Numeri in Virgola Fissa e Programmazione Orientata agli Oggetti in C++

Parte I: Rappresentazione Fixed-Point

1. Introduzione ai Numeri Fixed-Point

La rappresentazione in virgola fissa costituisce un metodo alternativo alla codifica floating-point per la manipolazione di valori numerici decimali mediante strutture dati intere. A differenza della notazione scientifica impiegata dai numeri in virgola mobile, dove un valore viene rappresentato attraverso una mantissa e un esponente, la virgola fissa adotta un approccio deterministico: divide a priori lo spazio di un intero in due regioni distinte, una destinata alla parte intera e una alla parte frazionaria.

Questo è ottenuto mediante un'operazione di shift binario, che fissa permanentemente la posizione del punto decimale entro la rappresentazione. Se si decide, ad esempio, di allocare otto bit per la frazione, allora il punto decimale sarà posizionato precisamente dopo il bit numero otto da destra, indipendentemente dal valore contenuto nel registro.

Il vantaggio principale risiede nella prevedibilità e nella costanza della precisione attraverso l'intero range di valori. Inoltre, l'assenza di operazioni aritmetiche in virgola mobile comporta performances superiori, aspetto critico negli ambienti embedded, real-time e nei sistemi dove il determinismo è prioritario. Tuttavia, questa semplicità computazionale comporta un compromesso: il range di valori rappresentabili è inferiore rispetto a quanto ottenibile con virgola mobile, e la precisione per numeri molto piccoli o molto grandi risulta limitata dalla granularità fissa scelta.

2. **Struttura Binaria e Bit Frazionari**

Consideriamo un intero a trentadue bit con un'allocazione di otto bit per la parte frazionaria. La suddivisione interna del registro segue questo schema:

- **Bit 31-8** (ventiquattro bit): Destinati alla rappresentazione della parte intera, con il bit 31 che funge da bit di segno in caso di rappresentazione complemento a due.
- **Bit 7-0** (otto bit): Riservati alla rappresentazione della parte frazionaria.

La quantità di bit dedicata alla frazione—nel nostro caso otto—determina la risoluzione minima rappresentabile. Con otto bit frazionari, la minima differenza distinguibile tra due valori consecutivi è pari a 2^(-8), equivalente a 1/256, approssimativamente 0.00390625.

Un intero archiviato nel formato fixed-point, quando reinterpretato secondo la convenzione descritta, rappresenta un valore numerico che può essere ottenuto dividendo il valore grezzo per 2^(numero di bit frazionari). Inversamente, per convertire un valore decimale nel formato fixed-point, si moltiplica il valore per 2^(numero di bit frazionari) e si arrotonda il risultato all'intero più prossimo.

3. Operazioni Aritmetiche Fondamentali

Addizione e Sottrazione

L'addizione di due numeri fixed-point con lo stesso numero di bit frazionari è un'operazione diretta: si sommano semplicemente i valori grezzi, poiché il fattore di scala è già coerente in entrambi gli operandi. Il risultato manterrà automaticamente il corretto positioning del punto decimale.

Analogamente, la sottrazione si riduce a un'operazione di sottrazione tra interi, senza la necessità di alcun aggiustamento di scala.

Moltiplicazione

La moltiplicazione, tuttavia, richiede un'attenzione particolare. Quando si moltiplicano due numeri fixed-point, ciascuno rappresentato come (valore × 2^b, dove b è il numero di bit frazionari), il prodotto risultante sarà proporzionale a ($valore_1$ × $valore_2$) × 2^(2b). Poiché desideriamo mantenere il formato fisso con b bit frazionari, è necessario ridurre il fattore di scala di un esponente: si esegue uno shift binario destro di b posizioni, che equivale a una divisione per 2^b, ripristinando così il corretto fattore di scala.

