Programmazione II

Sommario

Programmazione II

Programmazione II

Union

La union è una struttura in cui le stesse locazioni di memoria vengono utilizzate per contenere variabili di diverso tipo e quindi conviene utilizzarle allocando memoria uguale per i diversi tipi; es: int A, short B[2], char C[4], così facendo ho allocato 32 bit di spazio condiviso tra le tre variabili.

Programmazione II

Algoritmo divisioni e moltiplicazioni successive

Tramite l'algoritmo delle divisioni successive si può convertire la parte intera di un numero decimale in un numero con base diversa, mentre tramite l'algoritmo delle moltiplicazioni successive si può convertire la parte frazionaria. Se si mettono insieme questi due algoritmi si può convertire un qualsiasi numero razionale in una base diversa da quella decimale.

Programmazione II

Rappresentazione dei numeri

I numeri si rappresentano in memoria in tre modi diversi in base al tipo di dato rappresentato:

- Complemento a 2: per ottenere tale rappresentazione si complementano tutti i bit di partenza, e successivamente si aggiunge 1 al numero binario ottenuto per ottenere la rappresentazione in complemento a 2; in questa rappresentazione il range non è simmetrico.
- Segno e modulo, l'intervallo di rappresentazione è simmetrico, e consiste nell'usare un bit per il segno e gli altri n-1 bit per il modulo;
- Biased, questa rappresentazione utilizza il bias, ossia il numero che equivale a $2^{n-1} - 1$ nell'intervallo considerato(circa la metà), e tale intervallo non è simmetrico.

La mantissa di un numero reale può essere rappresentata, più comodamente, tramite la notazione scientifica, che contiene le cifre significative, e l'esponente, che indica gli zeri significativi. Per individuare univocamente la notazione scientifica di un numero si effettua la normalizzazione della mantissa di quest'ultimo, da cui discende il valore dell'esponente.

Nello standard IEEE 754 un numero reale floating-point normalizzato viene rappresentato, nei numeri reali a singola precisione(float), con 1 bit per il segno, 8 bit per l'esponente che viene visualizzato tramite rappresentazione biased, e 23 bit per la mantissa che viene rappresentata tramite segno e modulo, con l'utilizzo del bit implicito, che per i numeri normalizzati vale sempre 1. Nei numeri reali a doppia precisione(double) invece, si utilizza 1 bit per il segno, 11 per l'esponente e 52 per la mantissa.

Programmazione II

Valori esponente e mantissa casi specifici:

Oggetto	Caratterizzazione: esponente **e**	mantissa **m**	ℓ
Numeri normalizzati valore $= (-1)^s[\ell.m]\times 2^{e-Bias}$	$E_{min} < e < E_{max}$	$m \geq 0$	1
Infinito con segno	$e = E_{max}$	$m = 0$	-
NaN (Not A Number)	$e = E_{max}$	$m \neq 0$	-
Zero con segno	$e = E_{min}$	$m = 0$	-
Numeri denormalizzati valore $= (-1)^s[\ell.m]\times 2^{e-Bias+1}$	$e = E_{min}$	$m \neq 0$	0

Nello schema Round to Nearest(RN) l'arrotondamento avviene in base al valore del primo bit da eliminare della mantissa, ossia il t+2-simo: se questo vale 1, allora viene aggiunto un 1 nel bit meno significativo della mantissa. Per evitare errori sistematici, se la mantissa del numero è equidistante dalla mantissa di due numeri floating-point, allora il RN arrotonda la mantissa del numero considerato alla mantissa del numero che tra i due è pari.

Programmazione II

Variabili extern

Le variabili extern sono variabili globali condivise da function memorizzate in diversi file dello stesso progetto.

Programmazione II

Aree di memoria

Esistono tre aree di memoria in C:

- Memoria automatica, detta stack(le variabili locali classiche);
- Memoria dinamica, detta heap(variabili che vengono allocate tramite malloc e calloc);
- Memoria statica, ossia le variabili contrassegnate dalla keyword static oppure le variabili globali.

Solo la memoria automatica è limitata superiormente, mentre la memoria dinamica e la memoria statica potenzialmente possono occupare tutto lo spazio messo a disposizione dal sistema operativo. L'ammontare delle memorie stack e heap varia durante l'esecuzione del programma.

