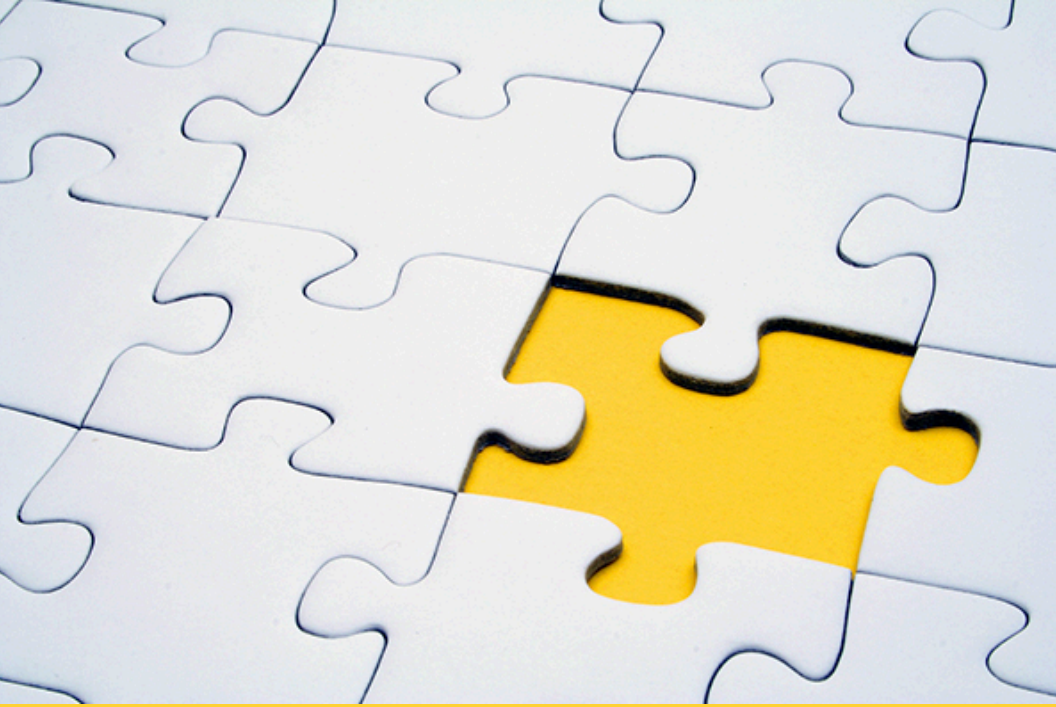

COMPENDIO

ITIS “E. MOLINARI”

PROGRAMMA DI INFORMATICA

Classe 4AI Serale Anno Scolastico 2022/2023

ITIS “E. MOLINARI”
PROGRAMMA DI INFORMATICA

Insegnante: Alemanno Giancarlo

Nel cuore della città di Milano, l'ITIS Molinari continua a formare generazioni di appassionati di tecnologia e programmazione. Questo compendio è il risultato di un anno di studi intensivi, di sfide affrontate e superate, di nuove competenze acquisite. È un tributo alla dedizione degli studenti della classe 4A e alla passione dei loro insegnanti.

Questo non è un libro nel senso tradizionale del termine. È un viaggio attraverso il mondo affascinante della programmazione in C++, un linguaggio potente e versatile. È un invito a scoprire il piacere di risolvere problemi e creare nuove cose con il codice. È un compendio di tutto ciò che abbiamo imparato, un riferimento a cui tornare, un punto di partenza per ulteriori esplorazioni.

ITIS Molinari Milano

20130 Milano, MI

www.itismolinari.edu.it

Prologo

Questo compendio di studi sulla programmazione in C++ è stato creato come un riepilogo e un riferimento per gli studenti della classe 4A dell'ITIS Molinari. Da concetti fondamentali come variabili e cicli, a temi più avanzati come puntatori, strutture dati dinamiche, ereditarietà e programmazione generica, ogni sezione riflette le lezioni apprese in aula e in laboratorio.

In questo compendio, abbiamo cercato di catturare la ricchezza e la profondità del corso di studi in C++ presso l'ITIS Molinari. Ma più di tutto, questo compendio è un tributo all'importanza dell'educazione e dell'apprendimento continuo. Speriamo che serva non solo come un utile strumento di revisione, ma anche come un promemoria del valore dell'indagine intellettuale e della scoperta.

A tutti gli studenti della classe 4A, e a chiunque altro possa trovare questo compendio utile, vi auguriamo buono studio e buona programmazione.

Dedica

Questo compendio è dedicato al nostro stimato professore dell'ITIS Molinari, il Prof. Alemanno Giancarlo.

La sua dedizione all'insegnamento e la sua passione per la programmazione hanno ispirato e guidato la creazione di questo documento.

Il Prof. Alemanno, con la sua profonda comprensione teorica del C++ e la sua capacità di spiegare concetti complessi in modo chiaro e comprensibile, ha fornito le basi su cui questo compendio è costruito.

Attraverso la sua esperienza pratica ha reso vivi questi concetti, dimostrando l'applicazione pratica della programmazione. La sua guida, il suo sostegno e la sua ispirazione sono stati fondamentali per i nostri percorsi di apprendimento.

