Domande e Risposte Fondamenti di Programmazione

- **Cos'è la grammatica di un linguaggio?**

La grammatica di un linguaggio è l'insieme di regole che rappresentano il modo in cui bisogna usarlo. E' suddivisa in sintassi e semantica. La sintassi spiega le regole secondo cui vanno scritte le istruzioni mentre la semantica ne indica il loro significato.

- **Qual è la differenza tra parametri formali e attuali?**

I parametri formali sono contenitori che vengono inizializzati con i parametri attuali passati alla funzione. Ad eccezione dei puntatori o al passaggio degli indirizzi, i parametri formali prendono il valore dei parametri attuali e non possono modificare i parametri attuali. Alla fine del blocco vengono distrutti. I parametri attuali sono invece variabili che vengono dichiarate fuori dal blocco delle funzioni, ad esempio nel main. Vengono distrutti alla fine del programma.

- **Qual è la differenza tra passaggio per valore e per riferimento?**

Gli argomenti attuali possono essere passati alle funzioni in due modi: per valore o per riferimento. Nel primo caso vengono copiati esclusivamente i valori delle variabili negli argomenti formali e la modifica di un argomento formale non si ripercuote sull'argomento attuale a lui assegnato. Nel secondo caso, nell'argomento formale viene copiato l'indirizzo di memoria della variabile a lui assegnata, ed avremo un altro nome per quella variabile da usare all'interno della nostra funzione. Quest'ultimo tipo di passaggio si ripercuote sull'argomento attuale in caso di modifica dell'argomento formale.

- **Differenza tra incremento Prefisso e Postfisso?**

La differenza tra incremento prefisso e postfisso sta nel momento dell'incremento della variabile e dal conseguente valore restituito.
L'incremento prefisso incrementa prima la variabile da utilizzare e poi viene utilizzata.
Esempio: int a=5; ++a -> incrementa a di uno portandola a 6 e poi restituisce a(6).
Nel postfisso la variabile viene prima utilizzata e poi viene incrementata.
Esempio: int a=5; a++ -> restituisce a(5) e poi incrementa a di uno portandola a 6.
L'incremento prefisso restituisce un left-value, mentre il postfisso un right-value; di conseguenza è corretto effettuare un doppio incremento prefisso ma non un doppio incremento postfisso, proprio perché utilizzando il postfisso una volta viene restituito un r value su cui non può essere nuovamente effettuato un incremento.

- **Cosa si intende per left/right values e qual è la differenza tra i due?**

Un left-value è un "valore di sinistra", nel senso che si trova a sinistra del simbolo di assegnamento **=** e rappresenta quindi una zona di memoria dove è possibile, in quella circostanza, effettuare una scrittura (writable memory location). Un right-value, al contrario, è un "valore di destra", che si trova quindi a destra dell'operatore di assegnamento. Rappresenta una zona di memoria che in quella circostanza è di sola lettura (read-only) o si tratta di un valore che non ha un indirizzo (ad esempio un numero) o ancora di una costante. Tutti i left-values sono anche right-values, perchè su una zona di memoria destinata alla scrittura è anche possibile eseguire una lettura, mentre non vale il viceversa.

- **Cosa e quali sono gli operatori logici?**

Gli operatori logici sono degli operatori che vengono utilizzati per effettuare dei controlli sulle variabili. I principali sono 3:
AND: è un operatore logico binario che serve a verificare se due o più condizioni siano vere contemporaneamente. Si indica con la coppia di caratteri speciali "&&".
OR: è un operatore logico binario che serve a verificare se almeno una delle condizioni sia vera. Si indica con la coppia di caratteri speciali "||".
NOT: è un operatore logico unario, e serve a controllare che una condizione non si verifichi.

