

L3: Verkettete Liste auf Arrays

Bekommen in L4

Abgabe in L5

Implementiere in C++ den gegebenen **Container (ADT)** mithilfe der gegebenen **Repräsentierung** und mit einer **verketteten Liste** (SLLA – Simple Linked List auf **Array**, DLLA – Double Linked List auf **Array**) als Datenstruktur, **repräsentiert auf einem Array**. Die verkettete Liste muss man selber implementieren. Für die Implementierung dürft ihr keine Containers oder Datenstrukturen aus STL (oder aus anderen Bibliotheken) benutzen.

Bemerkung. Für Repräsentierung, die nicht „standard“ sind, bitte siehe Vorlesung für Beispiele!

1. **ADT Matrix** – repräsentiert als schwachbesetzte Matrix (sparse), indem man ein SLLA von Tupeln der Form (Zeile, Spalte, Wert) (Wert \neq 0) benutzt, wobei die Tupel in lexikographischen Reihenfolge nach (Zeile, Spalte) gespeichert werden.
2. **ADT Matrix** – repräsentiert als schwachbesetzte Matrix (sparse), indem man ein DLLA von Tupeln der Form (Zeile, Spalte, Wert) (Wert \neq 0) benutzt, wobei die Tupel in lexikographischen Reihenfolge nach (Zeile, Spalte) gespeichert werden.
3. **ADT Bag** – repräsentiert mithilfe einer SLLA von Paaren der Form (Element, Frequenz).
4. **ADT Bag** – repräsentiert mithilfe einer DLLA von Paaren der Form (Element, Frequenz).
5. **ADT SortedBag** – mit Elementen vom Typ **TComp** repräsentiert mithilfe einer SLLA von Paaren der Form (Element, Frequenz), sortiert mithilfe einer Relation auf den Elementen
6. **ADT SortedBag** – mit Elementen vom Typ **TComp** repräsentiert mithilfe einer DLLA von Paaren der Form (Element, Frequenz), sortiert mithilfe einer Relation auf den Elementen
7. **ADT SortedSet** – mit Elementen vom Typ **TComp** repräsentiert mithilfe einer SLLA, sortiert mithilfe einer Relation auf den Elementen
8. **ADT SortedSet** – mit Elementen vom Typ **TComp** repräsentiert mithilfe einer DLLA, sortiert mithilfe einer Relation auf den Elementen
9. **ADT Set** – repräsentiert mithilfe einer SLLA von Elementen
10. **ADT Set** – repräsentiert mithilfe einer DLLA von Elementen
11. **ADT IndexedList** – repräsentiert mithilfe einer SLLA
12. **ADT IndexedList** – repräsentiert mithilfe einer DLLA
13. **ADT IteratedList** – repräsentiert mithilfe einer SLLA
14. **ADT IteratedList** – repräsentiert mithilfe einer DLLA
15. **ADT Map** – repräsentiert mithilfe einer SLLA von Paaren der Form (key, value)
16. **ADT Map** – repräsentiert mithilfe einer DLLA von Paaren der Form (key, value)
17. **ADT MultiMap** – repräsentiert mithilfe einer SLLA von Paaren der Form (key, value). Ein Schlüssel kann mehrmals vorkommen.

- 18. **ADT MultiMap** – repräsentiert mithilfe einer DLLA von Paaren der Form (*key*, *value*). Ein Schlüssel kann mehrmals vorkommen.
 - 19. **ADT MultiMap** – repräsentiert mithilfe einer SLLA mit eindeutigen Schlüsseln (*key*). Für jeden Schlüssel speichert man eine SLLA von Werten (*value*).
 - 20. **ADT MultiMap** – repräsentiert mithilfe einer DLLA mit eindeutigen Schlüsseln (*key*). Für jeden Schlüssel speichert man eine DLLA von Werten (*value*).
 - 21. **ADT SortedIndexedList** – repräsentiert mithilfe einer SLLA sortiert mithilfe einer Relation auf den Elementen
 - 22. **ADT SortedIndexedList** – repräsentiert mithilfe einer DLLA sortiert mithilfe einer Relation auf den Elementen
 - 23. **ADT SortedIteratedList** – repräsentiert mithilfe einer SLLA sortiert mithilfe einer Relation auf den Elementen
 - 24. **ADT SortedIteratedList** – repräsentiert mithilfe einer DLLA sortiert mithilfe einer Relation auf den Elementen
 - 25. **ADT PriorityQueue** – repräsentiert mithilfe einer SLLA von Paaren (Element, Priorität) sortiert nach den Prioritäten mithilfe einer Relation
 - 26. **ADT PriorityQueue** – repräsentiert mithilfe einer DLLA von Paaren (Element, Priorität) sortiert nach den Prioritäten mithilfe einer Relation
 - 27. **ADT SortedMap** – repräsentiert mithilfe einer SLLA von Paaren der Form (*key*, *value*) und sortiert mithilfe einer Relation auf den Schlüsseln (*key*)
 - 28. **ADT SortedMap** – repräsentiert mithilfe einer DLLA von Paaren der Form (*key*, *value*) und sortiert mithilfe einer Relation auf den Schlüsseln (*key*)
 - 29. **ADT SortedMultiMap** – repräsentiert mithilfe einer SLLA mit eindeutigen Schlüsseln (*key*) und sortiert mithilfe einer Relation auf den Schlüsseln (*key*). Für jeden Schlüssel speichert man eine SLLA von Werten (*value*).
 - 30. **ADT SortedMultiMap** – repräsentiert mithilfe einer DLLA mit eindeutigen Schlüsseln (*key*) und sortiert mithilfe einer Relation auf den Schlüsseln (*key*). Für jeden Schlüssel speichert man eine DLLA von Werten (*value*).
 - 31. **ADT SortedMultiMap** – repräsentiert mithilfe einer SLLA von Paaren der Form (*key*, *value*) und sortiert mithilfe einer Relation auf den Schlüsseln (*key*). Ein Schlüssel kann mehrmals vorkommen.
 - 32. **ADT SortedMultiMap** – repräsentiert mithilfe einer DLLA von Paaren der Form (*key*, *value*) und sortiert mithilfe einer Relation auf den Schlüsseln (*key*). Ein Schlüssel kann mehrmals vorkommen.
-

