Programmazione 2
e Laboratorio di Programmazione

Corso di Laurea in

Informatica

Università degli Studi di Napoli "Parthenope"

Anno Accademico 2023-2024

Prof. Luigi Catuogno

1

Informazioni sul corso

Docente	Luigi Catuogno **luigi.catuogno@uniparthenope.it**
Orario	Lun: 9:00-11:00 Mer: 11:00-13:00
Sede	Centro Direzionale Napoli **Aula Magna**
Ricevimento	Mer: 14:00-16:00 (previo appuntamento) Ufficio docente oppure Team: **cxxa3bo**

2

Libri di testo

Introduzione al linguaggio – costrutti e tecniche di base

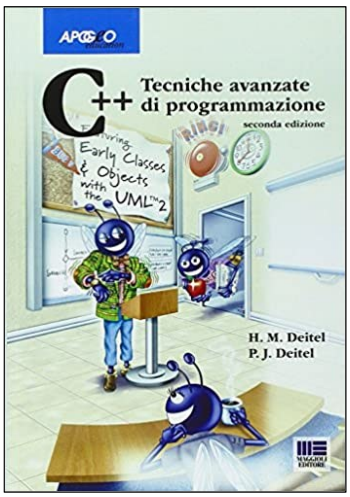
[FdP] H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

3

Libri di testo

Tecniche avanzate e strutture dati elementari

[TAP] H. M. Deitel, P. J. Deitel
C++ Tecniche avanzate di programmazione
II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6



4

Risorse on-line

Team del corso

Programmazione 2 AA 2023-24 - Prof. Catuogno
Comunicazioni, incontri e avvisi per il corso
Codice: **ftomzjx**

Piattaforma e-learning

Programmazione II e Laboratorio di Programmazione II - A.A. 2023-24
Materiale didattico, manualistica, esercitazioni.
URL: https://elearning.uniparthenope.it/course/view.php?id=2386

5

Strutture dati elementari

6

Stack e liste a puntatori

7

Lo stack

- L'implementazione di uno Stack utilizzando un array ha diversi vantaggi:
  - Semplice ed *efficiente*
  - Relativamente robusta (solo memoria statica per la gestione)
  - Utile in moltissimi casi ma…
- ha lo svantaggio di essere poco flessibile nei casi in cui la quantità di memoria richiesta risulti molto variabile a *run time*.

8

Lo stack

- Problemi (comuni a tutte le strutture dati «statiche»):
  - Riallocare lo stack ogni volta che si richiede nuovo spazio è costoso: (occorre poi «ricopiare» i dati nel nuovo stack);
  - Allocare stack molto grandi peggiora l'uso della memoria (inutilizzo)
  - Necessità di buffer di memoria contigui

➡ Diventa quindi preferibile optare per una implementazione basata su una struttura dati *dinamica*

9

Strutture Dati *dinamiche*

- Ne esiste un repertorio vastissimo e variegato

  Le caratteristiche che ci interessano qui:
  - Gestiscono in maniera *scalabile* quantità di memoria variabile nel tempo
  - Allocano nuova memoria solo quando richiesto e possono liberarla quando non è più necessaria
  - Lavorano con aree di memoria non contigue.

10

Liste *a puntatori*

o liste *concatenate*
o linked list…

11

Liste *a puntatori*

- Una lista è una sequenza lineare ordinata di elementi detti *nodi*
  - I nodi sono idealmente divisi in due aree che contengono:
  - I dati (il «carico pagante» o *payload*): *i.e.* gli item contenuti nella lista…
  - Ie informazioni necessarie al funzionamento della lista stessa, *in primis* un puntatore con cui ciascun nodo della lista è collegato (*linked*) con il successivo…
  - Puntatori/riferimenti al primo elemento della lista (*head* o *front*), e possibilmente all'ultimo (*tail* o *back*)
  - Altro, a seconda del tipo di lista e del suo impiego…

12

Liste *a puntatori*

- Una lista, può avere un insieme abbastanza vario di operatori:
  - Operatori di «*ispezione*» *e.g.* **`isEmpty()`**, **`isFull()`**, **`len()`**, …
  - Operatori di «*inserimento*» che aggiungono nodi alla lista. Possono essere diversi a seconda del punto della lista in cui i nuovi nodi sono aggiunti: in testa, in coda o nel mezzo, *e.g.* **`insertAtFront()`**, **`insertAtBack()`**, **`insertAt()`**..
  - Operatori di «*estrazione*», speculari ai precedenti… **`removeFromFront()`**, **`removeFromBack()`**, …
  - Operatori di «*lettura*»: che accedono, in maniera non distruttiva, ai dati della lista; operatori di «ricerca»: che scorrono la lista alla ricerca di un nodo contenente il dato richiesto; operatori specifici dei singoli nodi…

13

Liste *a puntatori*

- Le liste sono una struttura dati molto flessibile, utilissima per implementarne di altre più complesse.
  - Liste «*general purpose*» sono fornite nelle librerie standard dei linguaggi (*e.g.* la classe **`list`** inclusa nelle classi della STL di **C++**) per essere poi personalizzate per derivazione.

`https://en.cppreference.com/w/cpp/container/list`

…noi esamineremo i più comuni tipi di liste seguendo un approccio «incrementale»…

14

Esempio: lista di interi #1

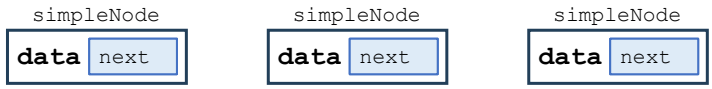
File: **simpleList.hpp**

```
…
4  class simpleNode {
5  private:
6          int data;
7          simpleNode *next;
8  public:
9          simpleNode(int);
10         ~simpleNode() {};
11         void setData(int);
12         int getData();
13         void setNext(simpleNode*);
14         simpleNode *getNext();
15         bool isLast();
16 };
```

I dati contenuti dalla lista sono memorizzati nel campo data dei singoli nodi.

Un nodo di una lista gioca un ruolo analogo alla cella dell'array

Il meccanismo di accesso è diverso…



15

Esempio: lista di interi #1

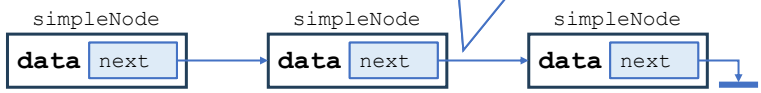
File: **simpleList.hpp**

```
…
4  class simpleNode {
5  private:
6          int data;
7          simpleNode *next;
8  public:
9          simpleNode(int);
10         ~simpleNode() {};
11         void setData(int);
12         int getData();
13         void setNext(simpleNode*);
14         simpleNode *getNext();
15         bool isLast();
16 };
```

…ogni nodo conserva un puntatore a quello che lo segue nella lista.

L'accesso al contenuto della lista è sequenziale.

