

Praktikumstermin Nr. 10, INF: Klassen

Abgabe im GIP-INF Praktikum der Woche 18.12.-22.12.2023.

(Pflicht-) Aufgabe INF-10.01: Klasse `MyCanvas`

Ein Canvas ist eine Zeichenfläche / Bildschirmfläche mit x- und y-Koordinaten. Der Canvas in dieser Aufgabe ist ein „Text-Canvas“, d.h. es wird ein `char` Zeichen an jeder x,y-Position gespeichert und der Canvas-Inhalt kann auf `cout` (`stdout`) ausgegeben werden.

Der Canvas `MyCanvas` in dieser Aufgabe speichert seine `char` Werte in einem eindimensionalen (!) Array, d.h. die Koordinate `[x][y]` wird auf die Arrayposition `[y * size_x + x]` abgebildet (und umgekehrt). Eine Abweichung von diesem Prinzip (also insbesondere die Nutzung eines zweidimensionalen Arrays) ist nicht gestattet!

Zwei-dimensional:

`size_x == 4, size_y == 3`

x = 0, y = 0	x = 1, y = 0	x = 2, y = 0	x = 3, y = 0
x = 0, y = 1	x = 1, y = 1	x = 2, y = 1	x = 3, y = 1
x = 0, y = 2	x = 1, y = 2	x = 2, y = 2	x = 3, y = 2

Ein-dimensional:

Indexposition im ein-dimensionalen Array: `index = y * size_x + x`

index == 0	1	2	3	4	5	6	7	8	9	10	11
x = 0, y = 0	x = 1, y = 0	x = 2, y = 0	x = 3, y = 0	x = 0, y = 1	x = 1, y = 1	x = 2, y = 1	x = 3, y = 1	x = 0, y = 2	x = 1, y = 2	x = 2, y = 2	x = 3, y = 2

Das Array zum Canvas soll auf dem Heap per „Array-new“ `new[]` angelegt werden und das `MyCanvas` Objekt speichert einen Pointer auf die Startadresse dieses Arrays auf dem Heap.

Legen Sie in Visual Studio Code ein neues Projekt an und übernehmen Sie die Dateien `MyCanvas.h`, `test_MyCanvas.cpp`, `main.cpp`,

`gip_mini_catch_v_2_3 .h` und `gip_mini_catch_heap_delete .h` aus Ilias. Legen Sie zusätzlich die Datei `MyCanvas.cpp` an.

Programmieren Sie in den entsprechenden Dateien die Klasse `MyCanvas` gemäß den folgenden Anforderungen (dabei **alle (!) Methoden in der .cpp Datei programmieren**, in die Klassendefinition in der Headerdatei nur die Prototypen; wir werden bei der Abgabe prüfen, ob Sie sich an diese Vorgabe gehalten haben):

Jedes Objekt der Klasse `MyCanvas` besitze zwei `unsigned int` Attribute `size_x` und `size_y` für die x- und y-Größe der Zeichenfläche (canvas) (Positionen 0 bis `size_x - 1` und 0 bis `size_y - 1`) sowie ein `char*` Attribut `canvas_array_ptr`. Alle diese Attribute sollen gegen Zugriff von außen geschützt werden, aber in möglichen späteren abgeleiteten Klassen zugreifbar sein (auch wenn es in diesem Praktikum noch keine abgeleiteten Klassen geben wird). Programmieren Sie die Getter und Setter für jedes dieser Attribute.

Der `MyCanvas` Konstruktor soll zwei `unsigned int` Parameter nehmen und die beiden `size...` Attribute entsprechend setzen. Der Konstruktor soll in seiner Initialisierungsliste das eindimensionale Array auf dem Heap allozieren mit Arraygröße passend zu den beiden Parameterwerten. Ferner soll im Rumpf des Konstruktors die Methode ...

```
void MyCanvas::init()
```

... aufgerufen werden, welche alle Einträge des Arrays mit dem Buchstaben „Punkt“ `'.'` initialisiert.