Programmazione II

Variable Length Array (VLA)

I Variable Length Array(VLA) sono array il cui size viene conosciuto al tempo dell'esecuzione. Il compilatore GNU gcc alloca queste variabili nella memoria automatica(stack) e l'accesso a tale memoria risulta più efficiente rispetto all'accesso nella memoria dinamica(heap), che richiede l'uso dei puntatori.

Programmazione II

File

Per utilizzare un file, sia di testo che binario, serve:

- Aprirlo
- specificare la modalità di apertura(es. “r”)
- dichiarare un puntatore di tipo FILE che aggiorneremo ad ogni operazione
- infine, il file dovrà essere chiuso.

In un file binario per scrivere e leggere si possono utilizzare solamente fread e fwrite, e per riposizionare il file pointer si può utilizzare fseek. Per quanto riguarda i file di testo invece, ci sono tante funzioni da poter utilizzare, ad esempio fgets e fputs per gli input non formattati e fscanf e fprintf per gli input formattati. La fine di un file testo può essere individuata sia confrontando il carattere letto con EOF che tramite la function C feof, che restituisce 0(falso) se non è stata raggiunta la fine del file e un valore diverso da 0(vero) se è stata raggiunta, mentre in un file binario la fine può essere individuata solo tramite la function feof.

Programmazione II

OOP (Object Oriented Programming)

Alla base della programmazione orientata agli oggetti(OOP – Object Oriented Programming) c'è l'oggetto, le cui caratteristiche sono descritte da una classe. Un oggetto è caratterizzato dai campi, ossia i suoi attributi che possono avere visibilità diverse(private, protected e public), e dai metodi di una classe, ossia i comportamenti o le azioni. Di solito le variabili che descrivono lo stato sono locali e non sono visibili fuori dall'oggetto(data hiding). Il comportamento di un oggetto è controllato da funzioni che agiscono sulle variabili di stato e si interfacciano con l'esterno(encapsulation).
Come detto in precedenza la visibilità di un attributo, o di un metodo, può essere di tre tipi:

- Public, funzioni e dati accessibili da altre parti del programma;
- Protected, funzioni e dati accessibili dalla classe e dalle sue sottoclassi;
- Private, funzioni e dati accessibili solo all'interno della classe.

In una classe un costruttore è un metodo con lo stesso nome della classe che viene utilizzato per istanziare un oggetto della classe e inizializzare i suoi dati, mentre il distruttore serve a deallocare la memoria allocata dal costruttore, anch'esso ha lo stesso nome della classe ma preceduto dal carattere "~".
Il costruttore e il distruttore non hanno valori di ritorno(void). Il distruttore non si può invocare da programma, infatti viene invocato automaticamente dal compilatore quando viene deallocato lo spazio di memoria assegnato all'oggetto. Se il programmatore non crea alcun costruttore, il compilatore crea automaticamente un costruttore di default; se invece viene implementato un costruttore con parametri, o di copia, se si vuole utilizzare anche il costruttore di default bisogna definirlo. Per quanto riguarda i distruttori invece, il distruttore di default elimina tutte le variabili statiche, mentre un distruttore non di default, se dichiarato e inserendo le specifiche giuste, può eliminare variabili dinamiche.

Programmazione II

Ereditarietà

Una classe(sottoclasse) può essere derivata da un'altra classe(superclasse) attraverso il principio dell'ereditarietà, che può avvenire con tre tipi di accesso diversi:

- Public, sono accessibili i membri public e quelli protected, con lo stesso specificatore d'accesso, ma non quelli private;
- Protected, sono accessibili i membri public e quelli protected, che però saranno accessibili solo tramite specificatore d'accesso protected, mentre non sono accessibili quelli private;
- Private, sono accessibili i membri public e quelli protected, che però saranno accessibili solo tramite specificatore d'accesso private, mentre non sono accessibili quelli private

Programmazione II

Upcasting

L’upcasting è la proprietà per cui si può accedere ad un oggetto di classe derivata tramite puntatore a classe base; si può però accedere solo ai membri della classe base ereditati dalla classe derivata. Non si può invece accedere ad un oggetto della classe base tramite puntatore ad una classe derivata.

