

JUTGE PRO2 FIB

L11. Classe Queue

GitHub: <https://github.com/MUX-enjoyer/PRO2-FIB-2025>

Índex de Fitxers

X17005 Mètode per a moure el primer element d'una cua cap a la última posició.cc
(pàgina 2)

X80705 Nou mètode de la classe Queue per a accedir indexadament als seus elements.cc (pàgina 5)

X86445 Mètode de Queue per a multiplicar els elements de la cua per un paràmetre.cc (pàgina 8)

Exercici: X17005 Mètode per a moure el primer element d'una cua cap a la última posició.cc

```
1 #include <iostream>
2
3 using namespace std;
4
5 template <typename T>
6 class Queue {
7
8 private:
9     struct Item {
10         T value;
11         Item *next;
12     };
13
14     Item *first;
15     Item *last;
16     int _size;
17
18     void copyItems(const Item *item, Item *(&first), Item *(&last), int &_size)
19     {
20         if (item == NULL) {
21             first = NULL;
22             last = NULL;
23             _size = 0;
24             return;
25         }
26         first = new Item();
27         first->value = item->value;
28         last = first;
29         _size = 1;
30         while (item->next != NULL) {
31             last->next = new Item();
32             last = last->next;
33             item = item->next;
34             last->value = item->value;
35             _size++;
36         }
37         last->next = NULL;
38     }
39
40     void deleteItems(Item *item) {
41         while (item != NULL) {
42             Item *aux = item;
43             item = item->next;
44             delete aux;
45         }
46     }
47
48 public:
49     Queue() {
50         first = last = NULL;
51         _size = 0;
52     }
53
54 }
55
```

```

56 Queue(Queue &q)
57 {
58     copyItems(q.first, first, last, _size);
59 }
60
61 ~Queue() {
62     deleteItems(first);
63     _size = 0;
64 }
65
66 Queue &operator=(const Queue &q) {
67     if (this != &q) {
68         deleteItems(first);
69         copyItems(q.first, first, last, _size);
70     }
71     return *this;
72 }
73
74 T front() {
75     if (first == NULL) {
76         cerr << "Error: front on empty queue" << endl;
77         exit(0);
78     }
79     return first->value;
80 }
81
82 void pop() {
83     if (first == NULL) {
84         cerr << "Error: pop on empty queue" << endl;
85         exit(0);
86     }
87     Item *aux = first;
88     first = first->next;
89     delete aux;
90     _size--;
91     if (first == NULL) last = NULL;
92 }
93
94 void push(T value) {
95     Item *pitem = new Item();
96     pitem->value = value;
97     pitem->next = NULL;
98     if (first == NULL) {
99         first = last = pitem;
100     _size = 1;
101     return;
102 }
103 last->next = pitem;
104 last = pitem;
105 _size++;
106 }
107
108 int size() {
109     return _size;
110 }
111
112 template<typename U> friend ostream &operator<<(ostream &os, Queue<U> &q);
113
114 template<typename U> friend istream &operator>>(istream &is, Queue<U> &q);
115
116 // Pre: // Post: L'element que estava en primera posició de la cua implícita
// ha estat mogut cap a la última posició. // En el cas en que la cua no tingués
cap element, res ha canviat. // Descomenteu les següents dues línies i

```

```
implementeu la funció: void moveFrontToLast() {
117 if (_size > 1) {
118 Item *primer = first;
119 first = first->next;
120 last->next = primer;
121 primer->next = NULL;
122 last = primer;
123 }
124 }
125 };
126
127
128 template<typename U>
129 ostream &operator<<(ostream &os, Queue<U> &q)
130 {
131 os << q._size;
132 for (typename Queue<U>::Item *item = q.first; item != NULL; item =
item->next)
133 os << " " << item->value;
134 return os;
135 }
136
137 template<typename U>
138 istream &operator>>(istream &is, Queue<U> &q)
139 {
140 int size;
141 is >> size;
142 if (size == 0) {
143 q = Queue<U> ();
144 return is;
145 }
146 for (int i = 0; i < size; ++i) {
147 U x;
148 cin >> x;
149 q.push(x);
150 }
151 return is;
152 }
```

Exercici: X80705 Nou mètode de la classe Queue per a accedir indexadament als seus elements.cc

```
1 #include <iostream>
2
3 using namespace std;
4
5 template <typename T>
6 class Queue {
7
8 private:
9 struct Item {
10 T value;
11 Item *next;
12 };
13
14 Item *first;
15 Item *last;
16 int _size;
17
18 void copyItems(const Item *item, Item *(&first), Item *(&last), int &_size)
19 {
20 if (item == NULL) {
21 first = NULL;
22 last = NULL;
23 _size = 0;
24 return;
25 }
26 first = new Item();
27 first->value = item->value;
28 last = first;
29 _size = 1;
30 while (item->next != NULL) {
31 last->next = new Item();
32 last = last->next;
33 item = item->next;
34 last->value = item->value;
35 _size++;
36 }
37 last->next = NULL;
38 }
39
40
41 void deleteItems(Item *item) {
42 while (item != NULL) {
43 Item *aux = item;
44 item = item->next;
45 delete aux;
46 }
47 }
48
49 public:
50
51 Queue() {
52 first = last = NULL;
53 _size = 0;
54 }
55 }
```

```

56 Queue(Queue &q)
57 {
58     copyItems(q.first, first, last, _size);
59 }
60
61 ~Queue() {
62     deleteItems(first);
63     _size = 0;
64 }
65
66 Queue &operator=(const Queue &q) {
67     if (this != &q) {
68         deleteItems(first);
69         copyItems(q.first, first, last, _size);
70     }
71     return *this;
72 }
73
74 T front() {
75     if (first == NULL) {
76         cerr << "Error: front on empty queue" << endl;
77         exit(0);
78     }
79     return first->value;
80 }
81
82 void pop() {
83     if (first == NULL) {
84         cerr << "Error: pop on empty queue" << endl;
85         exit(0);
86     }
87     Item *aux = first;
88     first = first->next;
89     delete aux;
90     _size--;
91     if (first == NULL) last = NULL;
92 }
93
94 void push(T value) {
95     Item *pitem = new Item();
96     pitem->value = value;
97     pitem->next = NULL;
98     if (first == NULL) {
99         first = last = pitem;
100     _size = 1;
101     return;
102 }
103 last->next = pitem;
104 last = pitem;
105 _size++;
106 }
107
108 int size() {
109     return _size;
110 }
111
112 template<typename U> friend ostream &operator<<(ostream &os, Queue<U> &q);
113
114 template<typename U> friend istream &operator>>(istream &is, Queue<U> &q);
115
116 // Pre: i està entre 0 i la mida de la cua implícita menys 1. // Post:
Retorna l'i-èssim valor de la cua implícita (indexat començant des de 0). //
Descomenteu les següents dues línies i implementeu el mètode: T operator[](int i)

