Gestione di pila e coda con i puntatori

I puntatori trovano un interessante utilizzo nella gestione delle pile e delle code, permettendo di superare i limiti imposti dall'utilizzo degli array.
Il concetto di base è quello di associare ad ogni singolo dato un puntatore che rimandi al dato successivo. Così, se dobbiamo costruire una lista di nomi di persone, possiamo organizzare i dati in questo modo:

Appare chiaro che ogni elemento dello schema precedente è una struttura composta da due campi, il primo che contiene un dato di tipo alfanumerico, il secondo un dato di tipo puntatore. Pertanto, possiamo definire la struttura *Nodo* utile per definire un elemento della coda o della pila:

```
struct Nodo {
   string nome;
   Nodo* succ;
};
```

succ è un dato di tipo puntatore alla struttura stessa, che contiene l'indirizzo all'elemento successivo della lista.

Con la struttura di dati precedente è possibile definire dati di tipo puntatore come illustrato di seguito:

```
Nodo* Primo; // puntatore al primo nodo della lista
```

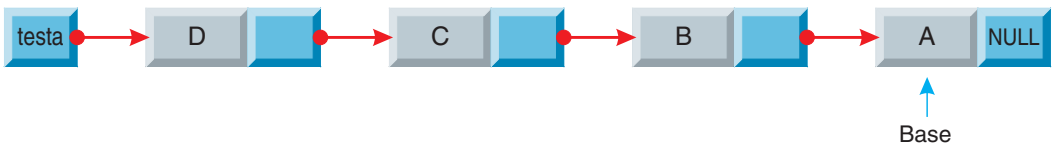
Con riferimento alla figura precedente, poiché ogni nodo è composto da una struttura, per individuare un campo della struttura dobbiamo usare l'**operatore ->** (freccia) e, quindi, possiamo dire che *Primo–>nome* contiene "Luigi" e *Primo–>succ* contiene l'**indirizzo** della struttura con il nodo di Piero.

ESEMPIO

Organizzare un programma per la gestione dinamica di una pila di dati.

DESCRIZIONE DEL PROBLEMA E TRACCIA PER LA SOLUZIONE

L'ultimo elemento caricato nella pila è individuato dal puntatore *testa*, mentre il puntatore di inizio o di base della pila è uguale a **NULL**, che significa nessun indirizzo.



Le operazioni di *Push* e di *Pop* si realizzano modificando il contenuto del puntatore *testa*.

Rispetto all'organizzazione generale del programma che fa ricorso agli array, sono state introdotte le funzioni *Inizializza* e *DistruggiPila*: la prima è composta da un'unica istruzione che assegna NULL al puntatore di testa; la seconda esegue tante operazioni di *Pop* finché la pila diventa vuota.

Le funzioni di *Push* e *Pop* intervengono sul puntatore di testa. La funzione *ScriviPila* è costituita da una ripetizione precondizionale che controlla quando si incontra il puntatore a NULL alla base della pila.

Si deve osservare che, usando i puntatori per gestire la struttura di dati, occorre introdurre due nuove operazioni:

- l'allocazione di uno spazio di memoria per l'area puntata da un nuovo puntatore, durante l'operazione di *Push*;
- il rilascio dello spazio di memoria puntato dal puntatore, durante l'operazione di *Pop*.

Il programma usa le istruzioni **new** e **delete** per allocare e rilasciare la memoria referenziata dai puntatori.

