# JUTGE PRO2 FIB

## L10. Classe Stack

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

## Índex de Fitxers

# Exercici: X72693 Mètode de Stack per a esborrar el segon element des del top.cc

```
#include <iostream>
#include <cstdlib>

using namespace std;

template <typename T>
class Stack {
private:

// Items:
struct Item {
T value;
Item* next;
};

// Data:
int _size;
Item* ptopitem;

// Helpers:
Item* copyItems(Item *pitem) {
if (pitem == NULL) return NULL;
Item *pfirstitem = new Item();
Item *pcurrentitem = pfirstitem;
while (pitem != NULL) {
pcurrentitem->value = pitem->value;
if (pitem->next != NULL) pcurrentitem->next = new Item();
else pcurrentitem->next = NULL;
pcurrentitem = pcurrentitem->next;
pitem = pitem->next;
}
return pfirstitem;
}

void deleteItems()
{
Item *pitem = ptopitem;
while (pitem != NULL) {
Item *pitemaux = pitem;
pitem = pitem->next;
delete pitemaux;
}
_size = 0;
}

void printRec(ostream &os, const Item *pitem) const
{
if (pitem == NULL) return;
printRec(os, pitem->next);
os << " ";
os << pitem->value;
}

public:

```

```
56 // Constructors/destructors. Stack() {
57 ptopitem = NULL;
58 _size = 0;
59 }
60
61 Stack(const Stack<T> &s) {
62 ptopitem = copyItems(s.ptopitem);
63 _size = s._size;
64 }
65
66 ~Stack() {
67 deleteItems();
68 }
69
70 // Assignment:
71 Stack<T> &operator=(const Stack<T> &s) {
72 if (this != &s) {
73 deleteItems();
74 ptopitem = copyItems(s.ptopitem);
75 _size = s._size;
76 }
77 return *this;
78 }
79
80 // Standard functions:
81 int size() {
82 return _size;
83 }
84
85 bool empty() const {
86 return ptopitem == NULL;
87 }
88
89 void push(T value) {
90 Item *pnewitem = new Item();
91 pnewitem->value = value;
92 pnewitem->next = ptopitem;
93 ptopitem = pnewitem;
94 _size++;
95 }
96
97 void pop() {
98 if (ptopitem == NULL) {
99 cerr << "Error: pop on empty stack" << endl;
100 exit(1);
101 }
102 Item *paux = ptopitem;
103 ptopitem = ptopitem->next;
104 delete paux;
105 _size--;
106 }
107
108 T top() {
109 if (ptopitem == NULL) {
110 cerr << "Error: accessing top of empty stack" << endl;
111 exit(1);
112 }
113 return ptopitem->value;
114 }
115
116 const T top() const {
117 if (ptopitem == NULL) {
118 cerr << "Error: accessing top of empty stack" << endl;
```

```
119 exit(1);
120 }
121 return ptopitem->value;
122 }
123
124 // Read and write:
125 template <typename U> friend ostream &operator<<(ostream &os, const Stack<U>
&s);
126 template<typename U> friend istream &operator>>(istream &is, Stack<U> &s);
127
128 // Pre: // Post: Elimina l'element de la pila implícita que es troba en
segona posició des del top. // Si la pila té només un element, llavors elimina
aquest element. // Si la pila no té cap element, llavors no canvia res. //
Descomenteu les següents dues linies i implementeu el mètode: void pop2() {
129 if (size() == 1) pop();
130 else if (size() > 1) {
131 Item* aux = ptopitem->next;
132 ptopitem->next = aux->next;
133 delete aux;
134 --_size;
135 }
136 }
137
138 };
139
140 // Implementation of read and write:
141 template <typename U> ostream &operator<<(ostream &os, const Stack<U> &s)
142 {
143 os << s._size;
144 s.printRec(os, s.ptopitem);
145 return os;
146 }
147
148 template <typename U> istream &operator>>(istream &is, Stack<U> &s)
149 {
150 int n;
151 is >> n;
152 s = Stack<U> ();
153 for (int i = 0; i < n; i++) {
154 U u;
155 is >> u;
156 s.push(u);
157 }
158 return is;
159 }
```

