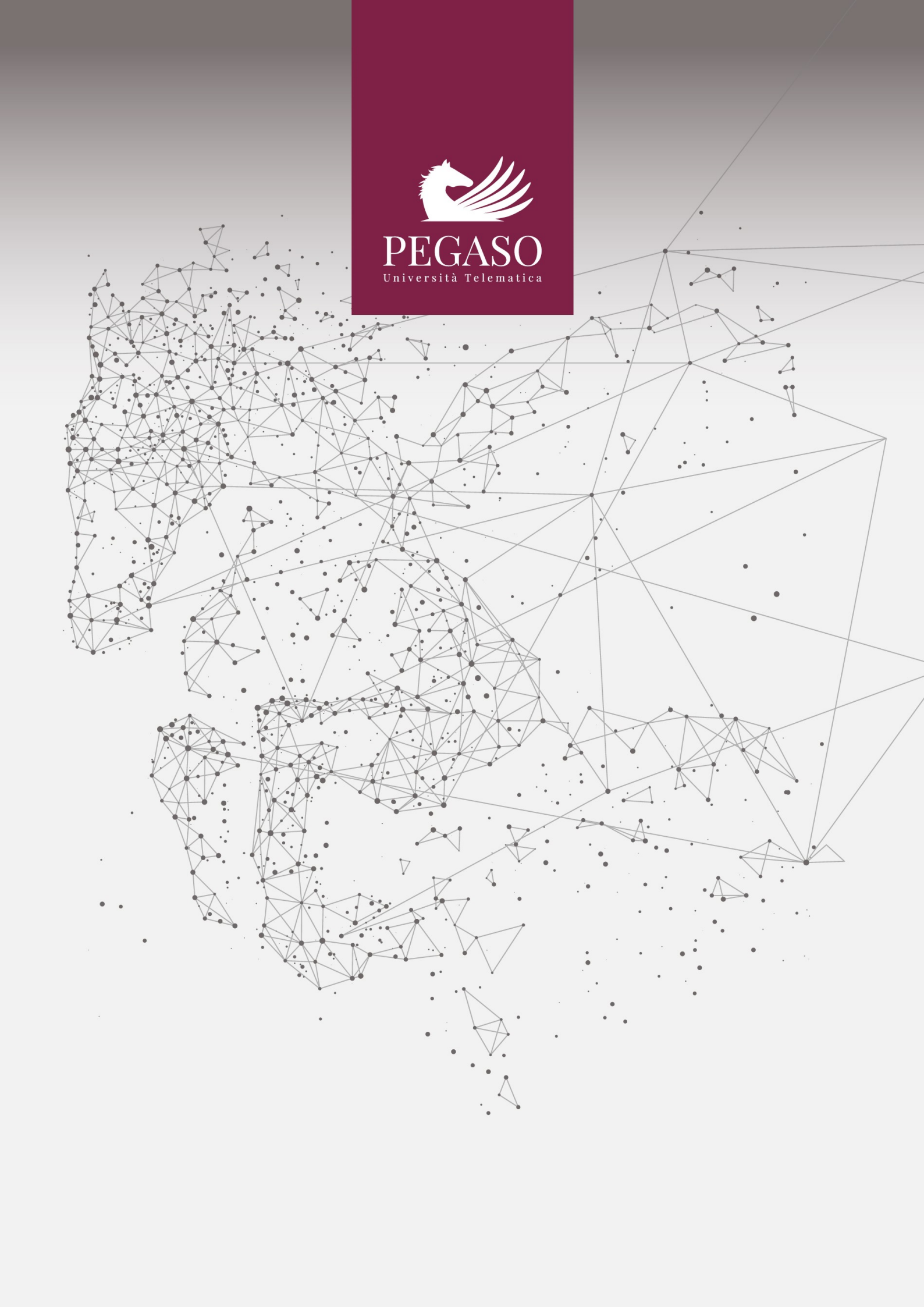
PEGASO
Università Telematica

Indice

1. Introduzione alle strutture dati

Una struttura dati è una rappresentazione organizzata e logica di un insieme di dati in un computer, che permette di accedere, modificare e gestire i dati in modo efficiente.