Vogliamo esprimere la nostra sincera gratitudine al Prof. Alemanno per tutto ciò che ha fatto per noi come insegnante e mentore. Questo compendio è un tributo al suo impegno nel coltivare le nostre menti e nel prepararci per un futuro nella programmazione.

Definizione di una funzione

Una funzione in C++ è un blocco di codice che esegue una specifica operazione. Una funzione è definita specificando il suo tipo di ritorno, il suo nome e i parametri che accetta. Quando una funzione viene definita, il codice della funzione viene caricato nella memoria (RAM) del computer. L'indirizzo di partenza di questo blocco di memoria è determinato dal sistema operativo e dal compilatore, e non è generalmente accessibile al programmatore.

// Definizione di una funzione che somma due numeri interi
int somma(int a, int b) {
return a + b;
}
```

**Dichiarazione di una funzione**

Una dichiarazione di funzione in C++ fornisce le informazioni di base sulla funzione senza fornire il corpo della funzione. Questo viene spesso utilizzato in combinazione con i file di intestazione per separare la definizione e l'implementazione di una funzione. Quando una funzione viene dichiarata, nessun codice viene effettivamente caricato nella memoria.

// Dichiarazione di una funzione
**int somma(int a, int b);**

// Altrove nel codice, o in un altro file, la funzione può essere definita
**int somma(int a, int b) {**
**return a + b;**
**}**
```

Istruzione return

L'istruzione return in C++ termina l'esecuzione di una funzione e restituisce un valore al chiamante. Quando l'istruzione return viene eseguita, il valore di ritorno viene memorizzato in un luogo speciale nella memoria che il chiamante può accedere. Questo spazio è generalmente

piccolo, abbastanza grande per contenere un valore di ritorno, e il suo indirizzo esatto è gestito dal compilatore e dal sistema operativo.

```
// Funzione che ritorna la somma di due numeri
int somma(int a, int b) {
    return a + b;  // L'istruzione return restituisce il risultato al chiamante
}
```

Comunicazioni fra programma chiamante e funzione

Le funzioni in C++ comunicano con il programma chiamante attraverso i parametri e il valore di ritorno. I parametri sono i valori che il chiamante fornisce alla funzione, e il valore di ritorno è il risultato che la funzione fornisce al chiamante. Quando una funzione viene chiamata, i parametri vengono copiati nella memoria. Questo spazio di memoria è chiamato stack e ogni chiamata di funzione ha il suo blocco di memoria nello stack. La dimensione di questo blocco di memoria dipende dal numero e dal tipo di parametri.

```
// Funzione che calcola il prodotto di due numeri
int prodotto(int a, int b) {
    return a * b;  // Il risultato viene restituito al chiamante

}
```

```
// Nel programma chiamante
int x = 5;
int y = 6;
int z = prodotto(x, y);  // Il risultato della funzione viene assegnato a z
```

Argomenti di default

In C++, è possibile specificare valori di default per i parametri di una funzione. Se il chiamante non fornisce un valore per un parametro con un valore di default, viene utilizzato il valore di default. Quando una funzione con argomenti di default viene chiamata, i valori di default vengono caricati nella memoria solo se il chiamante non fornisce un valore. Questi valori di default occupano lo stesso spazio di memoria che i parametri normali.

```
// Funzione con argomenti di default
int somma(int a, int b = 0) {
    return a + b;
}

// Nel programma chiamante
int x = somma(5);  // x sarà 5, perché il secondo parametro ha un valore di default di 0
```

Funzioni con overload

In C++, è possibile definire più funzioni con lo stesso nome ma con diversi parametri. Questo si chiama overload di funzioni. Quando una funzione viene sovraccaricata, ogni versione della funzione ha il proprio blocco di codice nella memoria. Quando la funzione viene chiamata, il

processore decide quale versione della funzione eseguire in base ai parametri forniti. Ogni versione della funzione occupa il proprio spazio nella memoria, e la quantità di memoria utilizzata dipende dalla dimensione del codice della funzione.

```
// Overload di funzioni
int somma(int a, int b) {
    return a + b;
}

double somma(double a, double b) {
    return a + b;
}

// Nel programma chiamante
int x = somma(5, 6);  // x sarà 11, viene chiamata la funzione che accetta interi
double y = somma(5.5, 6.6);  // y sarà 12.1, viene chiamata la funzione che accetta numeri in virgola mobile
```

Le direttive al preprocessore

in C++ sono istruzioni che vengono eseguite dal preprocessore prima che il codice venga effettivamente compilato. Queste direttive non sono istruzioni C++ e non terminano con un punto e virgola. Le direttive al preprocessore iniziano con il simbolo di cancelletto (#).

****Direttiva #include****

La direttiva #include in C++ viene utilizzata per includere un file di intestazione o un file di codice sorgente nel programma. Questo è particolarmente utile per includere librerie standard o personalizzate. Quando il preprocessore incontra una direttiva #include, sostituisce la direttiva con l'intero contenuto del file specificato. Questo significa che il codice del file incluso viene caricato nella memoria del computer insieme al resto del programma. L'indirizzo di partenza di questo blocco di memoria è determinato dal sistema operativo e dal compilatore, e non è generalmente accessibile al programmatore.

