Mohamad Raafat baki

4.1. Queue :

4.1.1. Array Queue Auf einer anfangs leeren Array-Queue Q[1..4] mit head und tail Indices (also ohne „Aufrutschen“) werden die nachfolgenden Operationen ausgeführt. Tragen Sie in die Tabelle alle Änderungen nach jedem Funktionsaufruf ein. Falls ein Fehler auftritt, tragen Sie eine passende Fehlermeldung ein. Dokumentieren Sie in der Skizze den kompletten Zustand der Datenstruktur nach dem letzten Aufruf!

|  |  |  |  |
| --- | --- | --- | --- |
| Änderung nach … | head | Tail | Wertzuweisung bzw. Fehler |
| Initialer Zustand | 1 | 1 |  |
| enqueue(A) | 1 | 2 | Q[ 1 ] = A |
| enqueue(B) | 1 | 3 | Q[ 2 ] = B |
| dequeue() | 2 | 3 | return Q[ 1 ] |
| enqueue(C) | 2 | 4 | Q[ 3 ] = C |
| enqueue(D) | 2 | 5 | Q[ 4 ] = D |
| dequeue() | 3 | 5 | return Q[ 2 ] |
| enqueue(E) | 3 | 5 | Fehler,Array ist full |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
|  |  | C | D |

4.1.2. Ringpuffer Auf einem anfangs leeren Ringpuffer R[1..4] mit head und tail Indices und size Variable werden die nachfolgenden Operationen ausgeführt. Tragen Sie in die Tabelle alle Werte nach jedem Funktionsaufruf ein. Falls ein Fehler auftritt, tragen Sie eine passende Fehlermeldung ein. Dokumentieren Sie in der Skizze den kompletten Zustand der Datenstruktur nach dem letzten Aufruf!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Änderung nach … | head | tail | size | Wertzuweisung bzw. Fehler |
| Initialer Zustand | 1 | 1 | 0 |  |
| enqueue(A) | 1 | 2 | 1 | R[ 1 ] = A |
| enqueue(B) | 1 | 3 | 2 | R[ 2 ] = B |
| dequeue() | 2 | 3 | 1 | return R[ 1 ] |
| enqueue(C) | 2 | 4 | 2 | R[ 3 ] = C |
| enqueue(D) | 2 | 1 | 3 | R[ 4 ] = D |
| dequeue() | 3 | 1 | 2 | return R[ 2 ] |
| enqueue(E) | 3 | 2 | 3 | R[ 1 ] = E |
| enqueue(F) | 3 | 3 | 4 | R[ 2 ] = F |
| enqueue(G) | 3 | 4 | 5 | Error, Full |
| dequeue() |  |  |  |  |
| enqueue(H) |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| E | F | C | D |

4.2. Stack :

4.2.1 Array-Stack Auf einem anfangs leeren Array-Stack S[1..4] mit top Index werden die nachfolgenden Operationen ausgeführt. Tragen Sie in die Tabelle alle Werte nach jedem Funktionsaufruf ein und dokumentieren Sie in der Skizze den kompletten Zustand der Datenstruktur nach dem letzten Aufruf!

|  |  |  |
| --- | --- | --- |
| Änderung nach … | top | Wertzuweisung bzw. Fehler |
| (Startzustand) | 1 |  |
| push(A) | 2 | S[ 1 ] = A |
| push(B) | 3 | S[ 2 ] = B |
| push(C) | 4 | S[ 3 ] = C |
| pop() | 3 | return S[ 3 ] |
| push(D) | 4 | S[ 3 ] = D |
| push(E) | 5 | S[ 4 ] = E |
| pop() | 4 | return S[ 4 ] |
| pop() | 3 | return S[ 3 ] |
| push(F) | 4 | S[ 3 ] = F |
| push(G) | 5 | S[ 4 ] = G |
| pop() | 4 | return S[ 4 ] |

|  |  |
| --- | --- |
| 1 | A |
| 2 | B |
| 3 | F |
| 4 |  |

4.3. Double Ended Queue (DEQ) :

4.3.1. Implementierung einer Double Ended Queue (DEQ) als Ringpuffer in Pseudocode:

a) Implementieren Sie die zur fehlerlosen Benutzung der DEQ notwendigen Attribute mit denen vorab getestet werden kann, ob eine Operation zulässig ist.