```

```
const {
117 Item *aux = first;
118 for (int j = 0; j < i; ++j) {
119     aux = aux->next;
120 }
121 return aux->value;
122 }
123
124 };
125
126
127 template<typename U>
128 ostream &operator<<(ostream &os, Queue<U> &q)
129 {
130     os << q._size;
131     for (typename Queue<U>::Item *item = q.first; item != NULL; item =
item->next)
132         os << " " << item->value;
133     return os;
134 }
135
136 template<typename U>
137 istream &operator>>(istream &is, Queue<U> &q)
138 {
139     int size;
140     is >> size;
141     if (size == 0) {
142         q = Queue<U> ();
143         return is;
144     }
145     for (int i = 0; i < size; ++i) {
146         U x;
147         cin >> x;
148         q.push(x);
149     }
150     return is;
151 }
```

Exercici: X86445 Mètode de Queue per a multiplicar els elements de la cua per un paràmetre.cc

```
1 #include <iostream>
2
3 using namespace std;
4
5 template <typename T>
6 class Queue {
7
8 private:
9 struct Item {
10 T value;
11 Item *next;
12 };
13
14 Item *first;
15 Item *last;
16 int _size;
17
18 void copyItems(const Item *item, Item *(&first), Item *(&last), int &_size)
19 {
20 if (item == NULL) {
21 first = NULL;
22 last = NULL;
23 _size = 0;
24 return;
25 }
26 first = new Item();
27 first->value = item->value;
28 last = first;
29 _size = 1;
30 while (item->next != NULL) {
31 last->next = new Item();
32 last = last->next;
33 item = item->next;
34 last->value = item->value;
35 _size++;
36 }
37 last->next = NULL;
38 }
39
40
41 void deleteItems(Item *item) {
42 while (item != NULL) {
43 Item *aux = item;
44 item = item->next;
45 delete aux;
46 }
47 }
48
49 public:
50
51 Queue() {
52 first = last = NULL;
53 _size = 0;
54 }
55 }
```



```

56 Queue(Queue &q)
57 {
58     copyItems(q.first, first, last, _size);
59 }
60
61 ~Queue() {
62     deleteItems(first);
63     _size = 0;
64 }
65
66 Queue &operator=(const Queue &q) {
67     if (this != &q) {
68         deleteItems(first);
69         copyItems(q.first, first, last, _size);
70     }
71     return *this;
72 }
73
74 T front() {
75     if (first == NULL) {
76         cerr << "Error: front on empty queue" << endl;
77         exit(0);
78     }
79     return first->value;
80 }
81
82 void pop() {
83     if (first == NULL) {
84         cerr << "Error: pop on empty queue" << endl;
85         exit(0);
86     }
87     Item *aux = first;
88     first = first->next;
89     delete aux;
90     _size--;
91     if (first == NULL) last = NULL;
92 }
93
94 void push(T value) {
95     Item *pitem = new Item();
96     pitem->value = value;
97     pitem->next = NULL;
98     if (first == NULL) {
99         first = last = pitem;
100     _size = 1;
101     return;
102 }
103 last->next = pitem;
104 last = pitem;
105 _size++;
106 }
107
108 int size() {
109     return _size;
110 }
111
112 template<typename U> friend ostream &operator<<(ostream &os, Queue<U> &q);
113
114 template<typename U> friend istream &operator>>(istream &is, Queue<U> &q);
115
116 // Pre: // Post: Tots els elements de la cua implícita han estat multiplicats
// per 'value'. // Descomenteu les següents dues línies i implementeu el mètode:
void operator*=(T value) {

```

```
117 Item *pitem = first;
118 for (int i = 0; i < _size; ++i) {
119     pitem->value = pitem->value*value;
120     pitem = pitem->next;
121 }
122 }
123 };
124
125
126 template<typename U>
127 ostream &operator<<(ostream &os, Queue<U> &q)
128 {
129     os << q._size;
130     for (typename Queue<U>::Item *item = q.first; item != NULL; item =
item->next)
131         os << " " << item->value;
132     return os;
133 }
134
135 template<typename U>
136 istream &operator>>(istream &is, Queue<U> &q)
137 {
138     int size;
139     is >> size;
140     if (size == 0) {
141         q = Queue<U> ();
142         return is;
143     }
144     for (int i = 0; i < size; ++i) {
145         U x;
146         cin >> x;
147         q.push(x);
148     }
149     return is;
150 }
```