- **Cos'è l'overloading?**

L'Overloading (sovrapposizione) nel C++ si ha su due entità diverse:
1. **Overloading di funzioni**: due funzioni con lo stesso identificatore possono esistere se e solo se è diverso il numero totale degli argomenti formali o almeno uno di essi è di tipo differente. Sintassi:
function_name(type1 type_name);
function_name(type2 type_name);
function_name(type1 type_name,type2 type_name);

2. **Overloading degli operatori**: essendo le classi dei tipi derivati alcune operazioni non sono definite su di esse, quindi, invece di definire una funzione che attua quel determinato procedimento, il linguaggio ci permette di ridefinire l'operatore stesso. In questo modo possiamo estendere ai tipi classe una proprietà già valida per i tipi fondamentali. Sintassi:
//file.h
public: class_name operator<operator> (class_name obj_name);
//file.cpp
class_name class_name::operator<operator> (class_name obj_name)

- **Cosa sono i puntatori?**

I puntatori sono degli oggetti che hanno per valore l'indirizzo di memoria di un altro oggetto.

- **Come si creano?**

I puntatori prendono il tipo in base al tipo dell'oggetto puntato: si inserisce prima il tipo seguito dal simbolo '*' asterisco (in questo modo si viene a creare un tipo derivato) ed alla fine si sceglie l'identificatore della nostra variabile puntatore.

- **Come si usano?**

Come prima cosa un puntatore deve essere inizializzato a NULL o con l'indirizzo di un oggetto, altrimenti avrà un indirizzo di memoria casuale. Si possono effettuare numerose azioni tramite i puntatori, con la dereferenziazione ad esempio si può accedere direttamente alla variabile puntata , in questo modo si può modificare la variabile. Si possono utilizzare anche per mantenere l'indirizzo del primo elemento di un vettore o di una matrice, quindi si possono utilizzare per la gestione di vettori e matrici o anche di liste.

- **Quanto occupa in memoria un puntatore?**

Come gli interi, i puntatori occupano sempre 4 byte.

- **Possibili errori con puntatori non inizializzati a NULL?**

Come abbiamo detto poco fa, un puntatore non inizializzato punta ad un indirizzo di memoria random.

- **Cosa rappresenta il nome del puntatore?**

Il nome del puntatore rappresenta l'identificativo che si riferisce all'indirizzo della cella di memoria occupata dalla variabile che ad esso è stata assegnata.

- **Cos'è l'aritmetica dei puntatori?**

Per aritmetica dei puntatori si intende tutta una serie di operazioni che si possono applicare sui puntatori. Sapendo che un puntatore ha in memoria l'indirizzo di un oggetto, se questo oggetto è ad esempio di tipo vettore, e l'indirizzo del primo elemento è p, con l'espressione p+1 ritorna l'indirizzo di un oggetto di tipo vettore ma nell'area di memoria immediatamente successiva. Grazie ad un puntatore e l'aritmetica dei puntatori possiamo scorrere un intero vettore.
Oppure, se noi memorizziamo l'indirizzo del primo elemento di una matrice all'interno del nostro puntatore, con due cicli for uno per l'indice di riga e l'altro per l'indice di colonna, attraverso l'espressione (p+i*tot colonne +j) possiamo scorrere l'intera matrice.
Questo tipo di matrice è detta matrice linearizzata (o lineare?).

- **Qual è la differenza tra puntatori a costanti (area di memoria assegnata) e puntatori costanti(indirizzo puntatore)?**

Ci sono due modi per utilizzare la parola chiave const con i puntatori. Nel primo caso si inserisce prima del tipo del puntatore e sta a indicare che con quel puntatore non possiamo modificare l'oggetto da lui puntato, quindi l'oggetto può essere anche una costante.
Nel secondo caso const va inserita tra l'asterisco e l'identificatore, in questo modo sarà proprio il puntatore un oggetto costante (come tutte le costanti del linguaggio) che deve essere inizializzato e, una volta fatto, l'indirizzo a lui assegnato non potrà più essere cambiato.