b) Implementieren Sie in Pseudocode eine Double Ended Queue (DEQ) als Verallgemeinerung aus Queue und Stack auf Basis eines Ringpuffers R[1..n] mit zwei Indices head und tail sowie einer Füllstandsvariable size. Eine DEQ unterstützt die folgenden Operationen: push\_head(x), pop\_head(), push\_tail(x), pop\_tail() .

var N = 4

var A[N] = '?'

var tail = 0

var head = 0

var tailr = N + 1

var headr = N + 1

var size = 0

func empty()

if size == 0

return true

else

return false

func full()

if size == N

return true

else

return false

func puch\_links(x)

if full()

throw " error : full"

tail = tail + 1

if tail == tailr

tail = 1

A[tail] = x

size = size + 1

func pop\_links()

if empty()

throw " error : empty "

head = head + 1

if head == tailr

head = 1

var x = A[head]

size = size - 1

return x

func puch\_recht(x)

if full()

throw " error : full "

tailr = tailr - 1

if tailr == tail

tailr = N

A[tailr] = x

size = size + 1

func pop\_recht()

if empty()

throw " error : empty "

headr = headr - 1

if headr == tail

headr = N

var x = A[headr]

size = size - 1

return x

puch\_links('A')

puch\_links('B')

puch\_recht('C')

puch\_recht('D')

print pop\_links()

print pop\_links()

print pop\_recht()

print pop\_recht()

4.4. Pseudocode :

4.4.1. Ein Array von Buchstaben mit einem Stack umdrehen :

a) Notieren Sie die Operationen und Attribute für einen Stack in Pseudocode.

b) Schreiben Sie einen Algorithmus in Pseudocode, der die Eingabe umdreht. Beispiel: Eingabe „LEBEN16“ Ausgabe: „61NEBEL“.

Benutzen Sie einen Stack aber keine eigenen Datenstrukturen wie z.B. Arrays.

Benutzen Sie die folgenden Funktionen zur zeichenweisen Ein- und Ausgabe:

getchar(): liest das nächste Zeichen (Buchstaben) von der Eingabe, Benutzung: x = getchar()

end(): gibt eine 1 zurück, falls die Eingabe keine Zeichen mehr hat und 0 sonst.

putchar(x): schreibt ein Zeichen in die Ausgabe, Benutzung: putchar(x).

a) • push( x ) Element x in den Stack einfügen.

• pop() neuestes Element zurückgeben und aus dem Stack löschen.

• empty() gibt 1 zurück, falls der Stack leer ist und 0 sonst.

b)

var N = 100

var S[N] = '?'

var top = 0

func end()

if top == 0

return true

else

return false

func full()

if top == N

return true

else

return false

func putchar(x)

if full()

throw " Stack full "

top = top + 1

S[top] = x

func getchar()

if end()

throw " stack empty "

var x = S[top]

top = top - 1

return x

putchar('L')

putchar('E')

putchar('B')

putchar('E')

putchar('N')

putchar('1')

putchar('6')

while ! end()

print getchar()

4.4.2. Großbuchstaben in einem Array mit einer Queue herausfiltern :

a) Notieren Sie die Operationen und Attribute für eine Queue in Pseudocode.

b) Schreiben Sie einen Algorithmus in Pseudocode, der die Eingabe liest und dabei alle Zeichen sofort wieder ausgibt, bis auf die Großbuchstaben. Alle Großbuchstaben sollen am Ende ausgegeben werden und zwar genau in derselben Reihenfolge in der sie in der Eingabe standen. Beispiel: Eingabe „LebeN16“ Ausgabe: „ebe16LN“.

Benutzen Sie eine Queue aber keine eigenen Datenstrukturen wie z.B. Arrays.

Benutzen Sie die folgenden Funktionen zur zeichenweisen Ein- und Ausgabe:

getchar(): liest das nächste Zeichen (Buchstaben) von der Eingabe, Benutzung: x = getchar() .

end(): gibt eine 1 zurück, falls die Eingabe keine Zeichen mehr hat und 0 sonst.

putchar(x): schreibt ein Zeichen in die Ausgabe, Benutzung: putchar(x) .

cap(x): gibt eine 1 zurück, falls x ein Großbuchstabe (capital letter) ist und 0 sonst.

a) • enqueue( x ) Element x hinten anfügen .

• dequeue() erstes Element zurückgeben und aus der Queue löschen .

• empty() gibt true (wahr) zurück, falls die Queue leer ist und false (falsch) sonst.

b)

func cup()

while ! end

var x = getchar()

if cap(x) == 1

enqueue(x)

else

putchar(x)

while ! end

var x = dequeue()

putchar(x)