Programmazione II

Funzione virtuale

Una funzione virtuale è una funzione membro dichiarata in una classe base e ridefinita in una classe derivata e sono lo strumento principale del polimorfismo.

Programmazione II

Tipi di polimorfismo

Esistono due tipi di polimorfismo:

- Function overloading(compile time polymorphism) indica la possibilità di utilizzare più funzioni con lo stesso nome se differiscono nella signature, ossia per tipo, numero o posizione dei parametri. Le funzioni overloaded hanno lo stesso scope e vengono usate per funzioni con lo stesso nome che si comportano in modo diverso in base ai parametri passati ad esse;
- Function overriding(run-time polymorphism) indica la possibilità per una classe derivata di ridefinire una funzione della classe base con la stessa signature, cioè con tipo dei parametri e valore di ritorno uguali. Le funzioni overridden hanno scope differenti e sono necessarie quando una funzione della classe derivata deve avere qualche funzione aggiuntiva o diversa da quella della classe base.

Programmazione II

Classe astratta

Una classe è astratta se contiene almeno una funzione virtuale pura, che non ha una definizione in essa. Generalmente le funzioni virtuali pure si utilizzano nelle classi base astratte. In tal caso ciascuna classe derivata deve prevedere la propria definizione altrimenti verrà considerata anch'essa astratta. Non si può istanziare un oggetto di classe astratta.

Programmazione II

Template

In C++ è possibile definire delle funzioni o classi generiche che abbiano come parametro il tipo di dato, potendo utilizzare lo stesso codice per tipi differenti. In questo modo si crea un polimorfismo parametrico, ed in particolare il compilatore genera automaticamente il codice corretto per il tipo di dato usato nella chiamata alla funzione; si parla quindi di overloading automatico. Una funzione o classe generica si chiama template.

Programmazione II

Scope resolution operator

Il simbolo “`::`” è chiamato scope resolution operator e serve ad implementare un metodo al di fuori della classe, in particolare questo operatore informa il compilatore che tale metodo appartiene, nel caso

```
nome_classe :: nome_metodo
```

alla classe `nome_classe` e di conseguenza esso è nello scope di `nome_classe`.

Programmazione II

Namespace

Il namespace è una regione dichiarativa il cui obiettivo è “localizzare” i nomi degli identificatori evitando collisioni di nome. Gli elementi dichiarati in un namespace sono separati dagli elementi dichiarati in un altro namespace.

La clausola using ha due forme generali:

- `using namespace nome`: si riferisce al namespace che si vuole utilizzare;
- `using nome::member`: fa riferimento ad un singolo membro del namespace.

Programmazione II

Overload di operatori

In una classe si possono definire degli operatori che sostituiscano quelli di default del C++(overload di operatori) la cui sintassi è:

```
return_type class_name::operator#(arg_list)
```

Il simbolo `#` indica l'operatore che si vuole ridefinire.

Programmazione II

Member initializer list – lista di inizializzazione membri

La lista di inizializzazione dei membri è una caratteristica propria dei costruttori di classe; essa appare sempre tra la lista di argomenti del costruttore e il suo corpo ed è preceduta da ":". La lista di inizializzazione va usata:

- per inizializzare i dati membri costanti non statici;
- per inizializzare i dati membro reference;
- per inizializzare gli oggetti membro che non hanno un costruttore default;
- per inizializzare i dati membro della classe base;
- quando il nome del costruttore è lo stesso del dato membro(in alternativa si può usare il puntatore this).

Programmazione II

Strutture dinamiche

Le strutture dati dinamiche non hanno un numero fisso di componenti, ma consentono l'inserimento e l'eliminazione di dati dalla struttura durante l'elaborazione e sono classificate in base al tipo di collegamento logico tra le componenti: lineare, gerarchica e reticolare.

Programmazione II

Link

Il link rende esplicito l’ordinamento logico delle informazioni, senza soffermarsi sull’ordinamento fisico, ossia la posizione relativa di memorizzazione.