# Exercici: X80037 Mètode per a sumar el contingut que ve d'una altra pila.cc

```
#include <iostream>
#include <cstdlib>

using namespace std;

template <typename T>
class Stack {
private:

// Items:
struct Item {
T value;
Item* next;
};

// Data:
int _size;
Item* ptopitem;

// Helpers:
Item* copyItems(Item *pitem) {
if (pitem == NULL) return NULL;
Item *pfirstitem = new Item();
Item *pcurrentitem = pfirstitem;
while (pitem != NULL) {
pcurrentitem->value = pitem->value;
if (pitem->next != NULL) pcurrentitem->next = new Item();
else pcurrentitem->next = NULL;
pcurrentitem = pcurrentitem->next;
pitem = pitem->next;
}
return pfirstitem;
}

void deleteItems()
{
Item *pitem = ptopitem;
while (pitem != NULL) {
Item *pitemaux = pitem;
pitem = pitem->next;
delete pitemaux;
}
_size = 0;
}

void printRec(ostream &os, const Item *pitem) const
{
if (pitem == NULL) return;
printRec(os, pitem->next);
os << " ";
os << pitem->value;
}

public:

```

```
56 // Constructors/destructors. Stack() {
57 ptopitem = NULL;
58 _size = 0;
59 }
60
61 Stack(const Stack<T> &s) {
62 ptopitem = copyItems(s.ptopitem);
63 _size = s._size;
64 }
65
66 ~Stack() {
67 deleteItems();
68 }
69
70 // Assignment:
71 Stack<T> &operator=(const Stack<T> &s) {
72 if (this != &s) {
73 deleteItems();
74 ptopitem = copyItems(s.ptopitem);
75 _size = s._size;
76 }
77 return *this;
78 }
79
80 // Standard functions:
81 int size() {
82 return _size;
83 }
84
85 bool empty() const {
86 return ptopitem == NULL;
87 }
88
89 void push(T value) {
90 Item *pnewitem = new Item();
91 pnewitem->value = value;
92 pnewitem->next = ptopitem;
93 ptopitem = pnewitem;
94 _size++;
95 }
96
97 void pop() {
98 if (ptopitem == NULL) {
99 cerr << "Error: pop on empty stack" << endl;
100 exit(1);
101 }
102 Item *paux = ptopitem;
103 ptopitem = ptopitem->next;
104 delete paux;
105 _size--;
106 }
107
108 T top() {
109 if (ptopitem == NULL) {
110 cerr << "Error: accessing top of empty stack" << endl;
111 exit(1);
112 }
113 return ptopitem->value;
114 }
115
116 const T top() const {
117 if (ptopitem == NULL) {
118 cerr << "Error: accessing top of empty stack" << endl;
```

```
119 exit(1);
120 }
121 return ptopitem->value;
122 }
123
124 // Read and write:
125 template <typename U> friend ostream &operator<<(ostream &os, const Stack<U>
&s);
126 template<typename U> friend istream &operator>>(istream &is, Stack<U> &s);
127
128 // Pre: // Post: A l'element del top de la pila implícita se li ha sumat //
l'element del top de s, // al segon element des del top de la pila implícita se
li ha sumat // el segon element des del top de s, i així successivament. // Si la
pila implícita era més llarga que s, // llavors aquells elements extres no han
canviat. // Si s era més llarga que la pila implícita, llavors aquells elements
// extres de s han estat ignorats. // Descomenteu les següents dues linies i
implementeu la funció: void operator+=(const Stack<T> &s) {
129 Item *pitem1 = ptopitem;
130 Item *pitem2 = s.ptopitem;
131 while (pitem1 != NULL && pitem2 != NULL) {
132 pitem1->value += pitem2->value;
133 pitem1 = pitem1->next;
134 pitem2 = pitem2->next;
135 }
136 }
137 };
138
139 // Implementation of read and write:
140 template <typename U> ostream &operator<<(ostream &os, const Stack<U> &s)
141 {
142 os << s._size;
143 s.printRec(os, s.ptopitem);
144 return os;
145 }
146
147 template <typename U> istream &operator>>(istream &is, Stack<U> &s)
148 {
149 int n;
150 is >> n;
151 s = Stack<U> ();
152 for (int i = 0; i < n; i++) {
153 U u;
154 is >> u;
155 s.push(u);
156 }
157 return is;
158 }
```