Da die `MyCanvas` Objekte mit dem Array auf dem Heap eine interne Ressource verwalten, müssen gemäß der „Rule of 3“ auch der Destruktor, der Copy Konstruktor und der Assignment Operator für die Klasse `MyCanvas` von ihnen programmiert werden.

Die von außen aufrufbare Methode ...

```
void MyCanvas::set(unsigned int x, unsigned int y, char c)
```

... schreibe den Buchstaben `c` an die Stelle des internen Arrays, welche der logischen Position `x, y` entspricht.

Die von außen aufrufbare konstante Methode ...

```
char MyCanvas::get(unsigned int x, unsigned int y) const
```

... liefere den Buchstaben zurück, welcher sich an der logischen Position `x, y` befindet.

Die von außen aufrufbare konstante Methode ...

```
std::string MyCanvas::to_string() const
```

... liefere den String zurück, welcher der textuellen Darstellung des Canvas entspricht, sprich die Buchstaben aller Zeilen aneinandergereiht mit

jeweils einem Zeilenumbruch-Zeichen '`\n`' am Ende der Zeichen jeder Zeile (also im Resultatstring enthalten).

Die von außen aufrufbare konstante Methode ...

```
void MyCanvas::print() const
```

... soll den `to_string()` Wert auf den Bildschirm ausgeben, mit einem weiteren Zeilenumbruch danach.

Das Hauptprogramm in `main.cpp` startet die Unit Tests und gibt einen Canvas mit dem Text `GIP-INF` aus, gefolgt von einer Leerzeile.

Testlauf (keine Benutzereingaben):

```
Alle Tests erfolgreich (69 REQUIREs in 5 Test Cases)
```

```
.....
```

```
..GIP-INF..
```

```
.....
```

```
Drücken Sie eine beliebige Taste . . .
```

(Pflicht-) Aufgabe INF-10.02: Klasse `MyRectangle`

Legen Sie in Visual Studio Code ein neues Projekt an und in dieses Projekt dann die Vorgabedateien aus Ilias einkopieren. Zusätzlich leere Dateien `MyRectangle.h`, `MyRectangle.cpp` anlegen und die Lösung dort hinein programmieren. Kopieren Sie ferner die Dateien `MyCanvas.h` und `MyCanvas.cpp` aus der vorigen Aufgabe in dieses neue Projekt (diese werden hier erweitert, also kopieren!).

Programmieren Sie in den entsprechenden Dateien die Klasse `MyRectangle` gemäß den folgenden Anforderungen:

Jedes Objekt der Klasse `MyRectangle` besitze zwei `unsigned int` Attribute `x1` und `y1` für die linke obere Ecke des Rechtecks und zwei weitere `unsigned int` Attribute `x2`, `y2` für die rechte untere Ecke. Ferner speichere jedes `MyRectangle` Objekt einen Pointer `MyCanvas* canvas_ptr` auf ein `MyCanvas` Objekt.

Alle diese Attribute sollen gegen Zugriff von außen geschützt werden, aber Zugriff aus abgeleiteten Klassen soll erlaubt sein.

Programmieren Sie Getter und Setter für jedes dieser Attribute.

(Pflicht-) Aufgabe INF-10.03: Binärer Suchbaum mittels Objekt-Orientierter Programmierung

Diese Aufgabe hatte viele Ähnlichkeiten mit der ersten Aufgabe des vorigen Praktikums ...

Im Rahmen dieser Aufgabe sollen keine `namespaces` verwendet werden.

Programmieren Sie in einer Datei `BaumKnoten.h` die Klasse `BaumKnoten`. Jedes `BaumKnoten` Objekt soll in seinem Attribut `data` einen `int` Wert als Nutzdatenwert speichern können. Außerdem sollen in zwei Attributen `BaumKnoten* links` und `BaumKnoten* rechts` jeweils Pointer auf mögliche Kindknoten gespeichert werden (oder der `nullptr` Wert, falls kein entsprechender Kindknoten existiert). Alle diese Attribute sollen von außerhalb der Klasse nicht sichtbar sein.