Divisione

La divisione presenta un'esigenza inversa. Se dividiamo (a × 2^b) per (b × 2^b), otteniamo a/b, perdendo il fattore di scala desiderato. Per preservare la precisione, prima di eseguire la divisione si esegue uno shift sinistro del dividendo di b posizioni, moltiplicandolo per 2^b. In questo modo, la divisione restituisce un risultato già correttamente scalato al formato fixed-point.

4. Conversione tra Rappresentazioni

La conversione da un intero a formato fixed-point è ottenuta mediante uno shift sinistro del valore intero di b posizioni, dove b è il numero di bit frazionari designati. Questo equivale a moltiplicare per 2^b.

La conversione da un numero floating-point (tipicamente un float o double) richiede una moltiplicazione per 2^b, seguita da arrotondamento all'intero più prossimo, al fine di minimizzare l'errore di quantizzazione.

La conversione inversa, da fixed-point a float, si realizza dividendo il valore greggio per 2^b mediante operazioni in virgola mobile.

La conversione a intero, operazione che scarta la parte frazionaria, si ottiene mediante uno shift destro di b posizioni, poiché ciò equivale a una divisione intera per 2^b.

5. Caratteristiche e Limitazioni

La risoluzione di un sistema fixed-point è determinata irrevocabilmente dalla scelta del numero di bit frazionari. Tale scelta rappresenta un compromesso tra il range massimo rappresentabile e la granularità minima. Un'allocazione più generosa di bit alla frazione migliora la precisione ma riduce l'intervallo di interi rappresentabili; un'allocazione minore comporta il fenomeno opposto.

Il range massimo dipende sia dal numero totale di bit disponibili sia dall'allocazione tra parte intera e parte frazionaria. In uno schema a trentadue bit con otto bit frazionari e rappresentazione in complemento a due, il range approssimativo è compreso tra -8.388.607 e +8.388.607.

L'overflow e l'underflow rimangono rischi significativi, particolarmente in operazioni moltiplicative, dove il prodotto può eccedere il range dichiarato senza avviso esplicito in molti linguaggi.

Parte II: Programmazione Orientata agli Oggetti in C++

6. Principi della Programmazione Orientata agli Oggetti

La programmazione orientata agli oggetti rappresenta un paradigma dichiarativo e compositivo dove i dati e le operazioni su tali dati vengono incapsulati entro entità autonome denominate oggetti. Un oggetto è un'istanza di una classe, la quale definisce tanto la struttura dei dati (attributi) quanto le operazioni permesse su di essi (metodi).

7. La Forma Canonica Ortodossa (Orthodox Canonical Form)

Quando si progetta una classe in C++ che gestisce la semantica di copia e costruzione, è considerata una pratica fondamentale l'implementazione di quella che è comunemente denominata "Forma Canonica Ortodossa" o "Orthodox Canonical Form". Questo pattern prescrive l'implementazione di quattro funzioni membro speciali che definiscono il comportamento della classe:

7.1 Il Costruttore di Default

Il costruttore di default è invocato automaticamente al momento della creazione di un oggetto, qualora nessun argomento sia esplicitamente fornito. La responsabilità primaria di questo metodo è quella di inizializzare tutti i dati membro a uno stato valido e coerente.

L'assenza di un costruttore di default consente al compilatore di generarne uno implicitamente; tuttavia, la generazione implicita potrebbe non istanziare i dati membro in modo appropriato per il contesto specifico dell'applicazione. Per questa ragione, la definizione esplicita di un costruttore di default è spesso desiderabile, anche se il corpo della funzione risulta triviale.

7.2 Il Costruttore di Copia

Il costruttore di copia viene invocato quando un nuovo oggetto viene inizializzato come copia di un oggetto esistente della medesima classe. La sua responsabilità è duplicare lo stato del prototipo, assicurandosi che l'oggetto neocostruito sia una copia indipendente e funzionalmente equivalente.