# Exercici: X82586 Implementar mètode de la classe Stack per a intercanviar els dos elements del top.cc

```
#include <iostream>
#include <cstdlib>

using namespace std;

template <typename T>
class Stack {
private:

// Items:
struct Item {
T value;
Item* next;
};

// Data:
int _size;
Item* ptopitem;

// Helpers:
Item* copyItems(Item *pitem) {
if (pitem == NULL) return NULL;
Item *pfirstitem = new Item();
Item *pcurrentitem = pfirstitem;
while (pitem != NULL) {
pcurrentitem->value = pitem->value;
if (pitem->next != NULL) pcurrentitem->next = new Item();
else pcurrentitem->next = NULL;
pcurrentitem = pcurrentitem->next;
pitem = pitem->next;
}
return pfirstitem;
}

void deleteItems()
{
Item *pitem = ptopitem;
while (pitem != NULL) {
Item *pitemaux = pitem;
pitem = pitem->next;
delete pitemaux;
}
_size = 0;
}

void printRec(ostream &os, const Item *pitem) const
{
if (pitem == NULL) return;
printRec(os, pitem->next);
os << " ";
os << pitem->value;
}

public:

```

```
56 // Constructors/destructors. Stack() {
57 ptopitem = NULL;
58 _size = 0;
59 }
60
61 Stack(const Stack<T> &s) {
62 ptopitem = copyItems(s.ptopitem);
63 _size = s._size;
64 }
65
66 ~Stack() {
67 deleteItems();
68 }
69
70 // Assignment:
71 Stack<T> &operator=(const Stack<T> &s) {
72 if (this != &s) {
73 deleteItems();
74 ptopitem = copyItems(s.ptopitem);
75 _size = s._size;
76 }
77 return *this;
78 }
79
80 // Standard functions:
81 int size() {
82 return _size;
83 }
84
85 bool empty() const {
86 return ptopitem == NULL;
87 }
88
89 void push(T value) {
90 Item *pnewitem = new Item();
91 pnewitem->value = value;
92 pnewitem->next = ptopitem;
93 ptopitem = pnewitem;
94 _size++;
95 }
96
97 void pop() {
98 if (ptopitem == NULL) {
99 cerr << "Error: pop on empty stack" << endl;
100 exit(1);
101 }
102 Item *paux = ptopitem;
103 ptopitem = ptopitem->next;
104 delete paux;
105 _size--;
106 }
107
108 T top() {
109 if (ptopitem == NULL) {
110 cerr << "Error: accessing top of empty stack" << endl;
111 exit(1);
112 }
113 return ptopitem->value;
114 }
115
116 const T top() const {
117 if (ptopitem == NULL) {
118 cerr << "Error: accessing top of empty stack" << endl;
```

```
119 exit(1);
120 }
121 return ptopitem->value;
122 }
123
124 // Read and write:
125 template <typename U> friend ostream &operator<<(ostream &os, const Stack<U>
&s);
126 template<typename U> friend istream &operator>>(istream &is, Stack<U> &s);
127
128 // Pre: El paràmetre implícit és una pila amb com a mínim dos elements. //
Post: Els dos valors del cim de la pila s'han intercanviat. // Descomenteu les
següents dues linies i implementeu la funció: void swap2Topmost() {
129 Item* first = ptopitem;
130 Item* second = ptopitem->next;
131
132 ptopitem = second;
133 first->next = second->next;
134 second->next = first;
135 }
136 };
137
138 // Implementation of read and write:
139 template <typename U> ostream &operator<<(ostream &os, const Stack<U> &s)
140 {
141 os << s._size;
142 s.printRec(os, s.ptopitem);
143 return os;
144 }
145
146 template <typename U> istream &operator>>(istream &is, Stack<U> &s)
147 {
148 int n;
149 is >> n;
150 s = Stack<U> ();
151 for (int i = 0; i < n; i++) {
152 U u;
153 is >> u;
154 s.push(u);
155 }
156 return is;
157 }
```