Programmazione II

Pila (stack)

Lo stack(pila) è una struttura lineare aperta in cui l'accesso alle componenti per l'inserimento e l'eliminazione avvengono solo ad un estremo della struttura, detta testa dello stack. Lo stack utilizza la logica LIFO perché l'ultimo elemento inserito è il primo ad essere eliminato.

Programmazione II

Coda (queue)

La queue(coda) è una struttura lineare aperta in cui l'accesso alle componenti avviene solo ai due estremi: l'eliminazione avviene solo all'inizio della struttura(testa), mentre l'inserimento avviene solo alla fine(fondo).
La queue utilizza la logica FIFO perché il primo elemento inserito è il primo ad essere eliminato.

Programmazione II

Lista lineare

La lista è una struttura lineare, aperta o chiusa, in cui l'inserimento e l'eliminazione delle componenti possono avvenire in una qualsiasi posizione.

- Per inserire un nodo in testa, bisogna:
  1. inserire il link che va dall'elemento nuovo alla testa della struttura
  2. collegare la testa della struttura al nuovo elemento.
- Per inserire un nodo dopo un nodo corrente, bisogna:
  1. inserire il link che va dall'elemento nuovo al successivo
  2. modificare il link dell'elemento corrente affinché punti al nuovo elemento
- Per inserire un nodo in coda, bisogna aggiungere il link che punta dalla fine della struttura al nuovo elemento

La lista è una struttura ordinata perché l'accesso ad un elemento può avvenire solo dopo aver visitato tutti gli elementi che lo precedono.

Programmazione II

Albero

L’albero è una struttura dati gerarchica contenente diversi nodi.
Due tipi particolari di nodi sono la radice, se si parla dell’unico nodo che non ha padre, e le foglie che sono gli unici nodi che non hanno figli. Il grado di un nodo è il numero dei figli

Programmazione II

Albero binario

In un albero binario ogni nodo può avere al massimo due figli, in particolare partendo dalla radice tutto ciò che si trova a partire dal figlio sinistro in poi prende il nome di sottoalbero sinistro, mentre tutto ciò che si trova a partire dal figlio destro in poi prende il nome di sottoalbero destro. Un albero binario si dice completo se ha esattamente due figli per ogni nodo tranne che per le foglie, mentre si dice quasi completo se e solo se sono completi tutti i suoi livelli tranne al più l'ultimo. Il numero totale dei nodi, in un albero binario completo di profondità l, è $2^l - 1$, mentre il numero di foglie è 2^{l-1}.

Per rappresentare un albero binario completo tramite array in C si può partire con la memorizzazione degli elementi dall'indice 1, che sarà la radice; per accedere al figlio sinistro faremo 2*i, mentre per accedere al figlio destro 2*i+1. Al contrario invece per passare dal figlio al padre si prende la parte intera della divisione i/2. Se invece vogliamo partire dall'indice 0 dovremmo aggiungere 1 all'indice i che equivale alla posizione degli elementi.
La rappresentazione di un albero binario mediante array va bene solo se l'albero è quasi bilanciato ed inoltre nell'array sono inefficienti operazioni quali inserimento ed eliminazione di un nodo in quanto richiedono lo spostamento di porzioni di array.

Programmazione II

Algoritmo di visita

Qualunque struttura si utilizzi per la memorizzazione di un albero binario, che sia array o lista, l'algoritmo di visita deve far uso di uno stack, che va gestito esplicitamente nell'algoritmo iterativo, e implicitamente nell'algoritmo ricorsivo. Lo stack conserva i nodi di cui non si è ancora visitato il sottoalbero destro.

Programmazione II

Albero binario di ricerca

L’albero binario di ricerca(BST) è un particolare albero binario in cui la visita inorder permette di accedere alle informazioni in ordine crescente di chiave; inoltre, in questo particolare albero binario è molto semplice effettuare la ricerca binaria. Nel BST i nodi vengono visitati in modo crescente, in particolare, per ogni nodo, qualsiasi nodo che si trova a sinistra sarà minore del nodo considerato, mentre qualsiasi nodo che si trova a destra sarà maggiore del nodo considerato. La ricerca binaria risulta molto semplice perché si può considerare come “mediano” la radice, visto che a sinistra ci saranno tutti valori più piccoli e a destra tutti valori più grandi.

