){
        if(other.root != nullptr){
            this->root = new Node(*other.root);
        }
    }
    /**
```

```
     * @brief Generiert zufällige bool-Werte
     * @brief Generates random bool-values
     *
     * @return zufällig true oder false
     * @return randomly true or false
     *
     */
    bool get_rand() {
        index = (index + 1) % 15;
        return (randoms[index] % 2) == 1;
    }
    /**
     * @brief Rekursive Funktion zur Ausgabe des Baums in die Konsole
     * @brief Recursive function to print the tree to the console
     *
     */
    void print_recursive(const std::string& prefix, const Node* node, bool isLeft) {
        if( node != nullptr )
        {
            std::cout << prefix;

            std::cout << (isLeft ? "├──" : "└──" );

            // print the value of the node
            std::cout << node->data << std::endl;

            // enter the next tree level - left and right branch
            print_recursive( prefix + (isLeft ? "│   " : "    "), node->leftChild, true);
            print_recursive( prefix + (isLeft ? "│   " : "    "), node->rightChild, false);
        }
    }
    /**
     * @brief Funktion zur Ausgabe des Baums
     * @brief Function for printing the tree
     *
     */
    void print_tree(){
        print_recursive("",root,false);
    }
    /**
     * @brief Entfernt den Wurzelknoten aus dem Baum und stellt die Baumordnung wieder her. Die Funktion verwendet @ref get_leaf und @ref sift_down.
     *          Konzeptuell wird das Wurzelelement entfernt und durch ein Blatt ersetzt. Dieser Knoten wird anschließend im Baum nach unten geschoben,
     *          bis die Baumstruktur wiederhergestellt ist.
     * @brief Removes the root node and restores the tree structue. The function uses @ref get_leaf and @ref sift_down.
     *          The concept is to remove the root node and replace it by a random leaf node. Afterwards, the inserted node is shifted down the tree
     *          until the tree structure is restored.
     *
     */
    void pop(){
        if(root == nullptr){
            throw std::runtime_error("");
        }

        if(root->leftChild == nullptr && root->rightChild == nullptr){
            delete root;
            root = nullptr;
            return;
        }

        //Nehme Blatt und entferne es | Take leaf and remove it from tree
        Node* leaf = get_leaf();
        //Extra Check für "leere" Aufgabe | Extra check for "empty" exercise.
        if(leaf->data == -1){
            root->data=-1;
            delete leaf;
            return;
        }
        Node* leaf_parent = leaf->parent;
        if(leaf_parent->leftChild == leaf){
            leaf_parent->leftChild = nullptr;
        }
        else{
            leaf_parent->rightChild = nullptr;
        }

        //Ersetze Wurzel durch Blatt | Replace root by leaf.
        root->data = leaf-> data;
        delete leaf;

        //Schiebe neue Wurzel nach unten | Sift new root down.
        sift_down(root);
    }
    /**
     * @brief Fügt einen neuen Knoten ein. Die Funktion verwendet @ref get_leaf und @ref sift_down.
     *          Konzeptuell wird der neue Knoten an ein beliebiges Blatt angehängt. Dieser Knoten wird anschließend im Baum nach oben geschoben,
     *          bis die Baumstruktur wiederhergestellt ist.
     * @brief Removes the root node and restores the tree structue. The function uses @ref get_leaf and @ref sift_down.
     *          The concept is to append the new node to any leaf. Afterwards, the inserted node is shifted up the tree
     *          until the tree structure is restored.
     * @param item Priorität des einzufügenden Knotens
     * @param item Priority of the inserted node
     *
     */
    void push(int item){
        Node* newNode = new Node(item);
        if(root == nullptr){
            root = newNode;
            return;
        }

        //Anhängen an Blatt | Append to leaf node
        Node* leaf = get_leaf();
        //Extra Check für "leere" Aufgabe | Extra check for "empty" exercise.
        if(leaf->data == -1){
            root->data=-1;
            delete leaf;
            return;
```

```
        }
        if(get_rand()){
            leaf->rightChild = newNode;
        }
        else{
            leaf->leftChild = newNode;
        }
        newNode->parent = leaf;

        //Schiebe neuen Knoten nach oben | Sift new node up
        sift_up(newNode);
    }