# Exercici: X87185 Mètode de Stack per a esborrar el primer element igual al paràmetre.cc

```cpp
#include <iostream>
#include <cstdlib>

using namespace std;

template <typename T>
class Stack {
private:

// Items:
struct Item {
T value;
Item* next;
};

// Data:
int _size;
Item* ptopitem;

// Helpers:
Item* copyItems(Item *pitem) {
if (pitem == NULL) return NULL;
Item *pfirstitem = new Item();
Item *pcurrentitem = pfirstitem;
while (pitem != NULL) {
pcurrentitem->value = pitem->value;
if (pitem->next != NULL) pcurrentitem->next = new Item();
else pcurrentitem->next = NULL;
pcurrentitem = pcurrentitem->next;
pitem = pitem->next;
}
return pfirstitem;
}

void deleteItems()
{
Item *pitem = ptopitem;
while (pitem != NULL) {
Item *pitemaux = pitem;
pitem = pitem->next;
delete pitemaux;
}
_size = 0;
}

void printRec(ostream &os, const Item *pitem) const
{
if (pitem == NULL) return;
printRec(os, pitem->next);
os << " ";
os << pitem->value;
}

public:

```

```
56 // Constructors/destructors. Stack() {
57 ptopitem = NULL;
58 _size = 0;
59 }
60
61 Stack(const Stack<T> &s) {
62 ptopitem = copyItems(s.ptopitem);
63 _size = s._size;
64 }
65
66 ~Stack() {
67 deleteItems();
68 }
69
70 // Assignment:
71 Stack<T> &operator=(const Stack<T> &s) {
72 if (this != &s) {
73 deleteItems();
74 ptopitem = copyItems(s.ptopitem);
75 _size = s._size;
76 }
77 return *this;
78 }
79
80 // Standard functions:
81 int size() {
82 return _size;
83 }
84
85 bool empty() const {
86 return ptopitem == NULL;
87 }
88
89 void push(T value) {
90 Item *pnewitem = new Item();
91 pnewitem->value = value;
92 pnewitem->next = ptopitem;
93 ptopitem = pnewitem;
94 _size++;
95 }
96
97 void pop() {
98 if (ptopitem == NULL) {
99 cerr << "Error: pop on empty stack" << endl;
100 exit(1);
101 }
102 Item *paux = ptopitem;
103 ptopitem = ptopitem->next;
104 delete paux;
105 _size--;
106 }
107
108 T top() {
109 if (ptopitem == NULL) {
110 cerr << "Error: accessing top of empty stack" << endl;
111 exit(1);
112 }
113 return ptopitem->value;
114 }
115
116 const T top() const {
117 if (ptopitem == NULL) {
118 cerr << "Error: accessing top of empty stack" << endl;
```

```
119 exit(1);
120 }
121 return ptopitem->value;
122 }
123
124 // Read and write:
125 template <typename U> friend ostream &operator<<(ostream &os, const Stack<U>
&s);
126 template<typename U> friend istream &operator>>(istream &is, Stack<U> &s);
127
128 // Pre: // Post: La primera ocurrència de 'value' en la pila implícita ha
estat esborrada. // Si no hi havia cap ocurrència de 'value', llavors no ha
canviat res. // Descomenteu les següents dues linies i implementeu la funció:
void removeFirstOccurrence(T value) {
129 if (_size > 0) {
130 if (ptopitem->value == value) {
131 Item *paux = ptopitem;
132 ptopitem = ptopitem->next;
133 delete paux;
134 --_size;
135 return;
136 }
137
138 Item* prev = ptopitem;
139 Item* current = ptopitem->next;
140
141 while (current != NULL) {
142 if (current->value == value) {
143 prev->next = current->next;
144 delete current;
145 --_size;
146 return;
147 }
148 prev = current;
149 current = current->next;
150 }
151 }
152 }
153
154 };
155
156 // Implementation of read and write:
157 template <typename U> ostream &operator<<(ostream &os, const Stack<U> &s)
158 {
159 os << s._size;
160 s.printRec(os, s.ptopitem);
161 return os;
162 }
163
164 template <typename U> istream &operator>>(istream &is, Stack<U> &s)
165 {
166 int n;
167 is >> n;
168 s = Stack<U> ();
169 for (int i = 0; i < n; i++) {
170 U u;
171 is >> u;
172 s.push(u);
173 }
174 return is;
175 }
```