Per raggiungere un certo nodo intermedio, bisogna scorrere tutti i precedenti



16

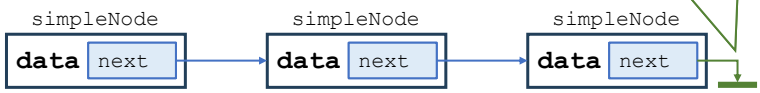
Esempio: lista di interi #1

File: **simpleList.hpp**

```
…
4  class simpleNode {
5  private:
6          int data;
7          simpleNode *next;
8  public:
9          simpleNode(int);
10         ~simpleNode() {};
11         void setData(int);
12         int getData();
13         void setNext(simpleNode*);
14         simpleNode *getNext();
15         bool isLast();
16 };
```

…L'ultimo nodo della lista assegna al suo attributo **next** un apposito mercatore, per indicare che non vi sono altri nodi nella lista.

Generalmente si utilizza la costante **nullptr**



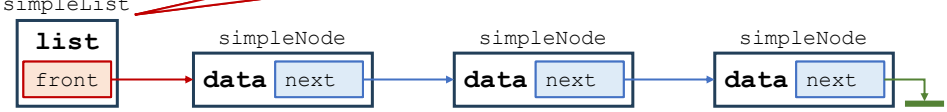
17

Esempio: lista di interi #1

File: **simpleList.hpp**

```
…
17 class simpleList {
18 protected:
19         simpleNode *front;
20 public:
21         simpleList(): front(nullptr) {};
22         virtual ~simpleList();
23         bool isEmpty();
24         virtual bool insertAtFront(int);
25         virtual bool removeFromFront(int &)
26 };
…
```

La lista, nel suo insieme, è rappresentata da una struttura di management che contiene il puntatore al primo nodo (**front**) della lista e altre informazioni necessarie al suo funzionamento (altri puntatori, *metadati*…)

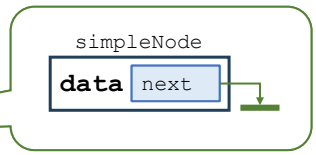


18

Esempio: lista di interi #1

File: **simpleList.cpp**

```
 4  simpleNode::simpleNode(int i) {
 5          data=i;
 6          next=nullptr;
 7  }
 8  void simpleNode::setData(int i) {
 9          data=i;
10  }
11  int simpleNode::getData() {
12          return data;
13  }
14  void simpleNode::setNext(simpleNode *n){
15          next=n;
16  }
17  simpleNode *simpleNode::getNext(){
18          return next;
19  }
20  bool simpleNode::isLast(){
21          return next==nullptr;
22  }
```



19

Esempio: lista di interi #1

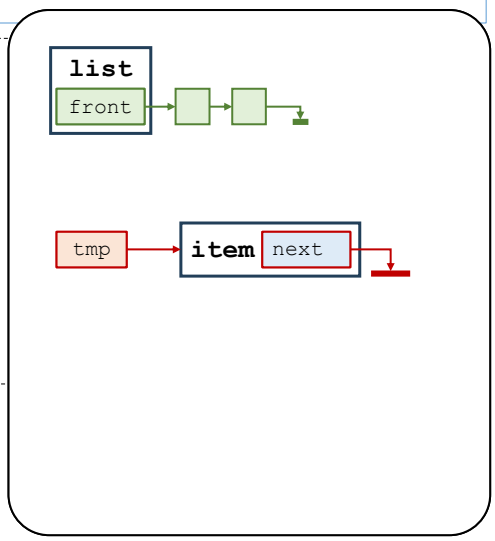
File: **simpleList.cpp**

```
24  bool simpleList::isEmpty() {
25          return front==nullptr;
26  }
27
28  bool simpleList::insertAtFront(int item) {
29          simpleNode *tmp;
30
31          tmp=new simpleNode(item);
32          tmp->setNext(front);
33          front=tmp;
34          return true;
35  }
…   …
```

20

Esempio: lista di interi #1

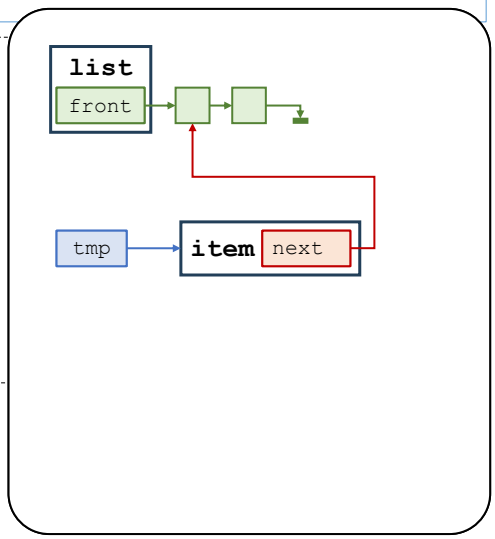
```
24  bool simpleList::isEmpty() {
25          return front==nullptr;
26  }
27
28  bool simpleList::insertAtFront(int item) {
29          simpleNode *tmp;
30
31          tmp=new simpleNode(item);
32          tmp->setNext(front);
33          front=tmp;
34          return true;
35  }
 …  …
```



21

Esempio: lista di interi #1

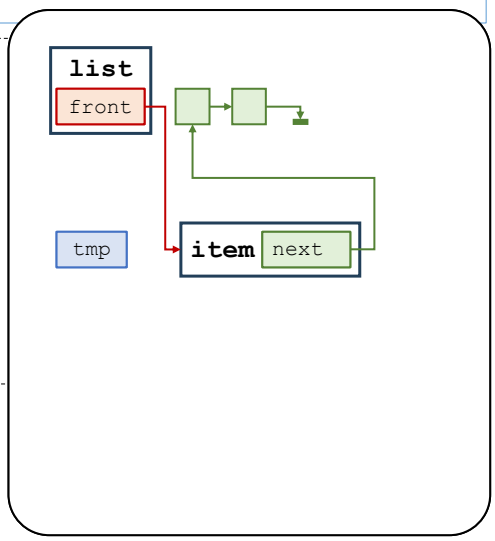
```
24  bool simpleList::isEmpty() {
25          return front==nullptr;
26  }
27
28  bool simpleList::insertAtFront(int item) {
29          simpleNode *tmp;
30
31          tmp=new simpleNode(item);
32          tmp->setNext(front);
33          front=tmp;
34          return true;
35  }
 …  …
```



22

Esempio: lista di interi #1

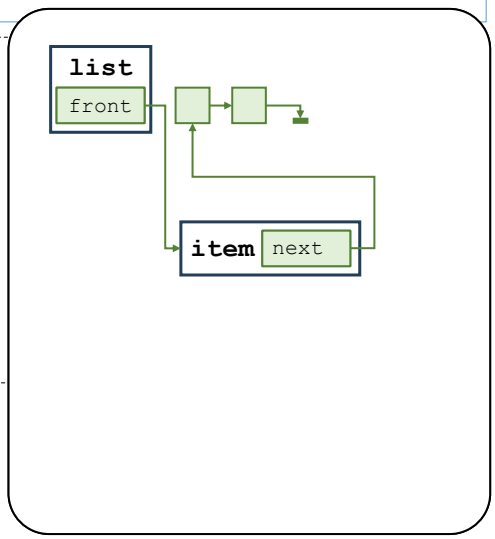
```
24  bool simpleList::isEmpty() {
25          return front==nullptr;
26  }
27
28  bool simpleList::insertAtFront(int item) {
29          simpleNode *tmp;
30
31          tmp=new simpleNode(item);
32          tmp->setNext(front);
33          front=tmp;
34          return true;
35  }
 …  …
```



