21. März 2025

FDS2 - Übung 3

Tim Peko

SS 2025

Inhaltsverzeichnis

1. Beispiel 1: ADT “Liste”

1.1. Lösungsansatz

Die Implementierung der Liste erfolgte als einfach verkettete Liste. Dazu gibt es eine vorgegebene `node_t` Struktur, die einen Wert und einen Zeiger auf den nächsten Knoten enthält.

Diese Struktur wird in der `slist` Klasse verwendet, die die geforderten Operationen bereitstellt.

1.2. Testfälle

Die implementierte Liste wurde mit verschiedenen Testfällen geprüft. Diese Testfälle sind in der `main` Funktion zu finden. Ein Programmstart führt zu folgendem Output:

```
Running testcases...

Testing push operations: push_back(3), push_front(1), push_back(5)
list = [1, 3, 5]
list.size() = 3
list.empty() = false

Testing 'at(1, value)':
list[1] = 3

Testing 'contains(...)':
list.contains(3) = true
list.contains(7) = false

Testing 'insert_sorted(...)': insert_sorted(2), insert_sorted(4)
list = [1, 2, 3, 4, 5]

Testing 'apply(print_value)':
1 2 3 4 5

Testing 'apply(double_value)':
list = [2, 4, 6, 8, 10]

Testing 'pop_front(...)':
list.pop_front() = 2

Testing 'pop_back(...)':
list.pop_back() = 10

list = [4, 6, 8]

Testing 'copy constructor':
list2 = [4, 6, 8]

Testing 'equal(list, list2)':
list.equal(list2) = true
list2.push_back(10); list.equal(list2) = false

Testing 'count(...)' and 'remove_all(...)':
list3 = [5, 1, 5, 3, 5]
list3.count(5) = 3
list3.remove_all(5) = 3
list3 = [1, 3]

Testing 'clear()':
list.clear() = 3
list = []
list.empty() = true
```

Es lassen sich also folgende Testfälle isolieren:

1.2.1. Test der `push_front` und `push_back` Operationen

- Hinzufügen von Elementen am Anfang und Ende der Liste

Ergebnis: success

1.2.2. Test der Größe und Leerheit

- Abfrage der Listengröße und Überprüfung, ob die Liste leer ist

Ergebnis: success

1.2.3. Test der `at` Funktion

- Zugriff auf Elemente an bestimmten Positionen

Ergebnis: success

1.2.4. Test von `contains`

- Überprüfung, ob bestimmte Elemente in der Liste enthalten sind

Ergebnis: success

1.2.5. Test von `insert_sorted`

- Einfügen von Elementen in eine sortierte Liste

Ergebnis: success

1.2.6. Test der `apply` Funktion

- Anwenden einer Funktion zum Ausgeben der Elemente
- Anwenden einer Funktion zum Verdoppeln der Elemente

Ergebnis: success

1.2.7. Test der `pop_front` und `pop_back` Operationen

- Entfernen von Elementen am Anfang und Ende der Liste

Ergebnis: success

1.2.8. Test des Kopierkonstruktors

- Erstellen einer Kopie der Liste
- Überprüfung der Gleichheit mit `equal`

Ergebnis: success

1.2.9. Test von `count` und `remove_all`

- Zählen der Vorkommen eines Elements
- Entfernen aller Vorkommen eines Elements

Ergebnis: success

1.2.10. Test der `clear` Funktion

- Löschen aller Elemente in der Liste

Ergebnis: success

2. Beispiel 2: Dynamic Array

2.1. Lösungsansatz

Das Dynamic Array wird durch ein zweidimensionales Array gestützt. Dieses zweidimensionale Array besteht aus einer festen Anzahl von Spalten (`m_cols = 20`) und einer variablen Anzahl von Zeilen (`m_rows`), die abhängig von der Größe des Arrays berechnet wird.

Zusätzlich wird eine `m_max_size` Variable gespeichert, die die maximale Größe des Arrays speichert. Diese wird benötigt, um die Gültigkeit von Indizes zu überprüfen.