Nel caso di classi le cui istanze contengono solamente tipi primitivi o riferimenti a oggetti senza gestione di risorse dinamiche, una copia bit-per-bit (copia superficiale) risulta appropriata. Tuttavia, qualora una classe gestisca memoria dinamica o altri tipi di risorse, è necessaria una copia profonda, dove le risorse allocate vengono replicate piuttosto che condivise.

7.3 L'Operatore di Assegnazione per Copia

L'operatore di assegnazione per copia, distinto dal costruttore di copia per il fatto che opera su un oggetto già costruito, trasferisce lo stato di un oggetto sorgente a un oggetto destinazione preesistente. A differenza del costruttore di copia, l'oggetto destinazione potrebbe già possedere risorse che necessitano essere rilasciate prima che le nuove risorse vengano acquisite.

Una pratica cruciale nell'implementazione di questo operatore è il controllo dell'auto-assegnazione, mediante il quale si verifica se l'indirizzo dell'oggetto corrente coincide con l'indirizzo dell'oggetto sorgente.

L'operatore di assegnazione deve ritornare un riferimento all'oggetto corrente, facilitando la concatenazione di assegnazioni sequenziali e conformandosi alle aspettative sintattiche di altre parti del codice.

7.4 Il Distruttore

Il distruttore è invocato automaticamente quando un oggetto cessa di esistere, sia attraverso il raggiungimento della fine del suo ambito di validità (scope) sia per mezzo di un'esplicita deallocazione. La responsabilità del distruttore è quella di liberare qualunque risorsa acquisita durante la vita dell'oggetto.

Per classi che non gestiscono risorse dinamiche, il distruttore può rimanere vuoto; il compilatore potrebbe addirittura generarne uno implicitamente. Tuttavia, la definizione esplicita, sebbene triviale, rappresenta una buona pratica documentaria.

9. Overload degli Operatori

L'overload degli operatori è un meccanismo che consente al programmatore di ridefinire il significato di operatori predefiniti (quali +, -, *, ==, <, etc.) quando applicati a istanze di classi personalizzate. Ciò consente ai tipi definiti dall'utente di comportarsi in modo analogo ai tipi primitivi del linguaggio, migliorando la leggibilità e l'intuitività del codice.

9.1 Operatori di Confronto

Gli operatori di confronto—maggiore di, minore di, uguaglianza, disuguaglianza, ecc.—permettono di stabilire relazioni ordinali tra istanze di una classe. Questi operatori sono tipicamente implementati come funzioni membro costanti, poiché non modificano lo stato degli oggetti coinvolti nel confronto.

L'implementazione di operatori di confronto coerenti e non contraddittori è essenziale per garantire il corretto funzionamento di algoritmi standard che si basano su tali relazioni, quali gli algoritmi di ordinamento forniti dalla Standard Library.

9.2 Operatori Aritmetici

Gli operatori aritmetici—addizione, sottrazione, moltiplicazione, divisione—consentono operazioni numeriche significativamente più naturali e leggibili rispetto alla chiamata a funzioni denominate convenzionalmente (quale un ipotico metodo `add()`).

Tipicamente, questi operatori sono implementati come funzioni membro costanti che retournano un nuovo oggetto contenente il risultato dell'operazione, mantenendo inalterate le istanze dei due operandi.

9.3 Operatori di Incremento e Decremento

Gli operatori di incremento e decremento presentano una sottigliezza sintattica: esiste sia una forma di pre-incremento (operatore applicato prima dell'estrazione del valore) sia una forma di post-incremento (operatore applicato dopo l'estrazione del valore).

Il pre-incremento è tipicamente implementato come una funzione membro costante che ritorna un riferimento all'oggetto medesimo dopo aver incrementato il suo stato. Il post-incremento, al contrario, salva lo stato precedente, esegue l'incremento e retorna una copia del valore precedente. Questa semantica differenziata è denotata in C++ mediante l'utilizzo di un parametro `int` dummy nel caso del post-incremento, convenzione che permette al compilatore di disambiguare tra le due varianti.