23

Esempio: lista di interi #1

```
24  bool simpleList::isEmpty() {
25          return front==nullptr;
26  }
27
28  bool simpleList::insertAtFront(int item) {
29          simpleNode *tmp;
30
31          tmp=new simpleNode(item);
32          tmp->setNext(front);
33          front=tmp;
34          return true;
35  }
 …  …
```



24

Esempio: lista di interi #1

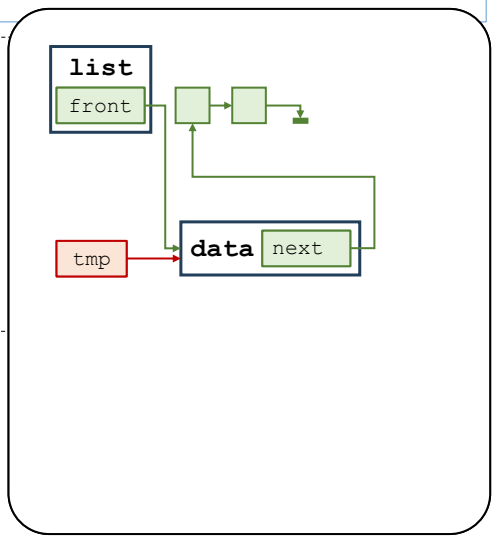
File: **simpleList.cpp**

```
36  bool simpleList::removeAtFront(int &item){
37          simpleNode *tmp;
38          if (isEmpty())
39                  return false;
40          tmp=front;
41          front=front->getNext();
42          item=tmp->getData();
43          delete tmp;
44          return true;
45  }
 …  …
```

25

Esempio: lista di interi #1

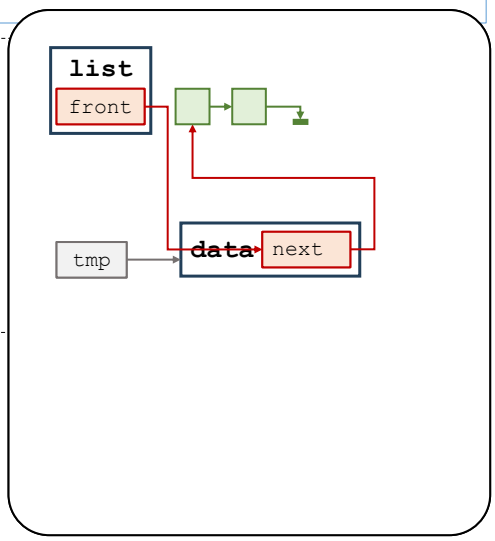
```
36  bool simpleList::removeAtFront(int &item){
37          simpleNode *tmp;
38          if (isEmpty())
39                  return false;
40          tmp=front;
41          front=front->getNext();
42          item=tmp->getData();
43          delete tmp;
44          return true;
45  }
 …  …
```



26

Esempio: lista di interi #1

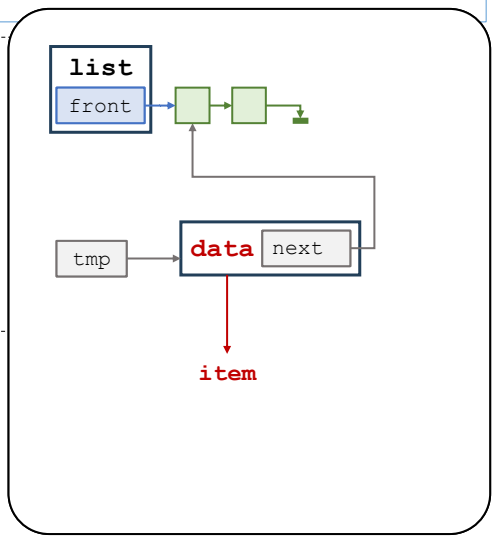
```
36  bool simpleList::removeAtFront(int &item){
37          simpleNode *tmp;
38          if (isEmpty())
39                  return false;
40          tmp=front;
41          front=front->getNext();
42          item=tmp->getData();
43          delete tmp;
44          return true;
45  }
 …  …
```



27

Esempio: lista di interi #1

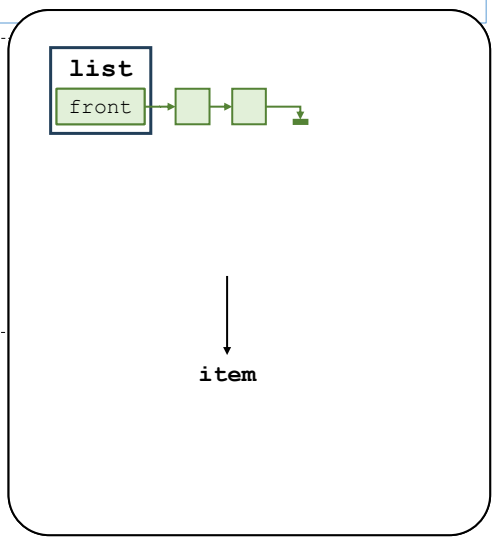
```
36  bool simpleList::removeAtFront(int &item){
37          simpleNode *tmp;
38          if (isEmpty())
39                  return false;
40          tmp=front;
41          front=front->getNext();
42          item=tmp->getData();
43          delete tmp;
44          return true;
45  }
 …  …
```



28

Esempio: lista di interi #1

```
36  bool simpleList::removeAtFront(int &item){
37          simpleNode *tmp;
38          if (isEmpty())
39                  return false;
40          tmp=front;
41          front=front->getNext();
42          item=tmp->getData();
43          delete tmp;
44          return true;
45  }
…   …
```



29

Esempio: lista di interi #1

File: **simpleList.cpp**

```
…   …
36  simpleList::~simpleList(){
37          simpleNode *current;
38
39          while(front!=nullptr){
40                  current=front;
41                  front=front->getNext();
42                  delete current;
43          }
44  }
…   …
```

30

Esempio: lista di interi #1

File: **simpleList.cpp**

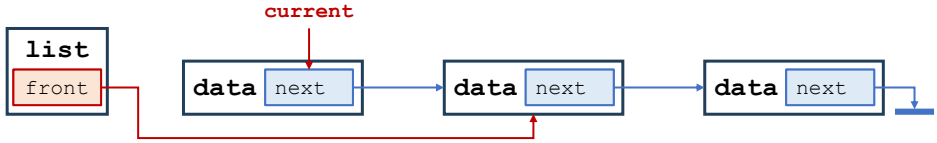
```
…
36 simpleList::~simpleList(){
37         simpleNode *current;
38
39         while(front!=nullptr){
40                 current=front;
41                 front=front->getNext();
42                 delete current;
43         }
44 }
…
```

31

Esempio: lista di interi #1

File: **simpleList.cpp**

```
…
36 simpleList::~simpleList(){
37         simpleNode *current;
38
39         while(front!=nullptr){
40                 current=front;
41                 front=front->getNext();
42                 delete current;
43         }
44 }
…
```

