Definizioni

Namespaces

In C++, i namespaces sono utilizzati per organizzare il codice in gruppi logici, allo scopo di evitare conflitti di nomi e migliorare la leggibilità del codice.

Un namespace è un'area di memoria in cui vengono memorizzati nomi di variabili, funzioni, classi e altre entità di programmazione. Un namespace può contenere altre entità di namespace, creando così una gerarchia di namespace.

Ad esempio, si può creare un namespace per le funzioni matematiche, come segue:

```
namespace math {
    double pi = 3.14159265358979323846;

    double radice_quadrata(double x) {
        return sqrt(x);
    }
}
```

In questo esempio, abbiamo creato un namespace chiamato `math`. All'interno di questo namespace, abbiamo definito una costante chiamata `pi` e una funzione chiamata `radice_quadrata`.

Per utilizzare le entità definite in un namespace, è necessario specificare il nome del namespace prima del nome dell'entità. Ad esempio, per utilizzare la costante `pi` definita nel namespace `math`, si può scrivere:

```
double raggio = 5.0;
double circonferenza = 2 * math::pi * raggio;
```

In questo esempio, stiamo utilizzando il nome del namespace `math` per accedere alla costante `pi` definita all'interno del namespace.

L'utilizzo dei namespaces può rendere il codice più leggibile, in quanto evita conflitti di nomi tra le entità di programmazione. Ad esempio, si può utilizzare lo stesso nome di funzione in due namespace diversi, senza generare conflitti tra i nomi.

Inoltre, l'utilizzo dei namespaces consente di utilizzare librerie esterne e di integrare codice da diverse fonti, senza generare conflitti tra i nomi delle entità di programmazione.

In generale, l'utilizzo dei namespaces è una pratica utile per organizzare il codice in modo coerente e migliorare la leggibilità del codice.

Torna all'indice

Templates

In C++, i templates sono una funzionalità che consente di creare codice generico e riusabile. In pratica, un template definisce un modello o un prototipo per una funzione o una classe, consentendo di creare versioni specializzate di tali funzioni o classi per diversi tipi di dati.

Ad esempio, si può creare un template per una funzione che restituisce il valore massimo tra due valori di un determinato tipo di dati:

```
template
T max(T a, T b) {
    return a > b ? a : b;
}
```

In questo esempio, abbiamo creato una funzione chiamata `max` utilizzando la sintassi del template. La parola chiave `template` viene utilizzata per definire il template, seguita dal nome del tipo di dati generico `T`. La funzione `max` prende

due argomenti di tipo `T` e restituisce il valore massimo tra di essi.

Si può utilizzare questa funzione per trovare il massimo tra due numeri interi o tra due numeri in virgola mobile, senza dover scrivere due funzioni diverse:

```
int a = 10, b = 20;
cout << max(a, b) << endl;

double x = 3.14, y = 2.71;
cout << max(x, y) << endl;
```

In questo esempio, stiamo utilizzando la funzione `max` per trovare il massimo tra due numeri interi e tra due numeri in virgola mobile.

I templates sono utili perché consentono di scrivere codice generico e riusabile, che può essere utilizzato con diversi tipi di dati. Inoltre, i templates consentono di scrivere codice più leggibile e mantenibile, poiché il codice ripetitivo può essere eliminato utilizzando il template.

In generale, l'utilizzo dei templates è una pratica utile per creare codice generico e riusabile in C++.

Torna all'indice

Funzioni inline

Le funzioni `inline` in C++ sono utilizzate per richiedere al compilatore di sostituire il codice della funzione al punto in cui viene chiamata. Ciò può migliorare le prestazioni del programma, in quanto evita il tempo di esecuzione richiesto per la chiamata della funzione.

In C++, le *funzioni inline* possono essere definite in due modi:

1. Dichiarazione inline: La parola chiave `inline` viene utilizzata davanti alla dichiarazione della funzione, come nel seguente esempio:

```
inline int somma(int a, int b) {
    return a + b;
}
```

2. Definizione inline: La definizione della funzione viene inserita direttamente nel corpo della classe, dichiarando la funzione come `inline` implicitamente. Questo metodo viene spesso utilizzato per le funzioni membro di classe, ma può essere utilizzato anche per le funzioni globali, come nel seguente esempio:

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

In entrambi i casi, l'utilizzo di una funzione `inline` è un suggerimento al compilatore per sostituire il codice della funzione al punto in cui viene chiamata, ma il compilatore non è obbligato a farlo. In altre parole, il compilatore può ignorare la richiesta di inline se lo ritiene necessario.

Vantaggi

Ci sono alcuni vantaggi nell'utilizzo delle funzioni inline in C++. In primo luogo, il tempo di esecuzione richiesto per la chiamata di una funzione viene eliminato, il che può migliorare le prestazioni del programma. In secondo luogo, l'utilizzo di funzioni inline può ridurre la dimensione del codice generato dal compilatore, in quanto il codice della funzione viene inserito direttamente nel punto in cui viene chiamata, anziché generare codice per la chiamata della funzione.

Svantaggi

Tuttavia, ci sono anche alcuni svantaggi nell'utilizzo delle funzioni `inline`. Ad esempio, l'utilizzo di funzioni inline può aumentare la dimensione del codice generato dal compilatore, in quanto il codice della funzione viene ripetuto in ogni punto in cui viene chiamata. Inoltre, l'utilizzo di funzioni `inline` può aumentare la complessità del codice, in quanto il compilatore deve gestire la sostituzione del codice della funzione al punto in cui viene chiamata.

In generale, le funzioni `inline` dovrebbero essere utilizzate con cautela, e dovrebbero essere applicate solo alle funzioni che vengono chiamate frequentemente e che sono relativamente brevi. Ciò può aiutare a migliorare le prestazioni del programma, senza aumentare la dimensione del codice o la complessità.

==TODO==

- `constexpr`
- `#define PROD(a,b) (a) * (b)`

Torna all'indice