- **Cos'è e come si gestisce la memoria dinamica?**

La memoria dinamica (heap) è un particolare tipo di memoria in cui gli oggetti sono allocati in modo sparso, a differenza dello stack dove sono allocati in maniera sequenziale.
La memoria dinamica viene utilizzata quando non si è a conoscenza della quantità di memoria che un oggetto necessita a tempo di compilazione ma solamente a tempo di esecuzione. Per allocare oggetti in questa memoria si utilizza la parola chiave "new" seguita dal tipo dell'oggetto che si vuole allocare. L'operatore new restituisce l'indirizzo del blocco di memoria che va salvato in un puntatore creato nello stack, per non perdere inesorabilmente l'oggetto appena creato. Se a questo puntatore viene assegnato un indirizzo di memoria diverso, l'oggetto in memoria dinamica precedentemente a lui assegnato viene perso definitivamente. L'oggetto continuerà ad esistere nella memoria dinamica fin quando non viene deallocato attraverso l'operatore "delete", oppure automaticamente alla fine del programma.

- **Cosa sono le classi?**

Una classe è un insieme di funzioni o variabili che trattano lo stesso argomento e che sfruttano gli stessi elementi per effettuare azioni diverse. Le classi sono un tipo derivato e differiscono dalle strutture perché al loro interno i membri possono essere suddivisi in parte privata e parte pubblica; la parte privata può essere raggiunta dalle funzioni membro della classe o dalle funzioni friend.

- **Cos'è una lista di inizializzazione?**

Una lista di inizializzazione è la prima parte di una funzione di un costruttore di classe. La funzione del costruttore è divisa in due parti: lista di inizializzazione e corpo del costruttore. La lista di inizializzazione è necessaria se nella classe sono stati dichiarati dei membri dato costanti. Dopo aver chiuso le parentesi tonde degli argomenti del costruttore si devono inserire i ':' due punti, subito dopo richiamare con il loro identificatore i membri dato costanti ed inizializzarli con un valore o con argomento formale della funzione costruttore. L'inizializzazione dei membri dato costanti sono separati da una virgola.

- **Cos'è il costruttore di default? E quello di copia?**

Il costruttore di default viene invocato automaticamente dal programma quando si ha la necessità di creare una nuova istanza per quella classe. Il costruttore di copia è un particolare tipo di funzione che assume lo stesso nome della classe di appartenenza (come il costruttore di default) ed effettua una copia membro a membro di tutte le variabili già inizializzate dal costruttore, così da creare un oggetto identico a quello precedentemente creato. Tuttavia è spesso necessario che il programmatore ridefinisca il costruttore di default e/o quello di copia poiché i dati da inizializzare/copiare possono essere troppo complessi per i costruttori predefiniti.

- **Cos'è il puntatore this? A cosa serve? A cosa punta?**

Il puntatore this è un particolare tipo di puntatore che viene dichiarato implicitamente nella classe. Prende l'indirizzo dell'istanza della classe che è stata creata e può essere utilizzato solo dalle funzioni membro. Gli usi sono molteplici: ad esempio, quando si ha la necessità, all'interno di una funzione, di restituire proprio l'oggetto classe o una delle variabili del campo dati della classe; o ancora per fare un controllo dell'aliasing con un'altra istanza della medesima classe.

- **Cos'è il Distruttore?**

Il Distruttore è quella particolare funzione che viene chiamata automaticamente al termine del programma per deallocare tutta la memoria allocata durante l'esecuzione del programma dal quella specifica classe.

- **Cosa sono le funzioni friend?**

Le funzioni friend sono particolari tipi di funzione che vengono dichiarate all'interno di una classe come friend ("amiche") e possono accedere a tutti i membri privati di una classe, non facendo effettivamente parte della classe stessa. Queste funzioni vengono poi definite al di fuori del file della classe.