Programmazione II

Heap

Un heap è un albero binario quasi completo i cui nodi padre devono sempre essere maggiori dei nodi figli(nella proprietà max-heap) o sempre minori(nella proprietà min-heap). Ne consegue che il massimo è memorizzato nella radice. Per rendere un albero binario quasi completo un heap, si deve applicare la procedura di heapify, la quale:

- considera un nodo ed i suoi figli
- determina il figlio con chiave massima
- se questo ha chiave maggiore del padre, allora effettua lo scambio tra loro.
- Se è avvenuto lo scambio, si scende poi nel relativo sottoalbero per continuare ad applicare la procedura di heapify.

Programmazione II

Coda con priorità

Una coda con priorità è una struttura dati che serve per mantenere un insieme di elementi, ciascuno con un valore che rappresenta la sua priorità. Analogamente agli heap, esistono due tipi di code con priorità, quelle con max-priorità e quelle con min-priorità.

Programmazione II

Grafo

Il grafo è una struttura reticolare formata da un insieme di nodi o vertici, ossia i punti, e un insieme di archi, cioè le connessioni tra due nodi. Il grafo è orientato quando è assegnato un verso di percorrenza agli archi, mentre non è orientato quando non vi è assegnato alcun verso. Nei grafi orientati è consentito che tra una coppia di nodi vi siano due archi purché questi abbiano versi opposti. Il grado di un grafo è il numero di archi incidenti.
Il cammino da x a y(due vertici) è la lista dei vertici connessi da archi che vanno da x a y e la sua lunghezza è determinata dal numero degli archi attraversati. Alcuni tipi particolari di cammino sono:

- quello che attraversa meno archi viene definito cammino minimo
- quello che non passa due volte per uno stesso nodo prende il nome di cammino semplice.

Se in un cammino semplice il primo e l'ultimo nodo coincidono, allora si parla di ciclo. Un caso particolare di ciclo, di lunghezza 1, prende il nome di cappio o loop.

Il ciclo Euleriano è un ciclo che attraversa ogni arco esattamente 1 volta, mentre il ciclo Hamiltoniano è un ciclo che visita ogni nodo, tranne il primo, esattamente 1 volta.

Programmazione II

Visita di un grafo

Per visitare i nodi di un grafo vi sono due algoritmi:

- Depth First Search(DFS – visita per profondità), che viene descritto molto semplicemente in forma ricorsiva, infatti basta visitare un nodo e ricorsivamente visitare ogni nodo adiacente al nodo di partenza e non ancora visitati. Nella versione ricorsiva lo stack, usato per ricordare i vertici ancora non visitati, è gestito implicitamente dalla ricorsione. La versione iterativa invece viene realizzata utilizzando esplicitamente uno stack, ad ogni visita di un nuovo nodo si inseriscono i suoi nodi adiacenti. Ad ogni passo si estrae un nodo dallo stack e, se non è stato visitato, si visita. Per descrivere il funzionamento dell'algoritmo si utilizza un flag per ogni vertice che indica se il nodo corrente è stato visitato o meno. L'algoritmo DFS coincide con la visita preorder di un albero la cui radice è il nodo da cui si parte; infatti, tale algoritmo trasforma il grafo in un albero Depth First Search in cui non sono presenti tutti gli archi, ma quelli presenti sono attraversati una volta. Invece nella visita DFS tutti gli archi del grafo sono attraversati due volte;
- Breadth First Search(BFS – visita per ampiezza), che viene normalmente implementata tramite una queue: ogni volta che si visita un nodo lo si mette in una coda e ad ogni passo si estrae un nodo e si controllano i nodi ad esso adiacenti, se non sono stati ancora visitati li si visita. La visita in ampiezza del grafo ha prodotto un albero BFS tale che:
  - La radice è il nodo sorgente;
  - Sono toccati tutti i nodi del grafo, se è connesso, mentre gli archi sono un sottoinsieme di quelli del grafo;
  - È possibile calcolare le lunghezze dei cammini che partono dalla radice mediante il livello;
  - È possibile stabilire se un dato sia connesso con la radice dell'albero.