    /**
     * @brief Schiebt den übergebenen Knoten im Baum so lange nach unten, bis beide
     *          Kinderknoten eine kleinere Priorität haben oder ein Blatt erreicht ist. Dies geschieht, indem der Knoten
     *          jeweils mit dem Kind mit größerer Priorität vertauscht wird. Ist der übergebene Zeiger leer,
     *          wird nichts getan.
     * @brief Shifts the given node down the tree until both child-nodes have a smaller priority
     *          compared to its own priority or a leaf node is reached. To do so, the node is always swapped with
     *          the child-node of larger priority. If the given node is empty, nothing is done.
     *
     * @param node Ein Zeiger, auf den zu verschiebenden Knoten
     * @param node A pointer to the node that is to be shifted
     */
    void sift_down(Node* node) {
        if(node != nullptr) {
            if(node->leftChild != nullptr && node->rightChild != nullptr){
                if(node->leftChild->data > node->rightChild->data && node->leftChild->data > node->data){
                    swap_data(node,node->leftChild);
                    sift_down(node->leftChild);
                }
                else if (node->rightChild->data > node->data){
                    swap_data(node,node->rightChild);
                    sift_down(node->rightChild);
                }
            }
            else if(node->leftChild != nullptr && node->leftChild->data > node->data){
                swap_data(node,node->leftChild);
                sift_down(node->leftChild);
            }
            else if(node->rightChild != nullptr && node->rightChild->data > node->data){
                swap_data(node,node->rightChild);
                sift_down(node->rightChild);
            }
        }
    }

    /**
     * @brief Tauscht die Daten der gegebenen Knoten.
     * @brief Swaps the data of the given nodes.
     */
    void swap_data(Node* node1, Node* node2){
        int temp = node1->data;
        node1->data = node2->data;
        node2->data = temp;
    }

    /**
     * @brief Destruktor (ruft @ref clear auf)
     * @brief Destructor (calls @ref clear)
     *
     */
    ~PriorityQueue(){
        if(root != nullptr)
            delete root;
        root = nullptr;
    }

    /**
     * @brief Prüft, ob die Prioritätswarteschlange leer ist.
     * @brief Checks whether the priority Queue is empty.
     *
     * @return true, falls die Warteschlange leer ist, und false sonst
     * @return true if the queue is empty and false otherwise
     */
    bool empty();
    /**
     * @brief Gibt den größten Prioritätswert in der Warteschlange zurück.
     * @brief Returns the largest entry in the priority Queue
     *
     * @return Der größte Wert in der Warteschlange
     * @return The largest value in the queue
     *
     * @throws std::runtime_error Wenn die Warteschlange leer ist.
     * @throws std::runtime_error If the queue is empty.
     */
    int top();
    /**
     * @brief Leert die Warteschlange.
     * @brief Removes all elements from the queue.
     *
     * @note Beachte den Fall, dass der Baum bereits leer ist.
     * @note Be aware of an already empty tree.
     *
     * */
    void clear();
    /**
     * @brief Bestimmt die Größe des Baums
     * @brief Determines the size of the tree.
     *
     * @return Die Anzahl an Elementen im Baum.
     * @return The number of entries in the tree.
     *
     * @note Es empfiehlt sich eine rekursive Implementierung
     * @note A recursive implementation is recommended.
     */
```

```
    int size();
    /**
     * @brief Findet ein beliebiges Blatt im Baum, indem ausgehend vom Wurzelknoten
     *          zufällig zu Nachfolgern gewechselt wird, bis ein Blatt erreicht ist.
     *          Für die zufällige Auswahl des Nachfolgers soll die Funktion @ref get_rand
     *          genutzt werden. Gibt die Funktion false zurück, wird zum linken Kind
     *          gewechselt, andernfalls zum rechten. Hat der Knoten nur einen Nachfolger,
     *          wird immer dieser als nächstes ausgewählt und die get_rand Funktion nicht
     *          aufgerufen.
     * @brief Finds an arbitrary leaf in the tree: starting from the root node a successor
     *          is visited until a leaf node is reached. The random choice of the next
     *          successor is done using the @get_rand function. If the function returns
     *          false, the left successor is visited, otherwise the right child is visited
     *          If there is only one successor, this one is always visited without calling
     *          the get_rand function.
     *
     * @return Einen Zeiger auf ein beliebiges Blatt im Baum.
     * @return A pointer to an arbitrary leaf of the tree.
     *
     * @throws std::runtime_error Wenn die Warteschlange leer ist.
     * @throws std::runtime_error If the queue is empty.
     */
    Node* get_leaf();
    /**
     * @brief Schiebt den übergebenen Knoten im Baum so lange nach oben, bis der
     *          Elternknoten eine höhere Priorität hat oder die Wurzel erreicht ist.
     *          Ist der übergebene Zeiger leer, wird nichts getan.
     * @brief Shifts the given node up the tree until the parent node has a larger priority
     *          compared to its own priority or the root node is reached. If the given node is
     *          empty, nothing is done.
     *
     * @param node Ein Zeiger auf den zu verschiebenden Knoten
     * @param node A pointer to the node that is to be shifted
     */
    void sift_up(Node* node);

public:
    /**
     * @brief Ein Zeiger auf das erste Element im Baum. Dieses ist das Element mit der höchsten Priorität.
     * @brief A pointer to the first element in the tree. This element has the largest priority.
     *
     * */
    Node* root;
};