Le strutture dati sono utilizzate per risolvere problemi specifici, come la gestione di grandi quantità di dati, la ricerca di informazioni specifiche, l'elaborazione di dati e la risoluzione di problemi di ottimizzazione.

È bene illustrare che relazione esiste tra un insieme e una struttura dati:

- un insieme è una raccolta di elementi;
- una struttura dati è una rappresentazione organizzata e logica di un insieme di dati.

Un insieme può essere considerato come una raccolta di elementi senza alcuna organizzazione specifica, ad esempio un insieme di numeri interi, che sono semplicemente una raccolta di numeri senza alcuna organizzazione specifica.

Una struttura dati, invece, fornisce una rappresentazione organizzata e logica dei dati all'interno di un insieme. Ad esempio, una lista è una struttura dati che rappresenta un insieme di elementi in modo ordinato e accessibile tramite indici. Oppure un albero binario è una struttura dati che rappresenta un insieme di elementi in modo gerarchico.

In sintesi, un insieme è una raccolta di elementi mentre una struttura dati è un modo per organizzare e rappresentare questi elementi in modo che possano essere gestiti e utilizzati in modo più efficiente.

I tipi di strutture dati più comuni sono:

- **Array**: una collezione ordinata di elementi, tutti dello stesso tipo, accessibili tramite un indice.
- **Liste**: una collezione ordinata di elementi, di qualsiasi tipo, accessibili tramite un indice. Le liste possono essere singolarmente collegate o a doppia.
- **Stack**: una collezione di elementi che supporta solo due operazioni principali: push (inserimento) e pop (estrazione). Gli elementi vengono inseriti e rimossi dalla cima dello stack.
- **Coda**: una collezione di elementi che supporta solo due operazioni principali: enqueue (inserimento) e dequeue (estrazione). Gli elementi vengono inseriti dalla coda e rimossi dalla testa.
- **Alberi**: una collezione di elementi gerarchicamente organizzati, in cui ogni elemento ha zero o più figli. Ci sono diversi tipi di alberi, come gli alberi binari, gli alberi di ricerca e gli alberi B.

- **Grafi**: una collezione di nodi (o vertici) e archi che collegano i nodi. Ci sono diversi tipi di grafi, come i grafi non orientati, i grafi orientati, i grafi ponderati e i grafi non ponderati.
- **Hash Table**: una struttura dati che utilizza una funzione hash per mappare gli elementi a una posizione specifica in un array. Le tabelle hash sono utilizzate per la ricerca efficiente di elementi in grandi insiemi di dati.

Una tassonomia classica delle strutture dati prevede tre dimensioni ortogonali tra loro:

- Statica o dinamica.
- Compatta o sparsa.
- Basata o non basata sull'ordinamento delle chiavi.

ATTENZIONE: possono esserci ulteriori classificazioni ma in linea generale le dimensioni di analisi sono quelle indicate in precedenza.

Una struttura **dinamica** è una struttura che è pensata per aggiungere o togliere elementi durante l'esecuzione di un algoritmo; un array ad es. è considerato una struttura **statica**. Possiamo dunque dire che le strutture dati statiche sono quelle in cui la dimensione e la configurazione degli elementi non cambiano durante l'esecuzione del programma, mentre le strutture dati dinamiche sono quelle in cui la dimensione e la configurazione degli elementi possono cambiare durante l'esecuzione del programma.

Quando si parla di struttura **compatta** o **sparsa** ci si riferisce alla posizione fisica degli elementi in memoria; tipicamente una struttura dinamica è sparsa, non si possono cioè fare ipotesi sulla posizione fisica degli elementi in memoria; gli array sono invece una struttura compatta: indipendentemente da dove è memorizzato, possiamo assumere che sia fatto in modo tale da avere n posizioni fisiche vicine tra loro.

A volte ci si riferisce a queste anche in termini di strutture dati **lineari** e **non lineari**: le strutture dati lineari sono quelle in cui gli elementi sono organizzati in una sequenza lineare e sono accessibili solo attraverso gli estremi della sequenza (come ad es. gli array, le liste e gli stack); le strutture dati non lineari sono quelle in cui gli elementi non sono organizzati in una sequenza lineare e possono essere accessibili attraverso più punti (ad es. gli alberi e i grafi).

