Programmazione Avanzata

Metodo Inline

Pierluigi Roberti

pierluigi.roberti@unitn.it

Inline

È stata introdotta la funzione **inline** che è una tecnica di ottimizzazione utilizzata dai compilatori soprattutto per ridurre i tempi di esecuzione.

Le funzioni possono essere impostate per renderle **inline** in modo che il compilatore possa sostituire quelle definizioni di funzione ovunque vengano chiamate. Il compilatore sostituisce la definizione delle funzioni **inline** in fase di compilazione invece di fare riferimento alla definizione della funzione in fase di esecuzione.

NOTA: questo è solo un **suggerimento al compilatore** per rendere la **funzione inline**, se la funzione è grande (in termini di istruzioni eseguibili ecc.), il compilatore può ignorare la richiesta "inline" e trattare la funzione come una normale funzione.

Uso di inline

```
class A1 {
 public:
    inline int add(int a, int b) {
       return (a + b);
    }
};

class A2 {
 public:
    int add(int a, int b);
};

inline int A2::add(int a, int b) {
   return (a + b);
}
```

Perchè usare inline 1/2

- In molti posti creiamo le funzioni per piccoli lavori/funzionalità che contengono un numero semplice e poche di istruzioni eseguibili. Pensiamo al sovraccarico delle loro chiamate ogni volta che vengono invocate.

- Quando viene incontrata una normale istruzione di chiamata di funzione, il programma **memorizza l'indirizzo di memoria** delle istruzioni immediatamente dopo l'istruzione di chiamata di funzione, carica la funzione chiamata in memoria, copia i valori degli argomenti, salta alla posizione di memoria della funzione chiamata, esegue il codici funzione, memorizza il valore di ritorno della funzione, quindi torna all'indirizzo dell'istruzione che era stata salvata appena prima dell'esecuzione della funzione richiamata.

- Troppo sovraccarico del tempo di esecuzione

Perchè usare inline 2/2

- La funzione **inline** C++ fornisce un'alternativa. Con la parola chiave inline, il **compilatore sostituisce l'istruzione di chiamata di funzione con il codice della funzione stessa** (processo chiamato espansione) e quindi compila l'intero codice. Pertanto, con le funzioni **inline**, il compilatore non deve passare a un'altra posizione per eseguire la funzione e quindi tornare indietro poiché il codice della funzione chiamata è già disponibile per il programma chiamante

- Tutte le **funzioni membro** dichiarate e definite all'interno della **classe sono Inline** per impostazione predefinita. Quindi non c'è bisogno di definire esplicitamente.
- I **metodi virtuali non dovrebbero essere inline**. Tuttavia, a volte, quando il compilatore può conoscere con certezza il tipo dell'oggetto (cioè l'oggetto è stato dichiarato e costruito all'interno dello stesso corpo della funzione), anche una funzione virtuale sarà inline perché il compilatore conosce esattamente il tipo dell'oggetto.

Uso inline: pro

1. Accelera il tuo programma evitando il sovraccarico delle chiamate di funzioni.

2. Risparmia il sovraccarico delle variabili push/pop sullo stack, quando si verifica la chiamata della funzione.

3. Risparmia il sovraccarico della chiamata di ritorno da una funzione.

4. Aumenta la località di riferimento utilizzando la cache delle istruzioni.

5. Contrassegnandolo come inline, si può inserire una definizione di funzione in un file di intestazione (cioè può essere inclusa in unità di compilazione multipla, senza che il linker si lamenti)

Uso inline: contro

1. Aumenta la dimensione dell'eseguibile a causa dell'espansione del codice.
2. L'inline di C++ viene risolto in fase di compilazione. Ciò significa che se si modifica il codice della funzione inline, è necessario ricompilare tutto il codice utilizzandolo per assicurarsi che venga aggiornato!
3. Se utilizzato in un'intestazione, ingrandisce il file di intestazione con informazioni che gli utenti non lo fanno cura.
4. Come accennato in precedenza, aumenta la dimensione dell'eseguibile, che potrebbe causare il **thrashing**(*) in memoria. Maggiore sono gli errori di pagina maggiore sarà la riduzione delle prestazioni del programma.
5. A volte non è utile, ad esempio nei sistemi embedded in cui non si preferisce affatto un eseguibile di grandi dimensioni a causa di vincoli di memoria.

(*) Il termine thrashing generalmente si riferisce alla **situazione in cui la memoria fisica è piena o quasi piena e si verificano scambi di pagine costanti**.