- 33. **ADT Matrix** – repräsentiert als schwachbesetzte Matrix (*sparse*) mit zirkulären Listen verbunden miteinander.

34. Roboter in dem Labyrinth:

In einem rechteckigen Labyrinth gibt es besetzte (X) und leere Zellen (*). Der Roboter kann sich in vier Richtungen bewegen: Nord, Süd, Ost, West.

X	*	*	X	X	X	*	*
X	*	X	*	*	*	*	*
X	*	*	*	*	*	X	*
X	X	X	*	*	*	X	*
*	X	*	*	R	X	X	*
*	*	*	X	X	X	X	*
*	*	*	*	*	*	*	X
X	X	X	X	X	X	X	X

- Bestimme ob es einen Pfad gibt, sodass der Roboter aus dem Labyrinth rauskommt (d.h. zu der ersten oder letzten Spalte oder zu der ersten oder letzten Zeile gelangen).
- Bestimme einen Pfad, falls es einen gibt.
- Bestimme einen Pfad mit minimaler Länge, falls es einen gibt.

Ihr müsst in der Implementierung **ADT Queue** repräsentiert als **SLLA** benutzen.

35. Roboter in dem Labyrinth:

In einem rechteckigen Labyrinth gibt es besetzte (X) und leere Zellen (*). Der Roboter kann sich in vier Richtungen bewegen: Nord, Süd, Ost, West.

X	*	*	X	X	X	*	*
X	*	X	*	*	*	*	*
X	*	*	*	*	*	X	*
X	X	X	*	*	*	X	*
*	X	*	*	R	X	X	*
*	*	*	X	X	X	X	*
*	*	*	*	*	*	*	X
X	X	X	X	X	X	X	X

- Bestimme ob es einen Pfad gibt, sodass der Roboter aus dem Labyrinth rauskommt (d.h. zu der ersten oder letzten Spalte oder zu der ersten oder letzten Zeile gelangen).
- Bestimme einen Pfad, falls es einen gibt.
- Bestimme einen Pfad mit minimaler Länge, falls es einen gibt.

Ihr müsst in der Implementierung **ADT Queue** repräsentiert als **DLLA** benutzen.

36. Rot-Schwarz Kartenspiel:

Zwei Spieler bekommen je $n/2$ Karten (rot und schwarz). Die zwei Spieler legen der Reihe nach je eine Karte (die oberste Karte aus ihrem Stapel) auf den Ablagestapel. Falls

ein Spieler eine rote Karte gelegt hat, dann nimmt der andere Spieler alle Karten von dem Ablagestapel und legt diese am Ende seines Stapels. Das Spiel ist beendet, wenn einer der Spieler alle n Karten in der Hand hat.

Simuliere das Spiel für n Karten.

Ihr müsst in der Implementierung **ADT Stack** repräsentiert mithilfe einer **SLLA** und **ADT Queue** repräsentiert mithilfe einer **DLLA** benutzen.

37. Rot-Schwarz Kartenspiel:

Zwei Spieler bekommen je $n/2$ Karten (rot und schwarz). Die zwei Spieler legen der Reihe nach je eine Karte (die oberste Karte aus ihrem Stapel) auf den Ablagestapel. Falls ein Spieler eine rote Karte gelegt hat, dann nimmt der andere Spieler alle Karten von dem Ablagestapel und legt diese am Ende seines Stapels. Das Spiel ist beendet, wenn einer der Spieler alle n Karten in der Hand hat.

Simuliere das Spiel für n Karten.

Ihr müsst in der Implementierung **ADT Stack** repräsentiert mithilfe einer **DLLA** und **ADT Queue** repräsentiert mithilfe einer **SLLA** benutzen.

38. Auswertung eines arithmetischen Ausdrucks in Infixnotation (der Ausdruck enthält Klammern). Man muss die Infixnotation in Postfixnotation umwandeln und dann die Postfixnotation auswerten.

Ihr müsst in der Implementierung **ADT Queue** repräsentiert mithilfe einer **SLLA** und **ADT Stack** repräsentiert mithilfe einer **DLLA** benutzen.

39. Auswertung eines arithmetischen Ausdrucks in Infixnotation (der Ausdruck enthält Klammern). Man muss die Infixnotation in Postfixnotation umwandeln und dann die Postfixnotation auswerten.

Ihr müsst in der Implementierung **ADT Queue** repräsentiert mithilfe einer **DLLA** und **ADT Stack** repräsentiert mithilfe einer **SLLA** benutzen.