Di seguito sono riportate le implementazioni delle funzioni di gestione della pila. Il programma deve essere completato con il menu delle scelte per l'utente.

```
// inizializzazione della pila
void Inizializza()
{
   testa = NULL;
} // Inizializza

// estrazione di un nodo della pila
void Pop()
{
   Nodo* p;
   if (testa == NULL)
       cout << "Pila vuota. Nessun nodo estratto" << endl;
   else {
       cout << "Nodo estratto = " << testa->dato << endl;
       p = testa->succ;
       delete testa;
       testa = p;
   }
} // Pop

// inserimento di un nodo nella pila
void Push()
{
   int DatoInp;                    // dato da inserire
   Nodo* nuovo;                    // puntatore ad un nuovo nodo
   cout << "Dato da inserire = ";
   cin >> DatoInp;
   nuovo = new Nodo;               // alloca la memoria per il nuovo nodo
   nuovo->dato = DatoInp;
   nuovo->succ = testa;
   testa = nuovo;
} // Push
```

```
// visualizzazione della pila
void ScriviPila()
{
   Nodo* p;
   cout << "Testa della pila" << endl;
   p = testa;
   while (p != NULL) {
       cout << p->dato << endl;
       p = p->succ;
   }
   cout << "Base della pila" << endl;
} // ScriviPila

// toglie tutti i nodi della pila
void DistruggiPila()
{
   while (testa != NULL) {
       Pop();
   }
   cout << "Pila vuota" << endl;
} // DistruggiPila
```

Utilizzando come riferimento il programma precedente e l'implementazione della pila con gli array, si possono risolvere i seguenti problemi:

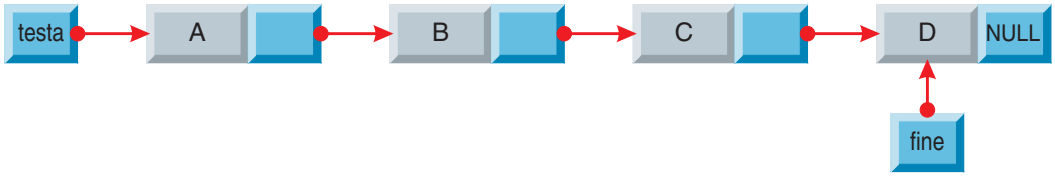
- Scrivere il programma per trasferire i dati di un array in una pila gestita con i puntatori.
- Scrivere il programma per trasferire i dati di una pila, gestita con i puntatori, in una pila gestita con un array.

ESEMPIO

Organizzare un programma per la gestione dinamica di una coda di dati.

DESCRIZIONE DEL PROBLEMA E TRACCIA PER LA SOLUZIONE

Nell'Implementazione di una coda con una struttura dinamica di dati, la differenza principale rispetto alla pila sta nel fatto che l'ultimo elemento a destra nella figura è puntato dal puntatore *fine*.



Ciò è dovuto al fatto che le operazioni di inserimento di un nuovo nodo (*Push*) avvengono alla fine e non in testa alla coda.

L'operazione di *Pop*, invece, agisce sul puntatore di testa in modo del tutto analogo a quanto già visto per la pila.

Le uniche differenze con il programma precedente, per la gestione di una pila, stanno nel fatto che si deve definire un nuovo puntatore *fine* e che si deve modificare l'algoritmo di *Push*.

Anche il programma C++ seguente utilizza l'istruzione **new** per allocare lo spazio in memoria per un nuovo nodo della coda nell'operazione di *Push* e l'istruzione **delete** per rilasciare la memoria puntata da un puntatore nell'operazione di *Pop*.

Viene presentata di seguito, come traccia per la soluzione del problema, l'implementazione della funzione *Push*.

```
// inserimento di un nodo nella coda
void Push()
{
   int DatoInp;                    // dato da inserire
   Nodo* nuovo;                    // puntatore ad un nuovo nodo
   cout << "Dato da inserire = ";
   cin >> DatoInp;
   nuovo = new struct Nodo;        // alloca la memoria per il nuovo nodo
   nuovo->dato = DatoInp;
   nuovo->succ = NULL;
   if (testa == NULL) testa = nuovo;
   else fine->succ = nuovo;
   fine = nuovo;
} // Push
```

Utilizzando come riferimento il programma precedente e l'implementazione della coda con gli array, si possono risolvere i seguenti problemi:

- Scrivere il programma per trasferire i dati di un array in una coda gestita con i puntatori.
- Scrivere il programma per trasferire i dati di una coda, gestita con i puntatori, in una coda gestita con un array.