- **Cosa sono le stringhe?**

Il tipo stringa non esiste in C++. Per far fronte a questa mancanza si è deciso di interpretare le stringhe come array di caratteri: ogni elemento dell'array contiene un carattere.
Ogni stringa termina con un carattere ben preciso, detto "di fine stringa", ovvero "\0", che pone lo stream in una situazione di "errore", settando il flag di lettura del carattere di fine stringa a 1, interrompendo così le operazioni che si stanno effettuando su di essa.
Includendo la libreria <cstring> possiamo effettuare svariate operazioni sulle stringhe, come: la copia di una stringa in una variabile, il confronto tra due stringhe, il controllo della

lunghezza della stringa eccetera. Inoltre, la libreria <string> permette di bypassare l'assenza del tipo stringa creandone uno derivato.

- **Cosa sono le classi di memorizzazione?**

Le classi di memorizzazione (storage class) rappresentano una particolare proprietà che definisce il tempo di vita e la visibilità di un oggetto all'interno del programma.
Le classi di memorizzazione sono 3: automatica, statica ed esterna.
1. **Automatica**: E' quella usuale per le variabili locali, che vengono dichiarate all'interno di un blocco quando viene incontrata la loro definizione e distrutti alla fine del blocco. Se non vengono inizializzati il loro valore è indefinito. Esempio: Oggetti formali di una funzione sono pensati come automatici.
2. **Statica**: Una variabile locale statica è una variabile che vede associato uno spazio per tutto il tempo che il programma è in esecuzione. La zona di memoria utilizzata è quella in cui si trovano anche le variabili globali. Essa conserva il proprio valore (anche se inaccessibile) tra una chiamata e l'altra della funzione in cui è definita. Vengono creati all'inizio dell'esecuzione del programma principale e vengono distrutti alla fine del programma stesso. Se non sono stati inizializzati il loro valore è 0. Questa classe di memorizzazione può far comodo nel momento in cui si vuole limitare la visibilità di una variabile tra un file ed un altro
3. **Dinamica**: Sono gli oggetti che vengono creati quando ad un certo punto del programma si incontra l'operatore NEW.
Vengono distrutti o quando si incontra l'operatore DELETE o alla fine del programma stesso. Se non inizializzati hanno un valore indefinito.

- **Come si effettua la lettura/scrittura da/su di un file ?**

Per leggere o scrivere su un file si utilizza la libreria <fstream> che contiene le funzioni di lettura e scrittura sui file. Si crea quindi un oggetto fstream con un proprio identificatore, e si utilizzano le funzioni presenti nella libreria per aprire/chiudere un file.
Esempio:

fstream ff; ff.open("nomefile.txt", ios::out)

Questo crea un oggetto fstream chiamato ff su cui si invoca la funzione "open" che crea un file chiamato "nomefile.txt" in scrittura ("ios::out").
Il secondo parametro della funzione open può essere ios::out (scrittura), ios::in (lettura) oppure ios::out|ios::app (scrittura alla fine del file).
Successivamente, utilizzando il nome assegnato allo stream di lettura collegato al nostro file ed una variabile dello stesso tipo di quelle che dovrà essere letto dal file, possiamo effettuare la lettura.
Esempio:

int a =0; nome>>a;

La scrittura, invece, si effettua creando uno stream con la seguente sintassi:

-ofstream nome ("nomefile.txt");

ed in seguito utilizzando il nome del nostro stream di scrittura per scrivere cosa si vuole su file.
Esempio.

char s= 'S'; nome<<s;

- **Cosa sono e quali sono gli algoritmi di ordinamento dei vettori?**

Gli algoritmi di ordinamento di vettori sono dei procedimenti che spostano gli elementi di un vettori ordinandoli secondo un certo criterio.

I principali sono due:
-Bubble Sort: I valori più grandi "Fluttuano" verso la parte finale dell'array quindi i valori passano da più posizioni prima di raggiungere quella finale.
-Selection Sort: Ad ogni iterazione viene spostato un solo elemento (quello più piccolo) e lo mette all'inizio quindi in questo caso i valori arrivano direttamente alla loro posizione finale.

