

Programmazione

Andrea Malvezzi

Settembre 2024 - ..

Contents

1	HEAP e STACK	3
1.1	L'HEAP	3
1.2	Lo STACK	3
2	Puntatori	3
2.1	Cosa sono e come si definiscono	3
2.2	Operazioni con i puntatori	4
2.2.1	NULL	4
2.2.2	Operatore &	4
2.2.3	Dereferenziazione (*)	4
2.2.4	Istruzione new	5
2.2.5	Istruzione delete	5
2.3	L'istruzione typedef	5
3	Strutture dati dinamiche	6
3.1	Le liste	6

1 HEAP e STACK

L'HEAP e lo STACK sono due divisioni della memoria in cui vengono salvati i dati durante l'esecuzione di un programma. I due svolgono due ruoli diversi ma fondamentali:

1.1 L'HEAP

L'HEAP è una porzione di memoria dedicata all'allocazione dinamica delle variabili, quindi mediante istruzioni come `new` e `delete`. Grazie a questa dinamicità, qui si possono sfruttare strutture dati di lunghezza indefinita, come liste o alberi, ma con una rapidità limitata.

1.2 Lo STACK

Lo STACK è una porzione di memoria dedicata alla gestione automatica (quindi a cura del compilatore) di variabili locali e simili. Accedere ai dati al suo interno risulta veloce grazie alla filosofia LIFO (Last-In-First-Out).

2 Puntatori

2.1 Cosa sono e come si definiscono

Un puntatore è uno speciale tipo di dato capace di immagazzinare un indirizzo di memoria corrispondente ad un altro dato.

Quando si dichiara un puntatore bisogna specificare il tipo di dato a cui lo si vuole far puntare. Vediamo un esempio pratico:

```
1  int *int_p;           // Questo punta a un intero
2  char *char_p;        // Questo punta a un carattere
3  // E via dicendo ...
```

Inoltre in `cpp` la dichiarazione di un puntatore ha una sintassi variabile: il `*` può difatti essere posto dopo il tipo di puntatore o prima del nome di questo. A causa di questa caratteristica bisogna tuttavia prestare attenzione a quando si vogliono dichiarare due puntatori nella stessa riga:

```
1  // Qui r corrisponde a un intero, non a un puntatore!
2  int* p, r;
3
```

```

4 // Per due puntatori, occorre usare la seguente sintassi:
5 int *p, *r;

```

2.2 Operazioni con i puntatori

Si possono effettuare operazioni diversi con i puntatori, tra cui le più importanti sono:

2.2.1 NULL

Quando si assegna il valore NULL a un puntatore si indica che questo non punta a nessuna cella di memoria. Si usa spesso in seguito all'istruzione delete.

```

1 int *p = NULL; // p non punta a una cella di memoria

```

2.2.2 Operatore &

L'operatore ampersand '&' si usa in due contesti: per ottenere l'indirizzo di una variabile e per creare un riferimento a questa. Il primo caso risulta utile quando si vuole creare un puntatore ad una certa variabile:

```

1 int n = 10;
2 int *n_p = &n; // ora n_p punta all'indirizzo di n!

```

Mentre il secondo caso risulta più utile quando si vuole creare un alias di una variabile per modificare quest'ultima mediante reference.

```

1 int n = 10;
2 int &alias_n = n; // reference ad n
3 alias_n = 20; // n = 20;

```

La differenza sostanziale tra i due approci sta nella flessibilità: i puntatori possono essere riutilizzati e possono essere annullati, mentre le reference lavorano su una sola variabile definita esattamente alla definizione della reference stessa e non possono essere NULL.

2.2.3 Dereferenziazione (*)

La dereferenziazione si usa in due casi: per dichiarare un puntatore e per accedere all'indirizzo a cui questo sta puntando.

```

1  int value = 42;
2  int *p = &value;    // Dichiarazione puntatore
3  int dereferencedValue = *p; // Accede al valore 42

```

2.2.4 Istruzione new

L'istruzione new alloca dinamicamente memoria (nell'HEAP), restituendo un puntatore all'area di memoria allocata.

```

1  int *p = new int;
2  *p = 42;

```

Qui fare `p = 42` darebbe errore: staremmo assegnando un `int` a un `int*`. Invece in questa maniera assegnamo al contenuto della cella di memoria a cui punta `p` il valore 42.

2.2.5 Istruzione delete

L'operatore delete libera la memoria allocata dinamicamente mediante new.

```

1  int *p = new int;
2  delete p;
3  p = NULL;    // come mai p=NULL?

```

Nell'esempio fornito liberiamo la cella di memoria allocata dinamicamente con `int *p = new int`. Se lasciassimo `*p` senza una reference, questo diventerebbe un dangling pointer, ovvero un puntatore contenente un valore assegnato dal compilatore (randomico). Per evitarlo, annulliamo `*p` con `p = NULL`.

2.3 L'istruzione typedef

L'istruzione typedef permette di definire un nuovo tipo, utilizzabile durante le dichiarazioni di nuovi dati. Questo risulta particolarmente utile quando si lavora con i puntatori in quanto permette di scrivere quanto segue:

```

1  // fuori dal main, idealmente ...
2  typedef int *p_int;    // definisco tipo p_int
3
4  // nel main, ora...
5  p_int p, q;           // equivalente a int *p, *q
6

```

```
7 // ho dichiarato due puntatori in una sola riga!  
8 // Questo permette di evitare confusione
```

3 Strutture dati dinamiche

3.1 Le liste