



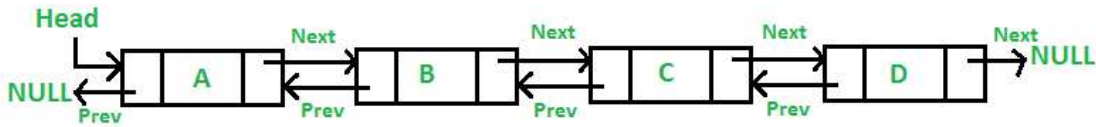


# Practicum woensdag: een linked list

 Collapse context



Deze week ga je met recursieve structs aan de slag. Hiervoor gaan we een dubbelgelinkte ketting/lijs implementeren. Een voorbeeld van zo een ketting zie je in onderstaande afbeelding.



Your answer passed the tests! Your score is 100.0%. [Submission #674f0c1bb6ace7c865e83f8b (2024-12-03 13:48:11)]

## Question 1: Structs

✓ Perfect

Definieer een struct `Node`, die bestaat uit drie onderdelen: de string `item`, een pointer naar de volgende node `next` en een pointer naar de vorige node `prev`.

Definieer vervolgens een struct `LinkedList`. Deze struct bestaat uit twee onderdelen, namelijk een integer `size` (die het aantal elementen in de lijst bijhoudt) en een pointer naar een `Node` `head` (het eerste element van de lijst).

```
1 struct Node
2 {
3     string item;
4     Node *next;
5     Node *prev;
6 };
7
8 struct LinkedList
9 {
10    int size;
11    Node *head;
12 };
```

Submit

## Question 2: Maak een lege lijst

✓ Perfect

Maak een functie `createList` die een pointer naar een lege `LinkedList` zal teruggeven. De `size` van een lege lijst is natuurlijk 0, en de `head` van deze lijst is een nullpointer.

```
1 LinkedList *createList()
2 {
3     LinkedList *list = new LinkedList;
4     list->head = nullptr;
5     list->size = 0;
6 }
```

```
7     return list;
8 }
```

Submit

### Question 3: Lijst is leeg

✓ Perfect

✕

Implementeer een functie `isEmpty(*list)`, die voor een lijst terug zal geven of de lijst leeg is.

```
1 bool isEmpty(LinkedList *list)
2 {
3     return list->size == 0;
4 }
```

Submit

### Question 4: Lengte van de lijst

✓ Perfect

✕

Implementeer vervolgens een functie `getLength(*list)`, die de size van een meegegeven lijst zal teruggeven.

```
1 int getLength(LinkedList *list)
2 {
3     return list->size;
4 }
```

Submit

### Question 5: Voeg toe

✓ Perfect

✕

Schrijf een functie `add(*list, newItem)` die een nieuw item aan het einde van de lijst toe zal voegen.

```
1 void add(LinkedList *list, string item)
2 {
3     Node *const head = list->head;
4
5     Node *newNode = new Node;
6     Node *currentNode = head;
7
8     newNode->item = item;
9 }
```

```

10     if (list->size == 0)
11     {
12         newNode->prev = nullptr;
13         newNode->next = nullptr;
14         list->head = newNode;
15         list->size++;
16         return;
17     }
18
19     while (currentNode->next != nullptr)
20     {
21         currentNode = currentNode->next;
22     }
23
24     newNode->prev = currentNode;
25     newNode->next = nullptr;
26
27     currentNode->next = newNode;
28     list->size++;
29 }

```

Submit

#### Question 6: Geef item

✓ Perfect



Schrijf nu een functie `get(*list, index)` die het item in de lijst dat op de meegegeven index staat teruggeeft (de index van de head van een lijst is 0). Als de index out of range is, geeft de functie `ERROR` terug.

```

1  string get(LinkedList *list, int index)
2  {
3      if (index >= list->size)
4      {
5          return "ERROR";
6      }
7
8      Node *currentNode = list->head;
9      int i = 0;
10     while (i != index)
11     {
12         currentNode = currentNode->next;
13         i++;
14     }
15
16     return currentNode->item;
17 }