- **Cos'è il Define e a cosa serve?**

In molti casi è utile assegnare a degli identificatori dei valori che restino costanti per tutto il programma e che non possono essere cambiati nemmeno per errore.
In C++ questo si può ottenere, oltre che con il modificatore const, con la direttiva DEFINE. Per la direttiva del preprocessore deve essere anteposto il simbolo del cancelletto (#), dopo la parola chiave DEFINE e successivamente l'identificatore con un determinato valore o operazione. Ogni qual volta all'interno del programma si incontra l'identificatore del DEFINE, esso viene sostituito con il valore a lui associato.
L'uso del DEFINE va oltre questi semplici collegamenti di identificatori/valori ma può costituire vere e proprie azione che devono essere ripetute all'interno di un programma, le cosiddette MACRO.

- **Cosa sono le macro?**

Una Macro è un'operazione "preimpostata" che solitamente verrà usata più volte in un programma. Se abbiamo necessità di uno scambio di variabili possiamo definire la seguente macro:

#define scambia (x,y,temp) (temp)=(x); (x)=(y); (y)=(temp);

a questo punto ogni qualvolta che viene incontrato la macro scambia viene sostituita alla serie di operazioni da compiere con quelle variabili.

- **Cos'è il cortocircuito AND e OR logico?**

La valutazione a corto circuito è un meccanismo relativo agli operatori booleani binari, per cui il secondo operando viene valutato se e solo se il valore del primo operando non è sufficiente da solo a determinare il risultato dell'operatore.
Il risultato dell'operatore logico AND è necessariamente falso se il primo operando è falso.
Il risultato dell'operatore logico OR è necessariamente vero se il primo operando è vero.
Quindi il risultato si può ottenere analizzando un solo operando (il primo).
//esempio di cortocircuito

- **Cosa sono le liste?**

Una lista è un tipo derivato composto da un struttura contenente uno o più membri informazione e uno o più membri puntatori. Le liste vengono utilizzate quando si ha la necessità di memorizzare un insieme di elementi ma senza saperne il numero a tempo di compilazione. In casi come questo, creare un array è una soluzione errata per vari motivi: spreco di memoria allocando un array molto grande o, esatto opposto, memoria insufficiente all'interno dell'array. Il concetto base delle liste è che dal primo elemento della lista è possibile passare a tutti gli elementi successivi, perché ai puntatori della struttura di tipo lista vengono assegnati gli indirizzi degli elementi di tipo lista successivi. E' possibile effettuare numerose operazioni sulle liste, le principali: stampa a video, inserimento/eliminazione in coda/testa/in base ad una condizione

- **Cosa si intende per programmazione a moduli?**

Per programmazione a moduli si intende un tipo di programmazione in cui il programma viene suddiviso in più file. Più semplicemente:

Modulo **Server:** che offre dei servizi

Modulo **Client:** che sfrutta i servizi messi a disposizione del modulo server.

Questo metodo viene utilizzato per semplificare lo sviluppo, il test e la manutenzione del programma.

- **Cosa sono le matrici?**

Le matrici o array multidimensionali sono una collezione di dati dello stesso tipo memorizzati sotto forma di matrice al quale si può accedere attraverso degli indici (uno rappresenta la riga l'altro la colonna) che individuano il valore all'interno di quella determinata posizione. Una matrice può essere vista anche come vettore linearizzato, in questo caso l'accesso ai vari membri della matrice avviene tramite l'espressione (riga corrente * numero colonne totale + colonna corrente).

- **Cosa bisogna fare per impedire che un array venga modificato?**

Basta aggiungere la parola chiave CONST prima del tipo dell'array.