#endif //PRIORITYQUEUE_HPP
```

Folgende Dateien haben Sie in der Prüfung bearbeitet bzw. erstellt:

main.cpp

```
        #include "priority_queue.hpp"

int main()
{
    PriorityQueue q(0);
    PriorityQueue q0;
    std::cout << std::endl;

    std::cout << "Aktueller Baum:"  << std::endl;
    std::cout << "Current tree:"  << std::endl;
    q.print_tree();

    std::cout << std::endl;
    try {
        std::cout << "Aufruf von empty() (erwartet 0): " << q.empty() << std::endl;
    }
    catch(std::logic_error error) {
        std::cout << "  Funktion noch nicht implementiert." << std::endl;
    }
    try {
    std::cout << "Calling empty() (expected 0): " << q.empty() << std::endl;
    }
    catch(std::logic_error error) {
        std::cout << "  Function not yet implemented." << std::endl;
    }

    std::cout << std::endl;
    std::cout << "Aufruf von top() (erwartet 15):: " << q.top() << std::endl;
    std::cout << "Calling top() (expected 15): " << q.top() << std::endl;

    try{
        std::cout << std::endl;
        std::cout << "Rufe top() auf leerem Baum auf (erwarte runtime_error): " << std::endl;
        std::cout << "Calling top() on empty tree (expected runtime_error): " << std::endl;
        q0.top();
        std::cout << "  runtime_error wurde nicht geworfen." << std::endl;
        std::cout << "  runtime_error was not thrown." << std::endl;
     }
    catch(std::runtime_error error){
        std::cout << "  runtime_error wurde geworfen. " << std::endl;
        std::cout << "  runtime_error was thrown." << std::endl;
    }

    std::cout << std::endl;
    std::cout << "Aufruf von size() (erwartet 6): " << q.size() << std::endl;
    std::cout << "Calling size() (expected 6): " << q.size() << std::endl;

    q.clear();
    std::cout << std::endl;
```

```
    std::cout << "Aufruf von clear() (erwartet nullptr):" << (q.root == nullptr ? "nullptr" : "not nullptr")  << std::endl;
    std::cout << "Calling clear() (expected nullptr): " << (q.root == nullptr ? "nullptr" : "not nullptr") << std::endl;

    std::cout << std::endl;
    std::cout << "Rufe size() auf leerem Baum auf (erwartet 0): " << q.size() << std::endl;
    std::cout << "Calling size() on empty tree (expected 0):" << q.size() << std::endl;

    try{
        std::cout << std::endl;
        std::cout << "Rufe get_leaf() auf leerem Baum auf (erwartet runtime_error): "  << std::endl;
        std::cout << "Calling get_leaf() on empty tree (expedted runtime_error): " << std::endl;
        q.get_leaf();
        std::cout << "  runtime_error wurde nicht geworfen." << std::endl;
        std::cout << "  runtime_error was not thrown." << std::endl;    }
    catch(std::runtime_error error){
        std::cout << "  runtime_error wurde geworfen. " << std::endl;
        std::cout << "  runtime_error was thrown." << std::endl;
    }

    PriorityQueue q2(0);
    std::cout << std::endl;
    std::cout << "Aktueller (neuer) Baum:"  << std::endl;
    std::cout << "Current (new) tree:"  << std::endl;
    q2.print_tree();

    std::cout << std::endl;
    std::cout << "Rufe get_leaf() auf (erwartet 9): " << q2.get_leaf()->data << std::endl;
    std::cout << "Calling get_leaf() (expected 9): " << q2.get_leaf()->data << std::endl;

    std::cout << std::endl;
    std::cout << "Füge 20 durch den Aufruf von push(20) ein. Resultierender Baum:" << std::endl;
    std::cout << "Adding 20 to the tree, by calling push(20). Resulting tree::" << std::endl;

    try {
        q2.push(20);
        q2.print_tree();
    }
    catch(std::logic_error error) {
        std::cout << "  Funktion empty() noch nicht implementiert." << std::endl;
        std::cout << "  Function empty() not yet implemented." << std::endl;
    }