```

Submit

#### Question 7: Print lijst

✓ Perfect



Schrijf nu een functie `print(*list)`, die elk element uit een lijst achter elkaar zal printen. De elementen worden gescheiden door newlines.

```

1  void print(LinkedList *list)

```

```

2 {
3     Node *const head = list->head;
4     Node *current = head;
5
6     while (current != nullptr)
7     {
8         cout << current->item << endl;
9         current = current->next;
10    }
11 }

```

Submit

#### Question 8: Voeg toe bij index

✓ Perfect



Implementeer de functie `set(*list, index, newItem)`. Deze functie zal het nieuwe item op de meegegeven index in de lijst invoegen (het werkt dus als een insert). Als deze operatie succesvol is, geeft de functie `true` terug. Als de index out of range van de lijst is, zal er niks aan de lijst toegevoegd worden en geeft de functie `false` terug.

```

1 bool set(LinkedList *list, int index, string newItem)
2 {
3     if (list->size < index)
4     {
5         return false;
6     }
7
8     Node *newNode = new Node;
9     newNode->item = newItem;
10
11     if (index == 0)
12     {
13         newNode->prev = nullptr;
14         newNode->next = list->head;
15         list->head->next->prev = newNode;
16         list->head = newNode;
17
18         list->size++;
19         return true;
20     }
21
22     Node *currentNode = list->head;
23     string item = get(list, index);
24
25     while (currentNode->item != item)
26     {
27         currentNode = currentNode->next;
28     }
29
30     currentNode->prev->next = newNode;
31     newNode->prev = currentNode->prev;
32     currentNode->prev = currentNode;
33     newNode->next = currentNode;
34
35     list->size++;
36
37     return true;
38 }

```

Submit

### Question 9: Verwijder

✓ Perfect

✕

Ga verder met de functie `remove(*list, index)`. Deze functie zal de Node bij de meegegeven index uit de lijst verwijderen. Opnieuw zal de functie `true` teruggeven als dit gelukt is, zo niet `false`.

```
1 bool remove(LinkedList* list, int index)
2 {
3     if (index >= list->size) {
4         return false;
5     }
6
7     if (index == 0) {
8         list->head = list->head->next;
9         delete list->head->prev;
10        list->head->prev = nullptr;
11        list->size--;
12        return true;
13    }
14
15    Node *currentNode = list->head;
16    string item = get(list, index);
17
18    while (currentNode->item != item)
19    {
20        currentNode = currentNode->next;
21    }
22
23    if (index == list->size - 1)
24    {
25        currentNode->prev->next = nullptr;
26    }
27    else
28    {
29        currentNode->prev->next = currentNode->next;
30        currentNode->next->prev = currentNode->prev;
31    }
32    delete currentNode;
33    list->size--;
34
35    return true;
36 }
```

Submit

### Question 10: Testen

✓ Perfect

✕

Je gaat nu de functie `destroyList(*list)` schrijven, die alle Nodes van een lijst en de lijst zelf gaat verwijderen.

Vul vervolgens onderstaande main functie aan om jouw code te testen.

1. Gebruik eerst de functie `createList` om een lege lijst `l` te maken.
2. Voeg nu een voor een de strings `Hello`, `World`, `!!` aan `l` toe. Gebruik hiervoor de functie `add`.
3. Gebruik `print` om je lijst te printen.

4. Voeg vervolgens met de functie `set` de string `dear` op index 1 van de lijst toe.
5. Print de lijst opnieuw.
6. Verwijder nu met `remove` het item op index 3 uit je lijst.
7. Print de lijst vervolgens af.

Zorg voordat de main functie eindigt ervoor dat er geen memory leaks in je programma optreden.

```
1 void destroyList(LinkedList *list)
2 {
3     if (list->size == 0)
4     {
5         return;
6     }
7
8     if (list->size == 1)
9     {
10        delete list->head;
11        delete list;
12        return;
13    }
14
15
16    while (list->head != nullptr)
17    {
18        Node* temp = list->head;
19        list->head = list->head->next;
20        delete temp;
21    }
22    delete list;
23 }
24
25 int main(int argc, char const *argv[])
26 {
27     LinkedList *list = createList();
28     add(list, "Hello");
29     add(list, "World");
30     add(list, "!!");
31     print(list);
32
33     set(list, 1, "dear");
34     print(list);
35
36     remove(list, 3);
37     print(list);
38
39     destroyList(list);
40 }
```

Submit

## Question 11: Queue

✓ Perfect



Je gaat nu een ADT "Queue" definiëren, hergebruik daarvoor de struct "Node". Het wordt dus een dubbelgelinkte queue. Je mag de vorige code niet hergebruiken.