- **Come si rappresentano i numeri reali?**

I numeri reali nel linguaggio c++ vengono rappresentati con la parola chiave DOUBLE e occupano 32 bit in memoria. Possiamo avere delle varianti come il FLOAT che rappresenta sempre i reali ma su 16 bit o ancora il LONG DOUBLE su 64 bit.
Sui reali sono definite tutte le operazioni aritmetiche e di confronto.

- **Cos'è il tipo enumerato?**

L'enumeratore è un tipo derivato che viene definito dal programmatore, quindi si viene a creare un nuovo tipo. Tutte le variabili con tipo enumeratore possono assumere solo i valori dichiarati dal tipo enumeratore.
Esempio:

enum giorni { lunedi, martedi, giovedi,ecc...};

Le costanti dichiarate all'interno del enumeratore giorni vengono viste come costanti numeriche a partire da 0 per lunedì e 6 per domenica. Creando una variabile di tipo giorni gli si può assegnare solo le costanti dichiarate all'interno dell'enumeratore giorni. Il tipo Enumeratore viene utilizzato più che altro per le operazione di confronto.

- **Cos’è una funzione ricorsiva?**

Con ricorsione si intende la possibilità data ad una funzione di invocare se stessa oltre che un’altra funzione. Per scrivere correttamente una funzione ricorsiva sono necessarie due componenti: un caso base (o più d'uno), che rappresenta la condizione per la quale la ricorsione termina, e un passo ricorsivo (o più d'uno). Il caso base è regolato da un argomento che viene modificato ad ogni passo della ricorsione fino a raggiungere una condizione tale da restituire un determinato valore e non avviare un’ulteriore ricorsione. Così facendo, vengono terminate tutte le istanze precedentemente avviate dalla ricorsione, ma in senso opposto, fino a ritornare al chiamante.

- **Cos'è il polimorfismo ?**

Il polimorfismo indica la possibilità di definire metodi e proprietà con lo stesso nome.
Il polimorfismo può essere distinto in due casi:
Polimorfismo del contesto della **programmazione ad oggetti:**
Si riferisce al fatto che in una classe derivata possa ridefinire un metodo della classe base con lo stesso nome. Se per esempio è una funzione , questa deve essere preceduta dalla parola chiave VIRTUAL che sta a significare che è possibile avere un'altra definizione per la stessa funzione all'interno della classe derivata. La funzione all'interno della classe derivata deve essere preceduta dalla parola chiave OVERRIDE che sta a significare una sovrascrittura di un metodo che ha lo stesso nome del metodo della classe Base.
Polimorfismo nel contesto della **programmazione generica:** si riferisce al fatto che si può stilare un programma senza un tipo specifico. Questo nel C++ viene effettuato attraverso i Templates.

- **Operatori bit a bit.**

1)AND bit a bit: (&) effettua un AND bit a bit tra due operandi , cioè, setta a 1 il bit quando i bit di quella posizione sono entrambi 1 e setta a 0 in tutti gli altri casi.
2)OR bit a bit: (|) effettua un OR bit a bit tra due operandi, cioè, setta a 1 il bit quando almeno uno dei bit in quella posizione è uguale ad 1 a 0 negli altri casi.
3)OR esclusivo bit a bit:(^)(caret) effettua un OR esclusivo bit a bit tra due operandi, cioè, setta ad 1 il bit quando i bit in quella posizione sono diversi, in tutti gli altri casi setta il bit a 0.

- **Che utilizzo ha la parola chiave *Extern*?**

Effettuando una dichiarazione con la parola chiave *extern* è possibile utilizzare un oggetto o una funzione definiti in un'altra unità di compilazione. Di default gli identificatori a livello di blocco hanno collegamento interno, quelli a livello di file hanno collegamento esterno (ad eccezione degli identificatori costanti che hanno sempre collegamento interno. *Extern* permette inoltre di condividere variabili e funzioni tra linguaggi diversi (ad esempio tra C e C++). I rispettivi compilatori devono però seguire determinati standard di compatibilità per permettere la "comunicazione" tra diverse unità di compilazioni.