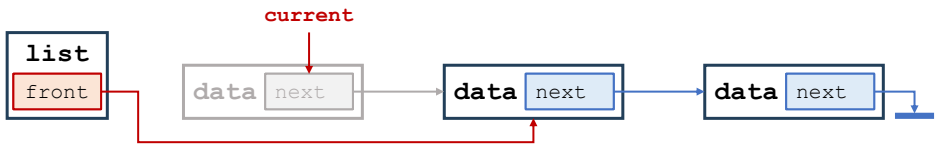


32

Esempio: lista di interi #1

File: **simpleList.cpp**

```
…
36 simpleList::~simpleList(){
37         simpleNode *current;
38
39         while(front!=nullptr){
40                 current=front;
41                 front=front->getNext();
42                 delete current;
43         }
44 }
…
```

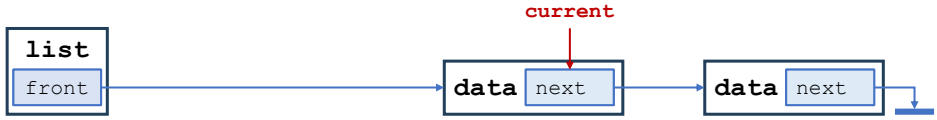


33

Esempio: lista di interi #1

File: **simpleList.cpp**

```
…
36 simpleList::~simpleList(){
37         simpleNode *current;
38
39         while(front!=nullptr){
40                 current=front;
41                 front=front->getNext();
42                 delete current;
43         }
44 }
…
```



34

Esempio: lista di interi #1

File: **simpleList.cpp**

```
…
36 simpleList::~simpleList(){
37         simpleNode *current;
38
39         while(front!=nullptr){
40                 current=front;
41                 front=front->getNext();
42                 delete current;
43         }
44 }
…
```

35

Esempio: lista di interi #1

File: **simpleList.cpp**

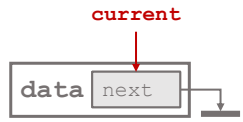
```
…
36 simpleList::~simpleList(){
37         simpleNode *current;
38
39         while(front!=nullptr){
40                 current=front;
41                 front=front->getNext();
42                 delete current;
43         }
44 }
…
```

36

Esempio: lista di interi #1

File: **simpleList.cpp**

```
…
36 simpleList::~simpleList(){
37         simpleNode *current;
38
39         while(front!=nullptr){
40                 current=front;
41                 front=front->getNext();
42                 delete current;
43         }
44 }
…
```



37

Esempio: lista di interi #1

File: **simpleList.cpp**

```
…
36 simpleList::~simpleList(){
37         simpleNode *current;
38
39         while(front!=nullptr){
40                 current=front;
41                 front=front->getNext();
42                 delete current;
43         }
44 }
…
```

38

Esempio: stack di interi #2

- Scriviamo la classe **`StackList`** che implementa uno stack di numeri interi, utilizzando la classe **`simpleList`** :
  - Il costruttore non prende un parametri. Lo stack non ha una taglia predefinita.
  - L'operatore **`isEmpty()`** che restituisce il valore booleano **`true`** se lo stack è vuoto;
  - Gli opertori **`push()`** e **`pop()`** per l'inserimento e l'estrazione di dati nello/dallo stack.

39

Esempio: stack di interi #2

File: **`StackList.hpp`**

```
…
4  class StackList {
5  protected:
6          simpleList *storage;
7  public:
8          StackList();
9          ~StackList();
10         bool isEmpty() const;
11         StackList *push(int);
12         bool pop(int&);
13 };
…
```

40

Esempio: stack di interi #2

File: **StackList.cpp**

```
 4  StackList::StackList() {
 5          storage = new simpleList();
 6  }
 7  StackList::~StackList(){
 8          delete storage;
 9  }
10  bool StackList::isEmpty() const{
11          return storage->isEmpty();
12  }
13
14  StackList *StackList::push(int item) {
15          storage->insertAtFront(item);
16          return this;
17  }
18  bool StackList::pop(int &item) {
19          return storage->removeAtFront(item);
20  }
```

41

La coda (*queue*)

- Uno coda (o *queue,* in inglese) è una struttura dati lineare che consente l'inserimento e l'estrazione dei dati:
  - Singolarmente
  - In maniera *sequenziale*
  - Applicando la politica *«First-in, First-out»* (LIFO)

42

La coda (*queue*)

- E' costituito da:
  - Un'area di memoria in cui *«espandersi»* man mano che i dati vengono inseriti;
  - un puntatore/riferimento alla *testa* della coda (*head*);
  - un puntatore/riferimento alla *fine* della coda (*tail*);
- I dati sono:
  - inseriti sempre in testa mediante l'operatore **`enqueue()`**
  - estratti sempre dal fondo , mediante l'operatore **`dequeue()`**

43

La coda (*queue*)

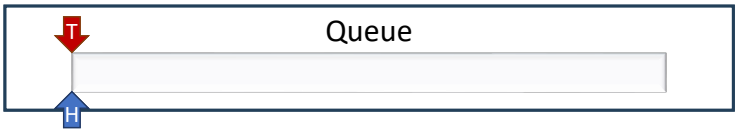
- Come gli stack, anche le code sono una struttura dati estremamente utile e molto utilizzata

  Ordinamento di eventi, controllo/regolazione degli accessi a risorse distribuite…
- Parimenti, in base alle circostanze, gli strumenti e l'ambito di applicazioni, le code sono implementate in vari modi…
  - mediante strutture dati statiche (*i.e.* array)
  - oppure dinamiche (*e.g. liste a puntatori*)

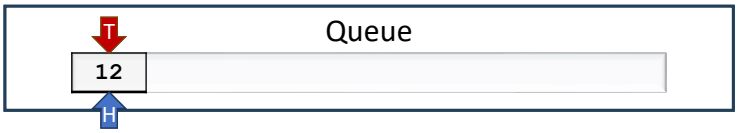
44

La coda (*queue*)

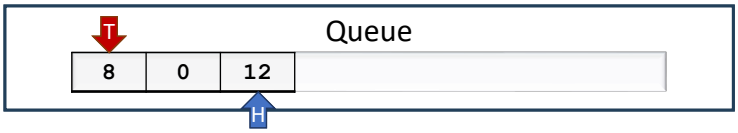
Coda vuota:



enqueue(12)



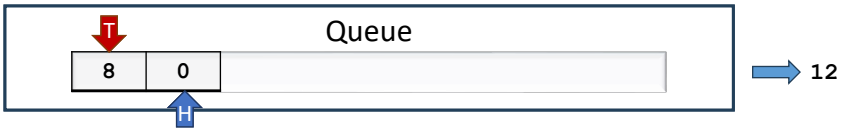
enqueue(0)
enqueue(8)



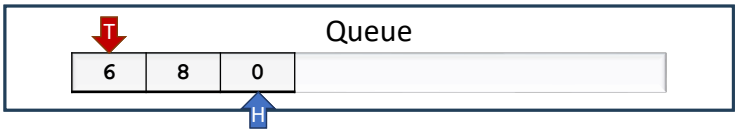
45

La coda (*queue*)