    std::cout << std::endl;
    std::cout << "Erwarteter Baum:" << std::endl;
    std::cout << "Expected tree:" << std::endl <<
        R"(└──20
    ├──9
    │   ├──5
    │   └──3
    └──15
        └──11
            └──9
    )";

    std::cout << std::endl;
    std::cout << "Entferne Wurzel durch Aufruf von pop(). Resultierender Baum:" << std::endl;
    std::cout << "Removing root by calling pop(). Resulting tree:" << std::endl;
    q2.pop();
    q2.print_tree();

    std::cout << std::endl;
    std::cout << "Erwarteter Baum:" << std::endl;
    std::cout << "Expected tree:" << std::endl <<
        R"(└──15
    ├──9
    │   ├──5
    │   └──3
    └──11
        └──9
    )";

    return 0;
}
```

priority_queue.cpp

```
#include <priority_queue.hpp>
#include <stdexcept>

bool PriorityQueue::empty() {
    if (root == nullptr) return 1;
    return 0;
    throw std::logic_error("Not yet implemented.");
}
/*
//liefert den höchsten Prioritätswert aus der Warteschlange zurück.
Falls keine Elemente enthalten sind, wird eine std::runtime_error-Exception ausgelöst.
*/
int PriorityQueue::top() {
    if (empty()) throw std::runtime_error("Queue empty");
    return root->data;
}

void PriorityQueue::clear(){
    delete root;
    root = nullptr;
}

int tree_size(Node* node){
    if (node == nullptr) return 0;
    return tree_size(node->rightChild) + tree_size(node->leftChild) +1;
}

int PriorityQueue::size(){
```

```
    return tree_size(root);
}

/*
gibt einen Zeiger auf ein beliebiges Blatt des Baums zurück.
Dafür soll ausgehend vom Wurzelknoten so lange zufällig zu einem der Nachfolgerknoten gewechselt werden,
bis ein Blatt erreicht wird.
Für die zufällige Auswahl des Nachfolgerknotens soll die Funktion get_rand genutzt werden.
Gibt diese Funktion false zurück, wird leftChild gewählt, andernfalls rightChild.
Hat der Knoten nur einen Nachfolger,
wird immer dieser als nächstes besucht und die get_rand-Funktion nicht aufgerufen.
Falls der Baum leer ist, wird eine std::runtime_error-Exception ausgelöst.
*/
Node* PriorityQueue::get_leaf(){
    if (empty()) throw std::runtime_error("Empty queue");
    Node * curr = root;
    while (tree_size(curr) > 1) {
        if (curr->rightChild == nullptr) {curr = curr->leftChild; continue;}
        else if(curr->leftChild == nullptr) {curr = curr->rightChild; continue;}
        if (get_rand()) curr = curr->rightChild;
        else curr = curr->leftChild;
    }
    return curr;
}
/*
vertauscht den Eintrag des übergebenen Knotens node so lange
mit seinem Elternknoten, bis dieser einen größeren Eintrag
enthält oder der Wurzelknoten erreicht ist.
Der Wurzelknoten ist daran zu erkennen, dass parent == nullptr
ist. Gilt node==nullptr, tut die Funktion nichts.
*/

void PriorityQueue::sift_up(Node* node){
    if (node==nullptr) return;

    while (node->parent != nullptr) {
        if (node->parent->data >= node->data) return;
        int hilf = node->data;
        node->data = node->parent->data;
        node->parent->data = hilf;
        node = node->parent;
    }
}
```

Bewertung mit Unittests

Testname	Assertions	Bestanden	Punkte
checkClearNullptr	1	1	1.00
checkClearNullptrMemory	1	1	1.00
checkClearViaDestructor	1	1	1.00
checkDoubleClear	1	1	1.00
checkEmptyOnDefault	1	1	1.00
checkEmptyOnDefaultPQueue	1	1	1.00
checkNormalTop	1	1	1.00
checkNotEmpty	1	1	1.00
checkOneNodeNotEmpty	1	1	1.00
checkSizeExample	1	1	1.00
checkSizeNewTree	1	1	1.00
checkSizeNull	1	1	1.00
checkSizeOneNode	1	1	1.00
checkTopExample	1	1	1.00
checkTopOnDefaultPQueue	1	1	1.00
checkTopOther1	1	1	1.00
checkTopOther42	1	1	1.00
checkTopThrows	1	1	1.00
getLeafEmpty	1	1	1.00
getLeafExample	2	2	1.00
getLeafExampleNewTree	2	2	1.00
siftUpExample	1	1	1.00
siftUpExampleNoSift	1	1	1.00
siftUpExampleNotToRoot	1	1	1.00
siftUpOnlySift	1	1	1.00