Quando si parla di **ordinamento** ci si riferisce al fatto se gli elementi sono disposti in maniera dipendente dal valore delle chiavi: in tal caso la struttura si basa su un ordinamento, altrimenti no. Una lista o un albero di ricerca sono tipicamente strutture ordinate mentre una tabella hash non lo è.

2. Insieme dinamico

Abbiamo detto in precedenza che un insieme è una raccolta di elementi mentre una struttura dati è un modo per organizzare e rappresentare questi elementi in modo che possano essere gestiti e utilizzati in modo più efficiente.

Un insieme può dunque essere rappresentato mediante una struttura dati (quella più adatta a rappresentare la realtà che si sta modellando).

Gli insiemi che sono di maggior interesse nel nostro ambito di lavoro, sono quelli considerati **dinamici**, cioè insiemi che prevedono la costruzione di algoritmi che consentono di inserire e cancellare degli elementi e/o di verificare l'appartenenza di un elemento all'insieme stesso.

Un insieme dinamico che supporta queste operazioni è detto **dizionario**. Altri algoritmi richiedono operazioni più complicate: inserire un elemento in un insieme o estrarre l'elemento più piccolo da un insieme… il modo migliore di implementare un insieme dinamico dipende dalle operazioni che devono essere supportate.

In una tipica implementazione di un insieme dinamico, ogni elemento è rappresentato da un oggetto i cui campi possono essere esaminati e manipolati se c'è un puntatore all'oggetto: per alcuni tipi di insiemi dinamici si suppone che uno dei campi dell'oggetto sia un campo **chiave di identificazione**.

Se le chiavi sono tutte diverse, possiamo pensare all'insieme dinamico come a un insieme di valori chiave. L'oggetto può contenere dati satelliti, che vengono spostati in altri campi dell'oggetto, senza essere utilizzati in altro modo dall'implementazione dell'insieme.

L'oggetto può anche includere campi che vengono manipolati dalle operazioni svolte sull'insieme; questi campi possono contenere dati o puntatori ad altri oggetti dell'insieme.

Le operazioni su un insieme dinamico possono essere raggruppate in due categorie:

- **interrogazioni** (query): restituiscono semplicemente informazioni sull'insieme;
- operazioni di **modifica**: cambiano l'insieme.

Elenchiamo di seguito una serie di operazioni tipiche (un'applicazione reale di solito richiede l'implementazione di un numero limitato di queste operazioni):

- **SEARCH(S, k)**: una query che, dato un insieme S e un valore chiave k, restituisce un puntatore x a un elemento di S tale che chiave[x] = k oppure NIL se un elemento così non appartiene a S.
- **MINIMUM(S)**: una query su un insieme totalmente ordinato S che restituisce un puntatore all'elemento di S con la chiave più piccola.

- **MAXIMUM(S)**: una query su un insieme totalmente ordinato S che restituisce un puntatore all'elemento di S con la chiave più grande.
- **SUCCESSOR(S, x)**: una query che, dato un elemento x la cui chiave appartiene a un insieme totalmente ordinato S, restituisce un puntatore al prossimo elemento più grande di S oppure NIL se x è l'elemento massimo.
- **PREDECESSOR(S, x)**: una query che, dato un elemento x la cui chiave appartiene a un insieme totalmente ordinato S, restituisce un puntatore al prossimo elemento più piccolo di S oppure NIL se x è l'elemento minimo
- **INSERT(S, x)**: un'operazione di modifica che inserisce nell'insieme S l'elemento puntato da x
- **DELETE(S, x)**: un'operazione di modifica che, dato un puntatore x a un elemento dell'insieme S, rimuove x da S (notate che questa operazione usa un puntatore a un elemento x, non un valore chiave).