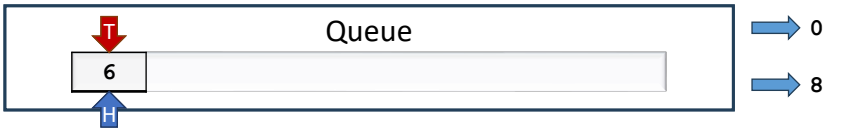
dequeue()



enqueue(6)



dequeue()
dequeue()



46

Esempio: coda di interi

a. Implementiamo la classe **`frontBackList`**, derivata da **`simpleList`**, che implementa anche i metodi **`insertAtBack()`**, **`removeFromBack()`**

b. Utilizziamo questa nuova lista per implementare una classica coda con la classe **`QueueList`**, che espone i metodi:

`isEmpty()`

`enqueue()`

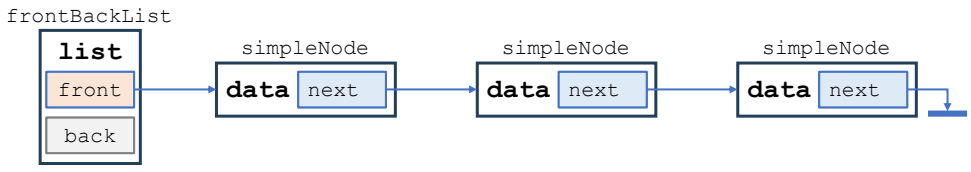
`dequeue()`

47

Coda di interi: la lista

File: **`frontBackList.hpp`**

```
 5  class frontBackList: public simpleList {
 6  protected:
 7          simpleNode *back;
 8  public:
 9          frontBackList(): back(nullptr) {};
10          virtual ~frontBackList() {};
11          bool insertAtFront(int) override;
12          bool removeFromFront(& int) override;
13          virtual bool insertAtBack(int);
14          virtual bool removeFromBack(int &);
15  };
```

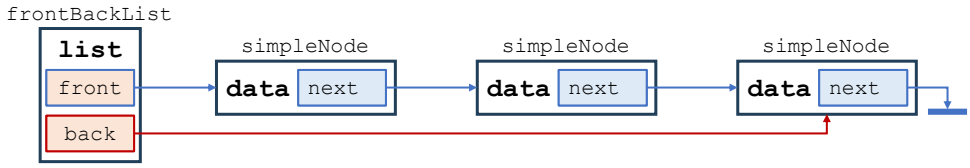


48

Coda di interi: la lista

File: **frontBackList.hpp**

```
 5  class frontBackList: public simpleList {
 6  protected:
 7          simpleNode *back;
 8  public:
 9          frontBackList(): back(nullptr) {};
10          virtual ~frontBackList() {};
11          bool insertAtFront(int) override;
12          bool removeFromFront(& int) override;
13          virtual bool insertAtBack(int);
14          virtual bool removeFromBack(int &);
15  };
```

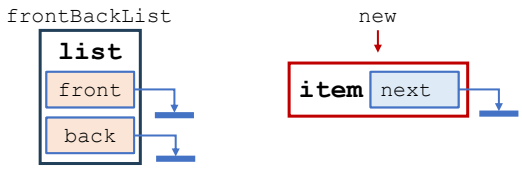


49

Coda di interi: la lista

File: **frontBackList.cpp**

```
…  …
5  bool frontBackList::insertAtFront(int item) {
6          if(isEmpty()){
7                  front=back=new simpleNode(item);
8                  return true;
9          }
10         return simpleList::insertAtFront(item);
11 }
…  …
```

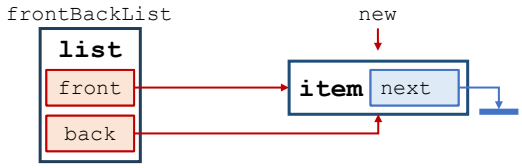


50

Coda di interi: la lista

File: **frontBackList.cpp**

```
…
5  bool frontBackList::insertAtFront(int item) {
6          if(isEmpty()){
7                  front=back=new simpleNode(item);
8                  return true;
9          }
10         return simpleList::insertAtFront(item);
11 }
…
```

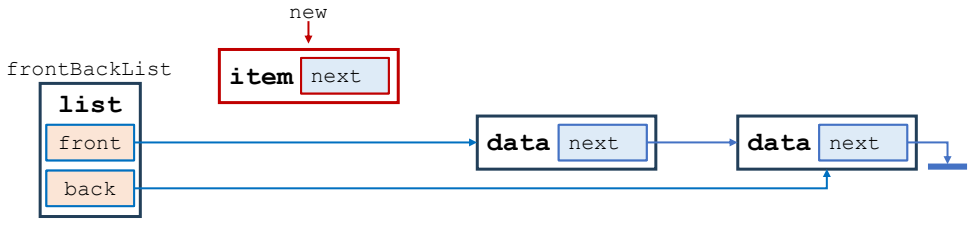


51

Coda di interi: la lista

File: **frontBackList.cpp**

```
…
5  bool frontBackList::insertAtFront(int item) {
6          if(isEmpty()){
7                  front=back=new simpleNode(item);
8                  return true;
9          }
10         return simpleList::insertAtFront(item);
11 }
…
```

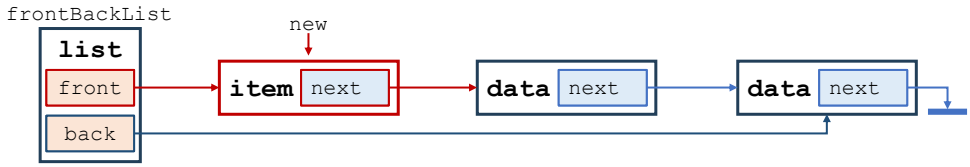


52

Coda di interi: la lista

File: **frontBackList.cpp**

```
…
5  bool frontBackList::insertAtFront(int item) {
6          if(isEmpty()){
7                  front=back=new simpleNode(item);
8                  return true;
9          }
10         return simpleList::insertAtFront(item);
11 }
…
```

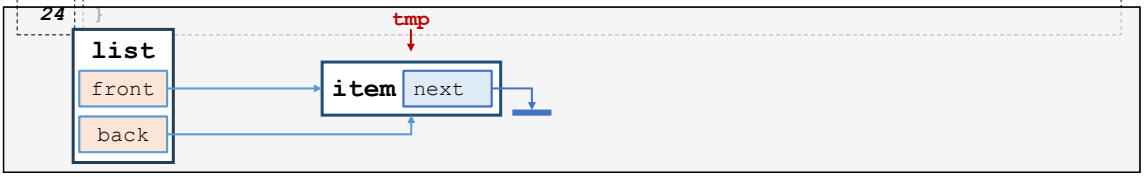


53

Coda di interi: la lista

File: **frontBackList.cpp**

```
12 bool frontBackList::removeFromFront(int &item)
13         simpleNode *tmp;
14         if(isEmpty())
15                 return false;
16         if(front==back){
17                 tmp=front;
18                 front=back=nullptr;
19                 item=tmp->getData();
20                 delete tmp;
21                 return true;
22         } else
23                 return simpleList::removeFromFront(item);
24 }
```