# Exercici: X94161 Mètode de Stack per a accedir al segon element des del top.cc

```
#include <iostream>
#include <cstdlib>

using namespace std;

template <typename T>
class Stack {
private:

// Items:
struct Item {
T value;
Item* next;
};

// Data:
int _size;
Item* ptopitem;

// Helpers:
Item* copyItems(Item *pitem) {
if (pitem == NULL) return NULL;
Item *pfirstitem = new Item();
Item *pcurrentitem = pfirstitem;
while (pitem != NULL) {
pcurrentitem->value = pitem->value;
if (pitem->next != NULL) pcurrentitem->next = new Item();
else pcurrentitem->next = NULL;
pcurrentitem = pcurrentitem->next;
pitem = pitem->next;
}
return pfirstitem;
}

void deleteItems()
{
Item *pitem = ptopitem;
while (pitem != NULL) {
Item *pitemaux = pitem;
pitem = pitem->next;
delete pitemaux;
}
_size = 0;
}

void printRec(ostream &os, const Item *pitem) const
{
if (pitem == NULL) return;
printRec(os, pitem->next);
os << " ";
os << pitem->value;
}

public:

```

```
56 // Constructors/destructors. Stack() {
57 ptopitem = NULL;
58 _size = 0;
59 }
60
61 Stack(const Stack<T> &s) {
62 ptopitem = copyItems(s.ptopitem);
63 _size = s._size;
64 }
65
66 ~Stack() {
67 deleteItems();
68 }
69
70 // Assignment:
71 Stack<T> &operator=(const Stack<T> &s) {
72 if (this != &s) {
73 deleteItems();
74 ptopitem = copyItems(s.ptopitem);
75 _size = s._size;
76 }
77 return *this;
78 }
79
80 // Standard functions:
81 int size() {
82 return _size;
83 }
84
85 bool empty() const {
86 return ptopitem == NULL;
87 }
88
89 void push(T value) {
90 Item *pnewitem = new Item();
91 pnewitem->value = value;
92 pnewitem->next = ptopitem;
93 ptopitem = pnewitem;
94 _size++;
95 }
96
97 void pop() {
98 if (ptopitem == NULL) {
99 cerr << "Error: pop on empty stack" << endl;
100 exit(1);
101 }
102 Item *paux = ptopitem;
103 ptopitem = ptopitem->next;
104 delete paux;
105 _size--;
106 }
107
108 T top() {
109 if (ptopitem == NULL) {
110 cerr << "Error: accessing top of empty stack" << endl;
111 exit(1);
112 }
113 return ptopitem->value;
114 }
115
116 const T top() const {
117 if (ptopitem == NULL) {
118 cerr << "Error: accessing top of empty stack" << endl;
```

```
119 exit(1);
120 }
121 return ptopitem->value;
122 }
123
124 // Read and write:
125 template <typename U> friend ostream &operator<<(ostream &os, const Stack<U>
&s);
126 template<typename U> friend istream &operator>>(istream &is, Stack<U> &s);
127
128 // Pre: La pila implícita té un element o més. // Post: Retorna l'element de
la pila implícita que es troba en segona posició des del top. // Si la pila té
només un element, llavors retorna aquest element. // Descomenteu les següents
dues linies i implementeu el mètode: T top2() {
129 if (size() == 1) return top();
130 else {
131 Item* aux = ptopitem->next;
132 return aux->value;
133 }
134 }
135 };
136
137 // Implementation of read and write:
138 template <typename U> ostream &operator<<(ostream &os, const Stack<U> &s)
139 {
140 os << s._size;
141 s.printRec(os, s.ptopitem);
142 return os;
143 }
144
145 template <typename U> istream &operator>>(istream &is, Stack<U> &s)
146 {
147 int n;
148 is >> n;
149 s = Stack<U> ();
150 for (int i = 0; i < n; i++) {
151 U u;
152 is >> u;
153 s.push(u);
154 }
155 return is;
156 }
```