Programmazione II

Ricorsione

La ricorsione si divide in:

- ricorsione diretta, che si divide a sua volta in:
  - Lineare, 1 sola chiamata ricorsiva nella procedura;
  - Binaria, 2 chiamate ricorsive nella procedura;
  - Non lineare, nella procedura almeno una chiamata ricorsiva avviene dentro un ciclo.
- ricorsione indiretta, dove vi è la mutua ricorsione, che riguarda una chiamata ricorsiva indiretta tramite un’altra procedura.

Programmazione II

Profondità di ricorsione

La profondità, o livello, di ricorsione è il massimo numero di chiamate ricorsive eseguite; essa dipende dal valore particolare dei dati di input.
La profondità di ricorsione moltiplicata per il numero di variabili usate dalla procedura indica l'occupazione in memoria dello stack di dati temporanei della procedura ricorsiva. La profondità di ricorsione equivale alla complessità computazionale dell'algoritmo.

Programmazione II

Problemi di ricerca e ordinamento dati

I problemi di ricerca e di ordinamento di dati, rispetto ad un campo dell'informazione detto chiave, sono molto importanti nelle applicazioni. Esistono vari metodi per risolvere lo stesso problema: questi si confrontano rispetto al parametro di efficienza, in relazione al particolare problema e all'organizzazione dei dati. Nella valutazione dell'algoritmo si considerano anche i casi limite di dati già ordinati oppure totalmente disordinati.
I metodi di ordinamento si dividono in:

- interni, in cui i dati si trovano nella memoria centrale
- esterni, in cui i dati sono sulla memoria di massa parzialmente in memoria centrale.

Le strutture di base utilizzate per ordinare i dati sono l'array e la lista. Alcuni algoritmi di ordinamento(su array) effettuano scambi di informazioni di tipo:

- reale, se i dati sono fisicamente scambiati di posto
- virtuale se i dati non sono fisicamente scambiati poiché si opera tramite puntatori.

Programmazione II

Merge sort

Nell'algoritmo di Merge sort l'idea iniziale è dividere un array in due metà, che a loro volta auto-attiveranno la procedura di ordinamento su se stesse, diminuendo di volta in volta il size della porzione considerata. Una volta ordinate, le due porzioni vengono fuse attraverso la classica procedura di merge tra due array ordinati. Quest'algoritmo non opera in place e la sua complessità di tempo è di tipo lin-log($n\log_2 n$), poiché al numero di passi del merge sort($\log_2 n$) si moltiplica la complessità di tempo dell'algoritmo di merge che è n.

Programmazione II

Quick sort

Nell'algoritmo di Quick sort l'idea da seguire è quella di scegliere un partizionatore, ossia un elemento che appunto partiziona il vettore in due sotto-vettori e che occupa il suo posto nel vettore ordinato. Ad ogni passo, dopo aver scelto il partizionatore:

1. si scorre il vettore da sinistra verso destra fino a trovare un elemento dell'array maggiore del partizionatore
2. si scorre da destra verso sinistra per trovare un elemento minore del partizionatore
3. si scambiano i due elementi.

In questo caso l'algoritmo di ordinamento opera in place, ma la complessità di tempo varia di caso in caso. Nel caso migliore si parla di complessità lin-log($\mathrm{nlog}_2\, n$), mentre nel caso peggiore si parla addirittura di complessità quadratica(n^2). La complessità di tempo di questo algoritmo dipende dalla scelta del partizionatore: infatti, questa influisce sul fatto che ad ogni passo esso sia più o meno bilanciato. Nel caso di dati quasi del tutto ordinati il Quick sort fornisce la peggiore prestazione e non va usato.

Il partizionatore si può scegliere in modo random, e in questo caso avremo una distribuzione uniforme, oppure sfruttando la distribuzione di probabilità dei dati se è nota. La miglior scelta per il partizionatore è il valore mediano, ossia quello che divide i dati in due sottoinsiemi di eguale cardinalità.

Programmazione II

Heap sort

Nell'algoritmo di Heap sort i dati sono memorizzati in un albero binario e l'idea è quella di costruire un heap, tramite la procedura heapify in modalità bottom-up e successivamente riordinare l'heap tramite l'utilizzo della procedura heapify, ma stavolta in modalità top-down. La complessità di quest'algoritmo è lin-log($n\log_2 n$).