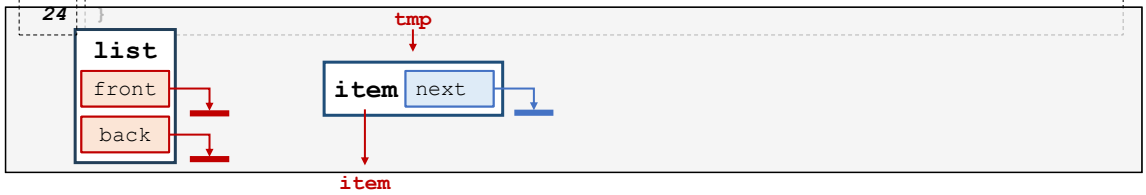


54

Coda di interi: la lista

File: **frontBackList.cpp**

```
12  bool frontBackList::removeFromFront(int &item)
13          simpleNode *tmp;
14          if(isEmpty())
15                  return false;
16          if(front==back){
17                  tmp=front;
18                  front=back=nullptr;
19                  item=tmp->getData();
20                  delete tmp;
21                  return true;
22          } else
23                  return simpleList::removeFromFront(item);
24  }
```

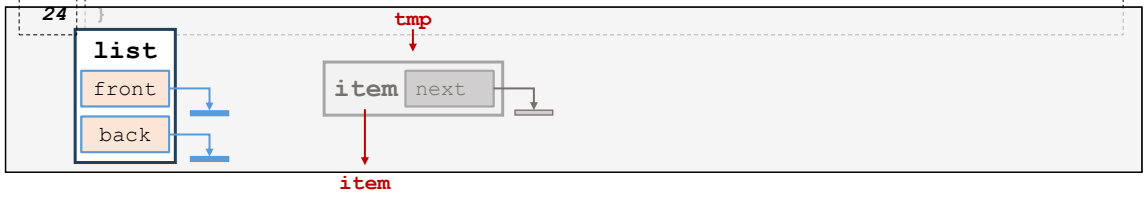


55

Coda di interi: la lista

File: **frontBackList.cpp**

```
12  bool frontBackList::removeFromFront(int &item)
13          simpleNode *tmp;
14          if(isEmpty())
15                  return false;
16          if(front==back){
17                  tmp=front;
18                  front=back=nullptr;
19                  item=tmp->getData();
20                  delete tmp;
21                  return true;
22          } else
23                  return simpleList::removeFromFront(item);
24  }
```

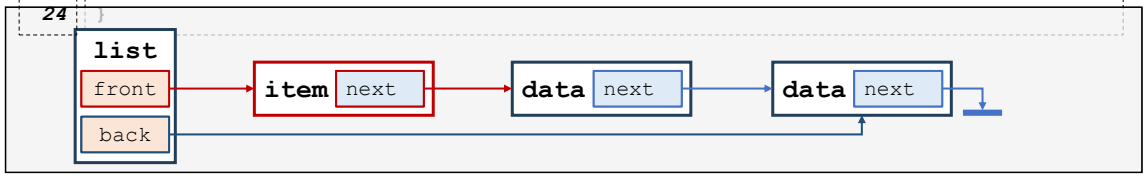


56

Coda di interi: la lista

File: **frontBackList.cpp**

```
12  bool frontBackList::removeFromFront(int &item)
13          simpleNode *tmp;
14          if(isEmpty())
15                  return false;
16          if(front==back){
17                  tmp=front;
18                  front=back=nullptr;
19                  item=tmp->getData();
20                  delete tmp;
21                  return true;
22          } else
23                  return simpleList::removeFromFront(item);
24  }
```

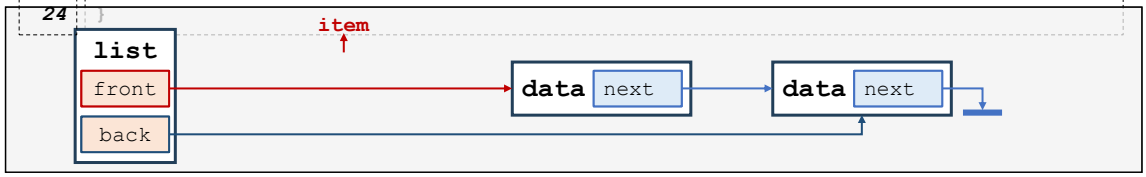


57

Coda di interi: la lista

File: **frontBackList.cpp**

```
12  bool frontBackList::removeFromFront(int &item)
13          simpleNode *tmp;
14          if(isEmpty())
15                  return false;
16          if(front==back){
17                  tmp=front;
18                  front=back=nullptr;
19                  item=tmp->getData();
20                  delete tmp;
21                  return true;
22          } else
23                  return simpleList::removeFromFront(item);
24  }
```



58

Coda di interi: la lista

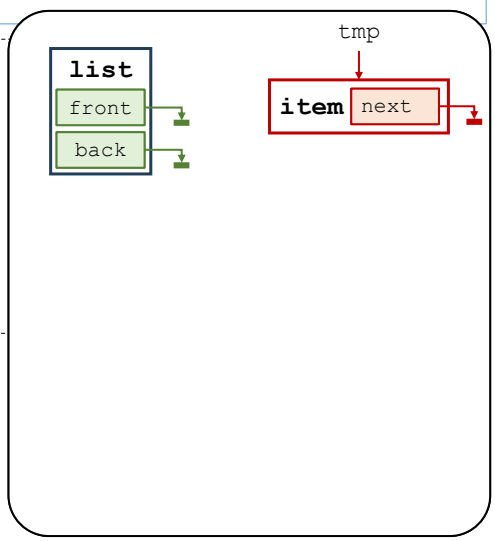
File: **frontBackList.cpp**

```
25  bool frontBackList::insertAtBack(int item) {
26          simpleNode *tmp;
27          tmp=new simpleNode(item);
28          if(isEmpty()) {
29                  front=back=tmp;
30                  return true;
31          }
32          back->setNext(tmp);
33          back=tmp;
34          return true;
35  }
```

59

Coda di interi: la lista

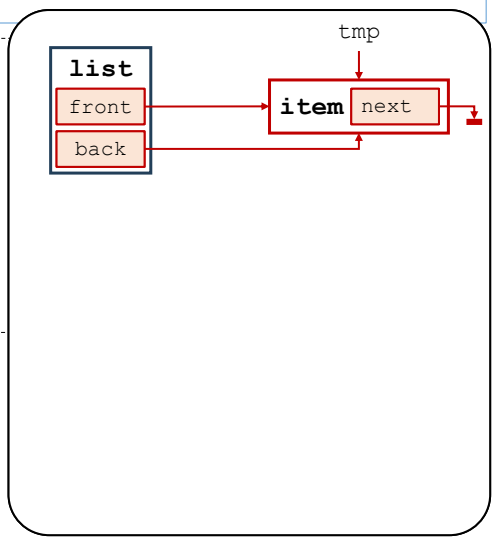
```
25  bool frontBackList::insertAtBack(int item)
26          simpleNode *tmp;
27          tmp=new simpleNode(item);
28          if(isEmpty()) {
29                  front=back=tmp;
30                  return true;
31          }
32          back->setNext(tmp);
33          back=tmp;
34          return true;
35  }
```