2.2. Testfälle

Auch hier wurden verschiedene Testfälle geschrieben, um die Funktionalität des Dynamic Arrays zu überprüfen. Sie sind auch hier in der `main` Funktion zu finden. Ein Programmstart führt zu folgendem Output:

```
Running testcases...

Testing default constructor:
DynArray(uninitialized)

Testing 'initialize(10)' method:
DynArray(capacity= 20)[
    0 => ""
    1 => ""
    2 => ""
    3 => ""
    4 => ""
    5 => ""
    6 => ""
    7 => ""
    8 => ""
    9 => ""
]

Testing 'set' method: set(0, 'First'), set(1, 'Second'), set(2, 'Third'), set(9,
'Last')
DynArray(capacity= 20)[
    0 => "First"
    1 => "Second"
    2 => "Third"
    3 => ""
    4 => ""
    5 => ""
    6 => ""
    7 => ""
    8 => ""
    9 => "Last"
]

Testing 'get' method: get(1), get(5), get(9)
arr1.get(1) = Second
arr1.get(5) =
arr1.get(9) = Last

Testing 'at' method: at(5) = 'Fifth'
DynArray(capacity= 20)[
```

```
    0 => "First"
    1 => "Second"
    2 => "Third"
    3 => ""
    4 => ""
    5 => "Fifth"
    6 => ""
    7 => ""
    8 => ""
    9 => "Last"
]

Testing invalid positions: set(20, 'Invalid')
arr1.set(20, 'Invalid') = false
arr1.get(20) =

Testing constructor with size: arr2(5)
DynArray(capacity= 20)[
    0 => "A"
    1 => "B"
    2 => "C"
    3 => "D"
    4 => "E"
]

Testing 'get_max_size()' method:
arr1.get_max_size() = 10
arr2.get_max_size() = 5

Testing 'clear()' method:
DynArray(uninitialized)
arr2.get_max_size() after clear = 0

Testing large array: arr3(30)
DynArray(capacity= 40)[
    0 => "Item 0"
    1 => "Item 1"
    2 => "Item 2"
    3 => "Item 3"
    4 => "Item 4"
    5 => "Item 5"
    6 => "Item 6"
    7 => "Item 7"
    8 => "Item 8"
    9 => "Item 9"
    10 => "Item 10"
    11 => "Item 11"
    12 => "Item 12"
    13 => "Item 13"
    14 => "Item 14"
    15 => "Item 15"
    16 => "Item 16"
    17 => "Item 17"
    18 => "Item 18"
    19 => "Item 19"
    20 => "Item 20"
```

```
    21 => "Item 21"
    22 => "Item 22"
    23 => "Item 23"
    24 => "Item 24"
    25 => "Item 25"
    26 => "Item 26"
    27 => "Item 27"
    28 => "Item 28"
    29 => "Item 29"
]
```

Dabei handelt es sich konkret um folgende Testfälle:

2.2.1. Test des Standardkonstruktors

- Erstellung eines leeren, uninitialisierten Arrays

Ergebnis: success

2.2.2. Test der `initialize` Methode

- Angelegtes Array wird mit einer bestimmten Größe initialisiert

Ergebnis: success

2.2.3. Test der `set` Methode

- Setzen von Werten an verschiedenen Positionen

Ergebnis: success

2.2.4. Test der `get` Methode

- Abrufen von Werten an verschiedenen Positionen

Ergebnis: success

2.2.5. Test der `at` Methode

- Zugriff auf Elemente per Referenz
- Änderung von Elementen über die Referenz

Ergebnis: success

2.2.6. Test von ungültigen Positionen

- Versuch, auf Elemente außerhalb des gültigen Bereichs zuzugreifen
- Überprüfung der Fehlerbehandlung

Ergebnis: success

2.2.7. Test des Konstruktors mit Größe

- Erstellung eines Arrays mit einer bestimmten Anfangsgröße
- Setzen und Abrufen von Werten

Ergebnis: success

2.2.8. Test der `get_max_size` Methode

- Abrufen der maximalen Größe der Arrays

Ergebnis: success

2.2.9. Test der `clear` Methode

- Löschen aller Elemente im Array

Ergebnis: success

2.2.10. Test eines großen Arrays

- Erstellung eines Arrays mit mehr als `m_cols` Elementen
- Setzen und Abrufen von Werten

Ergebnis: success

Aufwand in h: 7