Programmieren Sie die entsprechenden Getter und Setter für diese Attribute. Benennen Sie die Getter und Setter mit `get_<attributname>()` und `set_<attributname>()`, also z.B. `get_data()` und `set_data()`. Die Getter und Setter sollen natürlich von außerhalb der Klasse aufrufbar sein.

Programmieren Sie für die Klasse `BaumKnoten` einen Konstruktor, der einen `int` und zwei `BaumKnoten*` Parameter hat und diese drei Parameterwerte mittels Initialisierungsliste in die entsprechenden Attribute des Objekts überträgt.

Die Klasse `BaumKnoten` besitze ferner eine von außerhalb aufrufbare Methode `ausgeben()`, welche einen Parameter `unsigned int tiefe` habe. Im Rahmen der Klassendefinition in der Headerdatei soll der Prototyp dieser Methode definiert werden. Die Methode an sich soll in einer Datei `BaumKnoten.cpp` programmiert werden.

Programmieren Sie dann in einer Datei `BinaererSuchbaum.h` die Klasse `BinaererSuchbaum`.

Objekte dieser Klasse sollen ein von außen nicht sichtbares Attribut `root` vom Typ `BaumKnoten*` haben, welches auf den Wurzelknoten des Baumes verweist bzw. den `nullptr` Wert hat, wenn der Baum leer ist.

Programmieren Sie für das Attribut `root` einen Getter `get_root()`.

Der Konstruktor der Klasse sei parameterlos.

Die Klasse habe zwei von außen aufrufbare Methoden `ein fuegen()` und `ausgeben()`. Die Methode `ausgeben()` sei parameterlos, während `ein fuegen()` einen `int` Wert als Parameter nimmt, der dann in den Baum eingefügt wird (falls dort nicht schon vorhanden). Programmieren Sie die beiden Methoden in einer Datei `BinaererSuchbaum.cpp` und geben Sie in der Headerdatei (innerhalb der Klassendefinition) nur den Prototypen dieser Methoden an.

Legen Sie im Projekt eine leere Headerdatei `gip_mini_catch.h` an und kopieren Sie den Inhalt der in Ilias gegebenen gleichnamigen Datei in diese Headerdatei.

Legen Sie im Projekt eine Datei `test_binaerer_suchbaum_klasse.cpp` an und kopieren Sie den Inhalt der in Ilias gegebenen gleichnamigen Datei in diese Datei.

Legen Sie im Projekt eine Datei `suchbaum_klasse_main.cpp` an und kopieren Sie den Inhalt der in Ilias gegebenen gleichnamigen Datei in diese Datei.

Testläufe (Benutzereingaben sind unterstrichen):

Alle Tests erfolgreich (71 REQUIRES in 7 Test Cases)

Leerer Baum.

Naechster Wert (99 beendet): ? 10

Naechster Wert (99 beendet): ? 4

Naechster Wert (99 beendet): ? 6

Naechster Wert (99 beendet): ? 15

Naechster Wert (99 beendet): ? 13

Naechster Wert (99 beendet): ? 12

Naechster Wert (99 beendet): ? 15

Naechster Wert (99 beendet): ? 20

Naechster Wert (99 beendet): ? 11

Naechster Wert (99 beendet): ? 15

Naechster Wert (99 beendet): ? 99

++++20

++15

++++13

++++++12

+++++++11

10

++++6

++4

Drücken Sie eine beliebige Taste . . .

Alle Tests erfolgreich (71 REQUIRES in 7 Test Cases)

Leerer Baum.

Naechster Wert (99 beendet): ? 3

Naechster Wert (99 beendet): ? 3

Naechster Wert (99 beendet): ? 3

Naechster Wert (99 beendet): ? 2

Naechster Wert (99 beendet): ? 99

3

++2

Drücken Sie eine beliebige Taste . . .

Alle Tests erfolgreich (71 REQUIRES in 7 Test Cases)

Leerer Baum.

Naechster Wert (99 beendet): ? 99

Leerer Baum.

Drücken Sie eine beliebige Taste . . .