60

Coda di interi: la lista

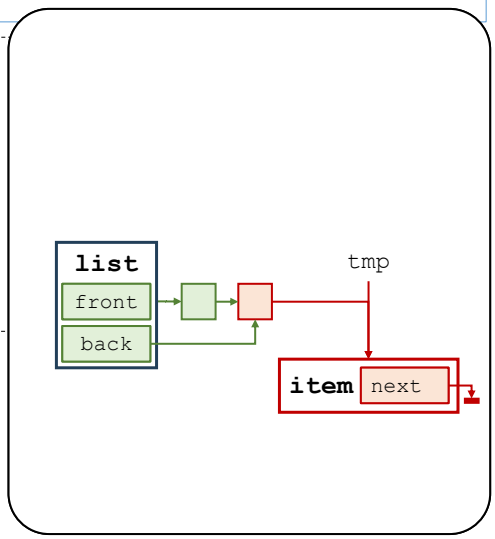
```
25  bool frontBackList::insertAtBack(int item)
26          simpleNode *tmp;
27          tmp=new simpleNode(item);
28          if(isEmpty()) {
29                  front=back=tmp;
30                  return true;
31          }
32          back->setNext(tmp);
33          back=tmp;
34          return true;
35  }
```



61

Coda di interi: la lista

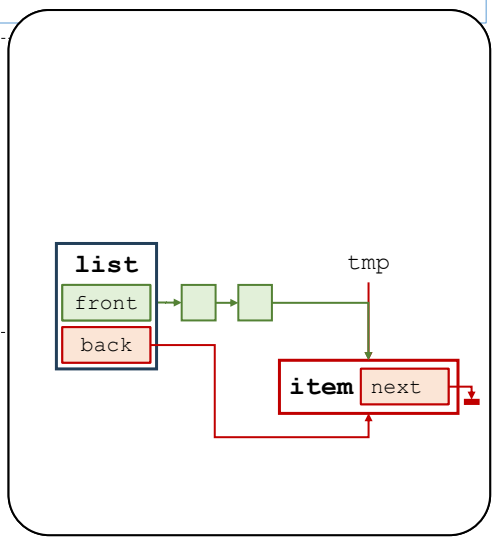
```
25  bool frontBackList::insertAtBack(int item)
26          simpleNode *tmp;
27          tmp=new simpleNode(item);
28          if(isEmpty()) {
29                  front=back=tmp;
30                  return true;
31          }
32          back->setNext(tmp);
33          back=tmp;
34          return true;
35  }
```



62

Coda di interi: la lista

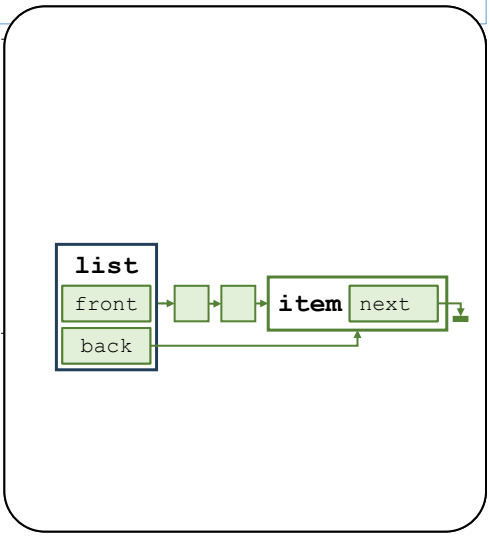
```
25  bool frontBackList::insertAtBack(int item)
26          simpleNode *tmp;
27          tmp=new simpleNode(item);
28          if(isEmpty()) {
29                  front=back=tmp;
30                  return true;
31          }
32          back->setNext(tmp);
33          back=tmp;
34          return true;
35  }
```



63

Coda di interi: la lista

```
25  bool frontBackList::insertAtBack(int item)
26          simpleNode *tmp;
27          tmp=new simpleNode(item);
28          if(isEmpty()) {
29                  front=back=tmp;
30                  return true;
31          }
32          back->setNext(tmp);
33          back=tmp;
34          return true;
35  }
```



64

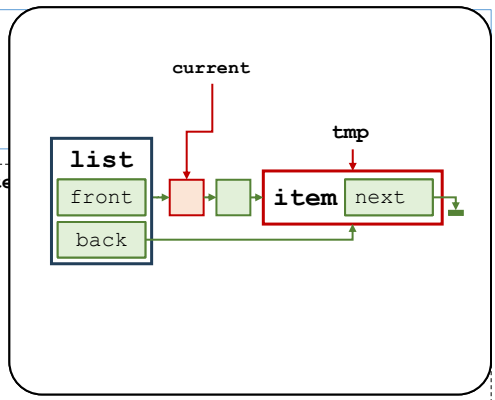
Coda di interi: la lista

File: **frontBackList.cpp**

```
36 bool frontBackList::removeFromBack(int &item)
37         simpleNode *tmp,*current;
38         if(isEmpty())
39                 return false;
40         tmp=back;
41         if (front==back) {
42                 front=back=nullptr;
43         } else {
44                 current=front;
45                 while(current->getNext()!=back)
46                         current=current->getNext();
47                 back=current;
48                 current->setNext(nullptr);
49         }
50         item=tmp->getData();
51         delete tmp;
52         return true;
53 }
```

65

Coda di interi: la lista

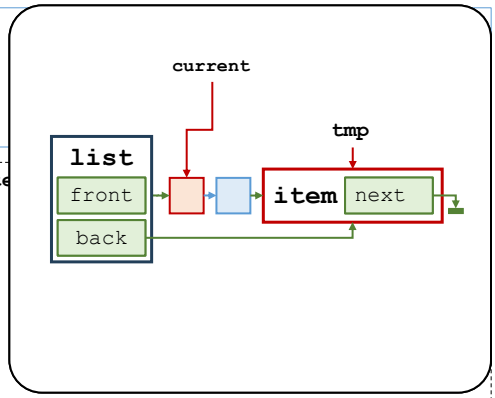



```
36 bool frontBackList::removeFromBack(int &ite
37         simpleNode *tmp,*current;
38         if(isEmpty())
39                 return false;
40         tmp=back;
41         if (front==back) {
42                 front=back=nullptr;
43         } else {
44                 current=front;
45                 while(current->getNext()!=back)
46                         current=current->getNext();
47                 back=current;
48                 current->setNext(nullptr);
49         }
50         item=tmp->getData();
51         delete tmp;
52         return true;
53 }
```

66

Coda di interi: la lista

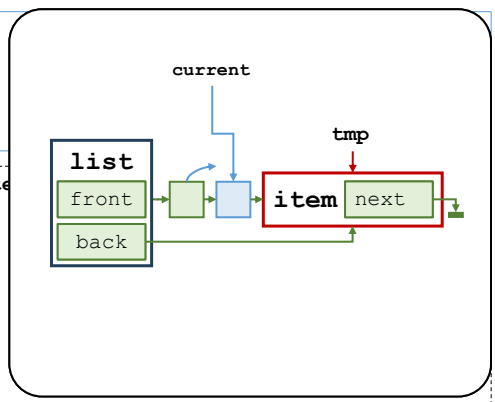
```
36  bool frontBackList::removeFromBack(int &ite
37          simpleNode *tmp,*current;
38          if(isEmpty())
39                  return false;
40          tmp=back;
41          if (front==back) {
42                  front=back=nullptr;
43          } else {
44                  current=front;
45                  while(current->getNext()!=back)
46                          current=current->getNext();
47                  back=current;
48                  current->setNext(nullptr);
49          }
50          item=tmp->getData();
51          delete tmp;
52          return true;
53  }
```