9.4 Operatori Non-Membro e Amicizia

Taluni operatori, per ragioni di simmetria sintattica o di progettazione, sono implementati come funzioni non-membro piuttosto che come metodi della classe. L'operatore di stream output (<<), ad esempio, è una funzione non-membro perché l'operando sinistro appartiene a una classe della Standard Library (`std::ostream`) su cui non si desidera modificare il codice.

10. Funzioni Membro Statiche

Una funzione membro statica è una funzione che appartiene alla classe piuttosto che a una specifica istanza. Non possiede un puntatore implicito `this` e può accedere solamente a dati membro statici o a altre funzioni statiche della classe.

Le funzioni statiche sono utili quando si desiderano operazioni logicamente associate a una classe ma che non necessitano dello stato di una particolare istanza. Ad esempio, una funzione che calcoli il minimo o il massimo tra due istanze è naturalmente candidate a essere statica, poiché rappresenta un'operazione di natura comparativa e generale piuttosto che un'operazione che modifica o legge lo stato di una specifica istanza.

L'invocazione di una funzione membro statica avviene attraverso il nome della classe, non attraverso un'istanza: `Classe::funzioneStatica()`.

Parte III: Applicazioni Geometriche

11. Binary Space Partitioning per Triangoli

Il Binary Space Partitioning (BSP) rappresenta una tecnica di decomposizione dello spazio geometrico in regioni disgiunte, storicamente utilizzata in applicazioni di computer graphics e computational geometry per accelerare operazioni quali il culling visuale, l'intersezione di geometrie, e l'ordinamento di superfici.

Nel contesto della determinazione dell'appartenenza di un punto a un triangolo, il BSP si basa su un algoritmo relativamente semplice: si verifica se un punto assegnato rimane dalla medesima parte rispetto a tutti e tre i lati del triangolo. Se questa condizione è soddisfatta, il punto è interno al triangolo; altrimenti, il punto è esterno.

12. Il Prodotto Vettoriale e la Determinazione dell'Orientamento

Il prodotto vettoriale in due dimensioni, talvolta denominato pseudo-prodotto vettoriale o determinante 2D, è un'operazione che produce uno scalare indicativo dell'orientamento relativo di due vettori.

Dati due vettori bidimensionali, il loro prodotto vettoriale è calcolato secondo la formula seguente: per i vettori u = (u_x, u_y) e v = (v_x, v_y), il prodotto è u_x × v_y - u_y × v_x.

Il segno del risultato comunica informazioni cruciali sull'orientamento: un risultato positivo indica che il secondo vettore si trova in orientamento antiorario rispetto al primo; un risultato negativo suggerisce un orientamento orario; un risultato nullo indica collinearità.

13. Algoritmo di Appartenenza Puntuale

Per determinare se un punto P si trova all'interno di un triangolo con vertici A, B e C, si calcolano i tre prodotti vettoriali seguenti: il prodotto associato al lato AB, il prodotto associato al lato BC, e il prodotto associato al lato CA.

Se il punto P giace esattamente su uno qualunque dei lati o dei vertici del triangolo, almeno uno di questi prodotti sarà nullo. Per le definizioni più comuni, i punti sui bordi sono considerati esterni al triangolo.

Se il punto P è interno al triangolo, tutti e tre i prodotti vettoriali avranno il medesimo segno algebrico: saranno cioè tutti positivi o tutti negativi, a seconda dell'orientamento globale del triangolo (orario o antiorario). Se i segni sono misti—alcuni positivi e altri negativi—il punto è esterno.

Questa proprietà geometrica discende dall'osservazione che un punto interno rimane invariabilmente dalla medesima parte rispetto a ogni lato del triangolo, mentre un punto esterno deve necessariamente attraversare almeno uno dei lati, modificando il segno del prodotto vettoriale associato.