1. Definieer een struct `Queue`, dat bestaat uit 3 elementen: `size`, `head` en `tail`. `head` en `tail` zijn allebei pointers naar Nodes die initeel `NULL` zijn.
2. Definieer de functie `createQueue` die een pointer naar een lege queue teruggeeft.
3. Definieer de functie `isEmpty(*queue)`, die `true` teruggeeft als de queue leeg is.

4. Definieer de functie `enqueue(*queue, newItem)` die newItem aan het eind van de queue zal invoegen. Als deze operatie geslaagd is, retournt de functie true.
5. Definieer de functie `dequeue(*queue)`. Deze functie retournt het head item en verwijdert deze uit de queue. Als de queue leeg is retournt deze functie "ERROR".
6. Definieer de functie `print(*queue)` om een queue uit te printen. Elk item van de queue wordt hier op een nieuwe lijn geprint vertrekkend van de head.
7. Definieer de functie `destroyQueue(*queue)` die de queue en al diens nodes verwijdert.

Je kunt zelf jouw code op dit programma testen.

```
int main(){
    Queue* q = createQueue();
    cout << boolalpha << isEmpty(q) << endl;
    cout << dequeue(q) << endl;
    enqueue(q, "Jos");
    enqueue(q, "An");
    enqueue(q, "Peter");
    print(q);           //Jos, An en Peter worden op drie regels geprint
    string first = dequeue(q);
    cout << first << endl; //Jos wordt geprint
    print(q);           //An en Peter worden geprint
    destroyQueue(q);
    return 0;
}
```

```
1  struct Node
2  {
3      string item;
4      Node *next;
5      Node *prev;
6  };
7
8  struct Queue
9  {
10     int size;
11     Node *head = nullptr;
12     Node *tail = nullptr;
13 };
14
15 Queue *createQueue()
16 {
17     Queue *newQ = new Queue;
18     newQ->size = 0;
19
20     return newQ;
21 }
22
23 bool isEmpty(Queue *q)
24 {
25     return q->size == 0;
26 }
27
28 bool enqueue(Queue *q, string item)
29 {
30
31     Node *const head = q->head;
32     Node *newNode = new Node;
33
34     newNode->item = item;
35
36     if (q->size == 0)
37     {
38         newNode->prev = nullptr;
39         newNode->next = nullptr;
40         q->head = newNode;
41         q->tail = newNode;
42         q->size++;
43         return true;
44     }
45 }
```

```

44     }
45
46     if (q->size == 1)
47     {
48         newNode->prev = q->head;
49         newNode->next = nullptr;
50         q->head->next = newNode;
51         q->tail = newNode;
52         q->size++;
53         return true;
54     }
55
56     newNode->prev = q->tail;
57     q->tail->next = newNode;
58     newNode->next = nullptr;
59     q->tail = newNode;
60
61     q->size++;
62     return true;
63 }
64
65 void print(Queue *q)
66 {
67     Node *current = q->head;
68
69     while (current != nullptr)
70     {
71         cout << current->item << endl;
72         current = current->next;
73     }
74 }
75
76 string dequeue(Queue *q)
77 {
78     if (q->size == 0)
79     {
80         return "ERROR";
81     }
82
83     string removedItem = q->head->item;
84     if (q->size == 1)
85     {
86         delete q->head;
87         q->size--;
88         return removedItem;
89     }
90
91     q->head = q->head->next;
92     delete q->head->prev;
93     q->head->prev = nullptr;
94
95     q->size--;
96     return removedItem;
97 }
98
99 void destroyQueue(Queue *q)
100 {
101     while (q->size != 0)
102     {
103         dequeue(q);
104     }
105     delete q;
106 }

```



Submit