- **Che utilizzo ha la parola chiave *Static*?**

Modifica il tipo di collegamento da esterno ad interno. Se utilizzata all'interno di un blocco definisce un oggetto statico (di classe statica e non più di classe automatica, come di default) che mantiene il valore anche dopo la fine del blocco (non viene di fatto distrutto) (chiamata anche classe di memorizzazione (storage class)). Static può essere usato allo stesso modo anche nelle classi per rendere una variabile "globale" e rimanere inalterata e condivisa rispetto alla creazione delle istanze di classe.

Gestione di una Lista

Inserimento in testa: il nuovo elemento avrà nel campo puntatore l'indirizzo della lista e quest'ultimo sarà quello del nuovo elemento:

```
elemento nuovoElemento = new elemento;
nuovoElemento->puntatore = lista;
lista = nuovoElemento;
```

Inserimento in coda: il nuovo elemento avrà nel campo puntatore il valore NULL e l'elemento precedente avrà nel proprio campo puntatore l'indirizzo del nuovo elemento:

```
elemento nuovoElemento = new elemento;
nuovoElemento->puntatore = NULL;
elementoPrecedente->puntatore = nuovoElemento;
```

Estrazione in testa: una variabile (preferibilmente passata per riferimento alla funzione oppure restituita come valore di ritorno) prende il campo informazione del primo elemento, un elemento temporaneo creato sul momento prenderà l'indirizzo della lista (che corrisponde a quello del primo elemento) che verrà poi modificato con l'elemento da essa puntato (corrispondente al secondo elemento). Infine verrà eliminato il primo elemento agendo sul puntatore temporaneo:

```
variabile = primoElemento->informazione;
elemento *q = lista;
lista = lista->puntatore;
delete q;
```

Estrazione in coda: una variabile (come sopra) prende il campo informazione dell'ultimo elemento della lista mentre un puntatore temporaneo prende il suo indirizzo di memoria. Il campo puntatore dell'elemento precedente sarà uguale a NULL, a cui seguirà una delete sul puntatore temporaneo che cancellerà dallo heap l'ultimo elemento.

```
variabile = ultimoElemento->informazione;
elemento *q = ultimoElemento;
penultimoElemento->puntatore = NULL
delete q;
```

Inserimento in una lista ordinata: si scorre la lista con due puntatori ausiliari fino a che il campo informazione del secondo puntatore non soddisferà la condizione di partenza. A questo punto, il nuovo elemento avrà nel campo puntatore l'indirizzo dell'elemento appena successivo (secondo puntatore) mentre l'elemento precedente (primo puntatore) avrà nel proprio campo puntatore l'indirizzo del nuovo elemento:

```
elemento nuovoElemento = new elemento;
elementoPrecedente->puntatore = nuovoElemento;
nuovoElemento->puntatore = elementoSuccessivo;
```

Estrazione di un elemento in una lista ordinata: si scorre la lista con due puntatori ausiliari fino a che il campo informazione del secondo puntatore non soddisferà la condizione di partenza. A questo punto, una variabile (come sopra) prenderà il campo informazione dell'elemento da estrarre (secondo puntatore) mentre l'elemento precedente avrà nel suo campo puntatore l'indirizzo dell'elemento successivo a cui si dovrà collegare, ottenuto dal campo puntatore dell'elemento da estrarre. Si procede quindi con una delete su quest'ultimo:

```
elemento *primoPuntatore, *secondoPuntatore;
```

```
for(secondoPuntatore = lista; secondoPuntatore->informazione = condizione;
    secondoPuntatore = secondoPuntatore->puntatore)
primoPuntatore = secondoPuntatore;

//primoPuntatore = elementoPrecedente, secondoPuntatore = elementoDaEstrarre

variabile = secondoPuntatore->informazione;
primoPuntatore->puntatore = secondoPuntatore->puntatore;
delete secondoPuntatore;
```