In questo contesto ha senso parlare di strutture dati concrete ed astratte. Una struttura dati **astratta** è un modello generale di una struttura dati che descrive le operazioni che possono essere effettuate su di essa e le relazioni tra i suoi elementi, senza specificare come queste operazioni vengono effettivamente implementate. Le strutture dati astratte sono utilizzate per descrivere l'interfaccia di una struttura dati e per garantire che tutte le implementazioni di una determinata struttura dati soddisfino determinati requisiti.

Una struttura dati **concreta** è invece una implementazione specifica di una determinata struttura dati astratta. Ad esempio, un array è una struttura dati concreta, mentre una lista è una struttura dati astratta. Una lista è un concetto generale che descrive un insieme ordinato di elementi, mentre un array è una implementazione specifica di una lista che utilizza una matrice di elementi contigui in memoria.

3. Puntatore

Abbiamo parlato in precedenza di "puntatore". Il puntatore è un concetto importante in relazione alle strutture dati in quanto consente di creare strutture dati dinamiche.

In alcune strutture dati, come le liste dinamiche e gli alberi, gli elementi sono memorizzati in modo dinamico, cioè vengono allocati e deallocati durante l'esecuzione del programma. In queste strutture dati, i puntatori vengono utilizzati per collegare gli elementi tra loro e consentire l'accesso ai singoli elementi.

Ad esempio, in una lista dinamica, ogni elemento contiene un puntatore che punta al successivo elemento nella lista. Questo consente di creare una struttura dati in cui gli elementi possono essere inseriti e rimossi dinamicamente, senza la necessità di riallocare la memoria per l'intera struttura dati.

Un puntatore non è altro che un tipo particolare di variabile adatta a memorizzare l'indirizzo di una locazione di memoria, ovvero adatta a memorizzare l'indirizzo di un'altra variabile, che prenderà il nome di variabile puntata.

Di fatto, sappiamo che anche gli array sono dei puntatori: quando si dichiara un array, in realtà si dichiara un puntatore, con alcune caratteristiche in più:

1. la dichiarazione di un puntatore comporta allocazione di memoria per una variabile puntatore, ma non per la variabile puntata;
2. la dichiarazione di un array comporta allocazione di memoria non solo per una variabile puntatore (il nome dell'array), ma anche per l'area puntata, di cui viene predefinita la lunghezza; inoltre, il puntatore viene dichiarato `const` e inizializzato con l'indirizzo dell'area puntata (cioè del primo elemento dell'array).

Riprendiamo in esame il tema dei puntatori in C++ attraverso il seguente schema:

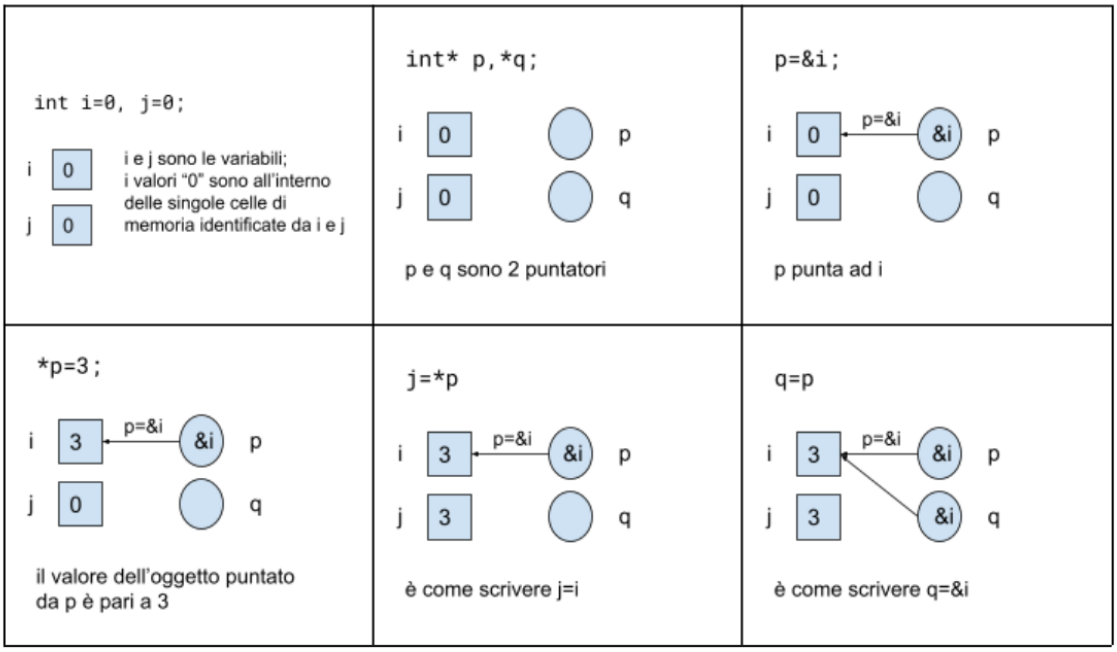

Figura 1 *- Puntatori in C++*

La dichiarazione, con conseguente definizione, di una variabile puntatore consiste nell'aggiungere il simbolo di asterisco * tra il tipo della variabile e il suo identificatore: $< tipo > * < identificatore >$.

Alla definizione viene subito allocato spazio in memoria anche per questo tipo di variabile, la quale conterrà quindi un valore casuale, ovvero un indirizzo casuale:

- per ottenere l'indirizzo di un oggetto si usa l'operatore &
- per accedere all'oggetto riferito da un puntatore si usa l'operatore *

Per evitare problemi, è buona regola annullare (o inizializzare) i puntatori nel momento in cui vengono dichiarati; il C++ supporta l'allocazione dinamica e la deallocazione degli oggetti usando gli operatori new e delete.

```
int* p = NULL;
p = new int;
*p = 123;
delete p;
int* q = NULL;
q = new int[100];
for (int i=0;i<100;i++) {
      q[i]=i;
}
[…]
delete [] q;
```

Cosa accade in linguaggi come il Python?

In Python, non esistono i puntatori nel senso tradizionale della parola, come sono presenti in C o in C++: in Python, tutte le variabili sono puntatori impliciti che puntano a oggetti presenti in memoria.

In Python, quando si assegna un valore a una variabile, si sta creando un "riferimento" all'oggetto in memoria, piuttosto che copiare il valore in un'altra posizione di memoria.

Per questo motivo, non esiste la necessità di utilizzare i puntatori espliciti per accedere agli oggetti in memoria.

Inoltre, Python gestisce automaticamente la memoria tramite un meccanismo chiamato Garbage Collection, che si occupa di liberare automaticamente la memoria degli oggetti non più utilizzati.

Alcuni esempi di applicazione del puntatore in strutture dati possono essere il seguente:

- **Liste dinamiche**: in una lista dinamica, ogni elemento contiene un puntatore che punta al successivo elemento nella lista. Questo consente di creare una struttura dati in cui gli elementi possono essere inseriti e rimossi dinamicamente, senza la necessità di riallocare la memoria per l'intera struttura dati.
- **Alberi**: in un albero, ogni nodo contiene puntatori ai nodi figli. Questo consente di creare una struttura dati gerarchica in cui ogni nodo può avere un numero qualsiasi di figli.
- **Grafi**: in un grafo, ogni nodo contiene una lista di puntatori ai nodi adiacenti. Questo consente di creare una struttura dati in cui ogni nodo può avere un numero qualsiasi di archi uscenti.

Alcuni esempi di utilizzo del puntatore:

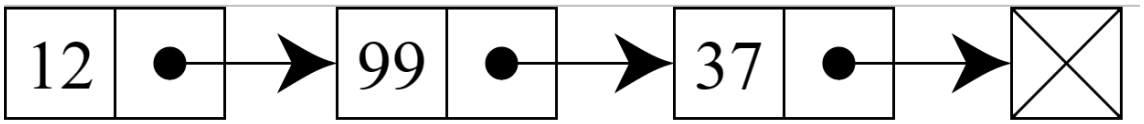

Figura 2 *- Lista*

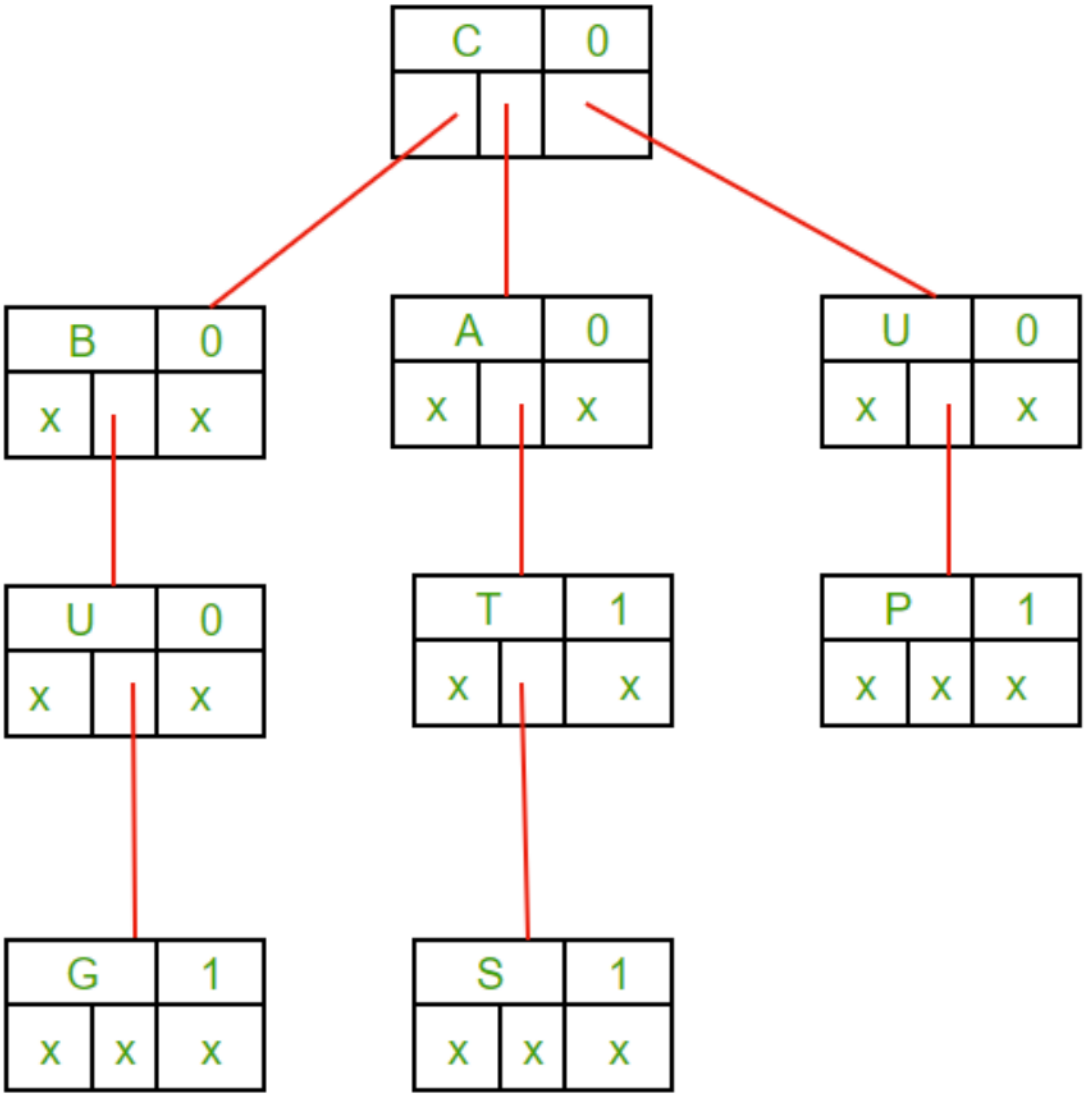

Figura 3 *- Albero*

Bibliografia

- Alan Bertossi, Alberto Motresor: Algoritmi e strutture di dati, Città Studi Edizioni, terza edizione.
- C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione.
- Crescenzi, Gambosi, Grossi: Strutture di Dati e Algoritmi, Pearson/Addison-Wesley.
- Sedgewick: Algoritmi in C, Pearson, 2015.
- Cormen Leiserson Rivest Stein-Introduzione Agli Algoritmi E Strutture Dati-Prima Edizione.