6. Contenitori standard

I contenitori della libreria STL sono classificati in tre categorie:

- **Sequenze**:

  vector
  list
  double-ended queues (deque)

- **Adattatori**

  stack
  queue
  priority queue

- **Contenitori associativi**

  map
  multimap
  set
  multiset

Le caratteristiche della sequenza **deque** sono illustrate successivamente con un esempio.
Gli **adattatori** si chiamano così perché sono un'implementazione delle sequenze standard (*vector*, *list* e *deque*) adattate in modo da risolvere specifici problemi applicativi: **stack** e **queue** rappresentano le strutture di pila e coda di cui si è parlato nel Capitolo 7, mentre il contenitore **priority queue** consente di organizzare i dati in modo ordinato, con in testa il valore più grande.
Questi contenitori possiedono i metodi **push()** e **pop()**, che implementano le operazioni di inserimento e di estrazione degli elementi, spiegate nel Capitolo 7.
I **contenitori associativi**, oltre a *map*, presentato nel paragrafo precedente, sono: **multimap**, per consentire l'uso di mappe nelle quali il valore della chiave non è univoco; **set**, un insiemi di dati di valore diverso tra loro, secondo il senso matematico del termine *insieme*; a questo contenitore si possono applicare le operazioni di unione, intersezione e differenza; **multiset**, un contenitore *set* nel quale uno stesso valore può essere presente più volte.
Per usare un contenitore occorre inserire nel programma la dichiarazione di inclusione, scrivendo il nome del contenitore tra parentesi angolari dopo la direttiva *#include*; per esempio, per usare il contenitore *stack*, si deve scrivere:

```
#include <stack>
```

Vediamo ora un esempio di applicazione del contenitore **deque**, che è una sequenza di dati come i contenitori *vector* e *list*. *Deque* (si legge *deck*) è l'abbreviazione di **double-ended queue** (*coda bifronte* o *coda a due ingressi*), cioè una coda nella quale è possibile inserire e togliere elementi in testa e alla fine.
Il contenitore *deque* possiede alcuni tra i metodi già visti per *vector* e *list*. In particolare, nell'esempio seguente verranno usati i metodi **push_back()** e **push_front()** per gli inserimenti alla fine o all'inizio della coda, **pop_front()** per l'estrazione del primo elemento della coda e **empty()** per controllare se la coda è vuota.
Il programma seguente utilizza anche il metodo **front()**, che restituisce il primo elemento della coda.

PROGETTO

Una macchina accetta lavorazioni aventi un codice identificativo di tipo stringa; al momento dell'inserimento di un nuovo codice occorre specificare anche la priorità della lavorazione: se la priorità è uguale a 1 (priorità alta) il codice della lavorazione viene inserito all'inizio della coda, se la priorità è uguale a 2 (priorità bassa) il codice della lavorazione viene inserito alla fine della coda.
Quando la macchina termina una lavorazione, manda in esecuzione la lavorazione che si trova in testa alla coda.

Il programma è organizzato con un menu che presenta le scelte possibili per l'utente:

- accodamento di una nuova lavorazione;
- esecuzione di una lavorazione;
- visualizzazione delle lavorazioni in coda.

Il contenitore delle lavorazioni è una coda di tipo *deque*:

```
deque<string>  lavoro;
```

Il valore della priorità determina la modalità di inserimento, all'inizio o alla fine della coda:

```
if (priorita == 1) lavoro.push_front(codlav);
else  lavoro.push_back(codlav);
```

L'avvio di una lavorazione viene simulata dal programma attraverso la visualizzazione del codice lavorazione che si trova in prima posizione nella coda e nell'estrazione del primo elemento dalla coda:

```
cout << "Avvio lavorazione: " << lavoro.front() << endl;
lavoro.pop_front();
```

Alla richiesta di esecuzione di una lavorazione, il programma controlla anche che la coda non sia vuota.

La visualizzazione dell'intera coda è realizzata con una ripetizione che usa un iteratore, come già visto per i contenitori presentati nei paragrafi precedenti.

```
deque<string>::iterator  i;
```

Il programma completo è riportato di seguito.

PROGRAMMA C++

```
// CDeque.cpp: contenitore deque
#include <iostream>
#include <deque>
#include <string>
using namespace std;

     deque<string> lavoro; // contenitore delle lavorazioni
```

```
// prototipi delle funzioni
int MenuScelte();
void Accoda();
void Esegui();
void Visualizza();

// funzione principale
int main()
{
   int scelta;
   bool finito = false;

   do {
      scelta = MenuScelte();
      switch (scelta) {
      case 1:
         Accoda();
         break;
      case 2:
         Esegui();
         break;
      case 3:
         Visualizza();
         break;
      case 4:
         cout << "Fine lavoro" << endl;
         finito = true;
      }
   } while (!finito);

   return 0;
}

// menu delle scelte
int MenuScelte()
{
   int s;

   cout << "--------------------------------" << endl;
   cout << "1. Accodamento nuova lavorazione" << endl;
   cout << "2. Esecuzione" << endl;
   cout << "3. Visualizza coda lavorazioni" << endl;
   cout << "4. Fine lavoro" << endl;
   cout << "Scegli: ";
   cin >> s;
   cout << "--------------------------------" << endl;

   return s;
}

// accodamento
void Accoda()
{
   int priorita;         // priorità (1, 2)
   string codlav;        // codice lavorazione
```

```
   cout << "Priorita' (1, 2): ";
   cin >> priorita;
   cout << "Codice lavorazione: ";
   cin >> codlav;
   if (priorita == 1) lavoro.push_front(codlav);
   else lavoro.push_back(codlav);
} // Accoda

// esecuzione
void Esegui()
{
   if (lavoro.empty())
       cout << "Coda lavorazioni vuota" << endl;
   else {
       cout << "Avvio lavorazione: " << lavoro.front() << endl;
       lavoro.pop_front();
   }
} // Esegui

// visualizzazione coda lavorazioni
void Visualizza()
{
   cout << "----------------------" << endl;
   cout << "Coda delle lavorazioni" << endl;
   // lettura con iteratore
   deque<string>::iterator i;
   for (i=lavoro.begin(); i!=lavoro.end(); i++)
       cout << *i << endl;
   cout << "----------------------" << endl;
} // Visualizza
```

ESERCIZI

1 Completare la seguente tabella classificando nelle categorie indicate sulle colonne i contenitori riportati sulle righe:

	Sequenze	Adattatori	Contenitori associativi
deque			
list			
map			
multimap			
multiset			
priority queue			
queue			
set			
stack			
vector			

2 Utilizzare una coda a due ingressi per memorizzare i titoli delle canzoni preferite: le canzoni che si vogliono ascoltare per prime sono poste all'inizio della coda, le altre alla fine. Simulare un riproduttore di motivi musicali con le seguenti funzionalità: accettazione di un titolo dalla tastiera, visualizzazione del titolo della canzone in testa alla coda che viene mandata in riproduzione, lista completa delle canzoni in attesa.