67

Coda di interi: la lista

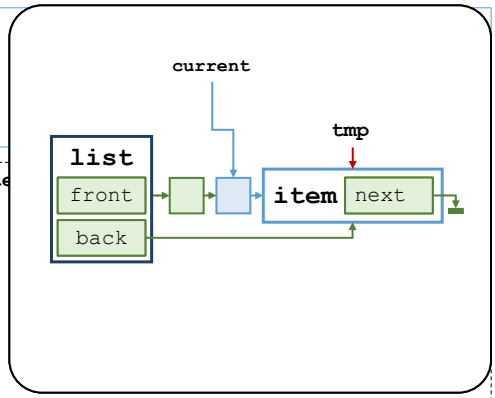
```
36  bool frontBackList::removeFromBack(int &ite
37          simpleNode *tmp,*current;
38          if(isEmpty())
39                  return false;
40          tmp=back;
41          if (front==back) {
42                  front=back=nullptr;
43          } else {
44                  current=front;
45                  while(current->getNext()!=back)
46                          current=current->getNext();
47                  back=current;
48                  current->setNext(nullptr);
49          }
50          item=tmp->getData();
51          delete tmp;
52          return true;
53  }
```



68

Coda di interi: la lista

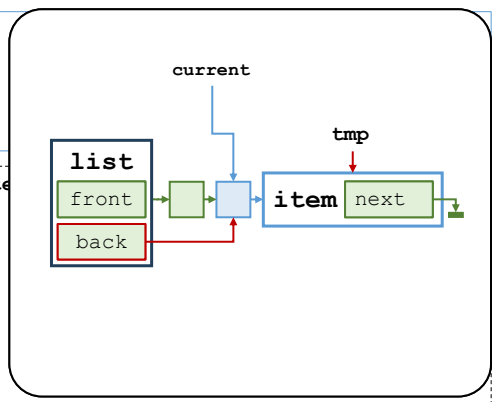
```
36  bool frontBackList::removeFromBack(int &ite
37          simpleNode *tmp,*current;
38          if(isEmpty())
39                  return false;
40          tmp=back;
41          if (front==back) {
42                  front=back=nullptr;
43          } else {
44                  current=front;
45                  while(current->getNext()!=back)
46                          current=current->getNext();
47                  back=current;
48                  current->setNext(nullptr);
49          }
50          item=tmp->getData();
51          delete tmp;
52          return true;
53  }
```



69

Coda di interi: la lista

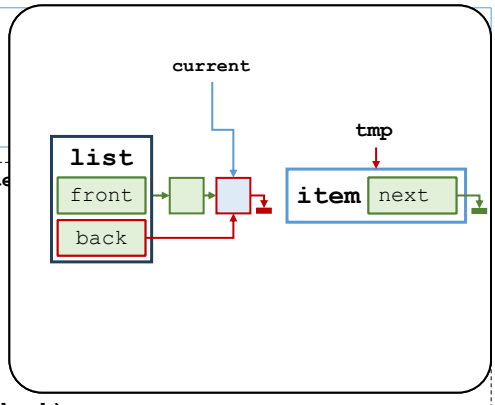
```
36  bool frontBackList::removeFromBack(int &ite
37          simpleNode *tmp,*current;
38          if(isEmpty())
39                  return false;
40          tmp=back;
41          if (front==back) {
42                  front=back=nullptr;
43          } else {
44                  current=front;
45                  while(current->getNext()!=back)
46                          current=current->getNext();
47                  back=current;
48                  current->setNext(nullptr);
49          }
50          item=tmp->getData();
51          delete tmp;
52          return true;
53  }
```



70

Coda di interi: la lista

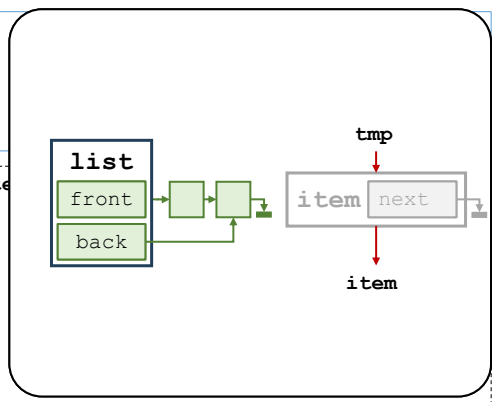
```
36  bool frontBackList::removeFromBack(int &ite
37          simpleNode *tmp,*current;
38          if(isEmpty())
39                  return false;
40          tmp=back;
41          if (front==back) {
42                  front=back=nullptr;
43          } else {
44                  current=front;
45                  while(current->getNext()!=back)
46                          current=current->getNext();
47                  back=current;
48                  current->setNext(nullptr);
49          }
50          item=tmp->getData();
51          delete tmp;
52          return true;
53  }
```



71

Coda di interi: la lista

```
36  bool frontBackList::removeFromBack(int &ite
37          simpleNode *tmp,*current;
38          if(isEmpty())
39                  return false;
40          tmp=back;
41          if (front==back) {
42                  front=back=nullptr;
43          } else {
44                  current=front;
45                  while(current->getNext()!=back)
46                          current=current->getNext();
47                  back=current;
48                  current->setNext(nullptr);
49          }
50          item=tmp->getData();
51          delete tmp;
52          return true;
53  }
```



72

Coda di interi: la classe **QueueList**

File: **QueueList.hpp**

```
 5  class QueueList {
 6  private:
 7          frontBackList *storage;
 8  public:
 9          QueueList();
10          ~QueueList();
11          bool isEmpty();
12          bool enqueue(int);
13          bool dequeue(int&);
14  };
```

73

Coda di interi: la classe **QueueList**

File: **QueueList.cpp**

```
 5  QueueList::QueueList() {
 6          storage=new frontBackList();
 7  }
 8  QueueList::~QueueList() {
 9          delete storage;
10  }
11  bool QueueList::isEmpty() {
12          return storage->isEmpty();
13  }
14  bool QueueList::enqueue(int item){
15          return storage->insertAtBack(item);
16  }
17  bool QueueList::dequeue(int &item){
18          return storage->removeFromFront(item);
19  }
```

74

Esercizio: liste, stack e code con i *template*

- Scriviamo una nuova classe **`frontBackList`** che parametrizzi il tipo degli item e utilizziamola per implementare una nuova versione delle classi **`stacklist`** e **`queuelist`**.

  Si scriva un breve programma demo in cui si utilizzano diverse strutture dati con item di tipo diverso.

```
template<class ItemType>
class frontBackList {
        …
};
template<class ItemType>
class StackList {
        …
};
```

75