Esempio di codice:

#include <iostream> // Include la libreria standard iostream

```

**\*\*Direttiva #define di una costante\*\***

La direttiva #define in C++ viene utilizzata per definire una costante o una macro. Quando il preprocessore
```

incontra una direttiva #define, sostituisce tutte le occorrenze successive del nome della costante con il suo valore. Questo non comporta l'allocazione di memoria in quanto la sostituzione avviene a livello di codice sorgente, prima della compilazione. Tuttavia, il valore della costante può influenzare la quantità di memoria utilizzata dal programma compilato.

Esempio di codice:
#define PI 3.14159 // Definisce una costante PI con valore 3.14159
```
```

Gli indirizzi e i puntatori

sono concetti fondamentali nella programmazione C++. Un indirizzo è un'ubicazione specifica nella memoria del computer, mentre un puntatore è una variabile che contiene l'indirizzo di un'altra variabile.

****Operatore di indirizzo &****

L'operatore di indirizzo (&) in C++ viene utilizzato per ottenere l'indirizzo di memoria di una variabile. Quando si utilizza l'operatore di indirizzo, non viene allocata alcuna memoria aggiuntiva. L'indirizzo restituito è l'indirizzo esistente nella memoria dove la variabile è già memorizzata.

Esempio di codice:

```
int x = 5;
int* p = &x;  // p contiene l'indirizzo di x
```

****Cosa sono i puntatori?****

Un puntatore in C++ è una variabile che contiene l'indirizzo di un'altra variabile. I puntatori sono utilizzati per l'indirezione e per implementare strutture dati come alberi e liste collegate. Quando si dichiara un puntatore, viene allocata una piccola quantità di memoria per memorizzare l'indirizzo. La dimensione esatta dipende dal sistema, ma è tipicamente 4 byte su un sistema a 32 bit e 8 byte su un sistema a 64 bit.

****Dichiarazione di una variabile di tipo puntatore****

Una variabile puntatore viene dichiarata specificando il tipo di variabile a cui punta, seguito da un asterisco (*). Quando si dichiara un puntatore, viene allocata memoria per memorizzare un indirizzo.

Esempio di codice:
```
int* p;  // Dichiarazione di un puntatore a int
```

Assegnazione di un valore a un puntatore
Un valore può essere assegnato a un puntatore utilizzando l'operatore di assegnazione (=). Il valore assegnato deve essere un indirizzo.

Esempio di codice:
```
int x = 5;
int* p = &x;  // Assegna l'indirizzo di x a p
```

Aritmetica dei puntatori
L'aritmetica dei puntatori in C++ consente di aggiungere o sottrarre numeri ai puntatori. Questo è utile per scorrere gli array e altre strutture di dati. Quando si esegue l'aritmetica dei puntatori, la quantità di memoria a cui il puntatore si riferisce cambia, ma non viene allocata o deallocata alcuna memoria aggiuntiva.

Esempio di codice:
```
int arr[5] = {1, 2, 3, 4, 5};
int* p = arr;  // p punta al primo elemento di arr
p++;  // p ora punta al secondo elemento di arr
```

Operatore di dereferenziazione *

L'operatore di dereferenziazione (*) in C++ viene utilizzato per accedere al valore a cui un puntatore punta. Quando si dereferenzia un puntatore, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice

```
int x = 5;
int* p = &x;  // p contiene l'indirizzo di x
int y = *p;  // y ora contiene il valore a cui p punta, cioè 5
```

Funzioni con argomenti puntatori

Le funzioni in C++ possono accettare puntatori come argomenti. Questo è utile per modificare variabili passate come argomenti o per passare array e altre strutture di dati di grandi dimensioni senza copiare l'intero dato. Quando si passa un puntatore a una funzione, viene creata una copia dell'indirizzo del puntatore, ma non dei dati a cui il puntatore punta.

Esempio di codice:

```
void incrementa(int* p) {
    (*p)++;  // Incrementa il valore a cui p punta
}

int x = 5;
incrementa(&x);  // x è ora
```

Gli array e i puntatori

in C++ sono strettamente correlati. Un array è una collezione di elementi dello stesso tipo, e un puntatore è una variabile che contiene l'indirizzo di un'altra variabile. In C++, il nome di un array è un puntatore al primo elemento dell'array.

****Analogia fra puntatori ed array****

In C++, gli array e i puntatori sono strettamente correlati. Il nome di un array è un puntatore al primo elemento dell'array. Questo significa che è possibile utilizzare un puntatore per accedere e manipolare gli elementi di un array. Quando si dichiara un array, viene allocata una quantità di memoria sufficiente a contenere tutti gli elementi dell'array. L'indirizzo di partenza di questo blocco di memoria è l'indirizzo del primo elemento dell'array.

Esempio di codice:

int arr[5] = {1, 2, 3, 4, 5};

int* p = arr; // p punta al primo elemento di arr

```

**\*\*Combinazione fra operazioni di dereferenziazione e di incremento\*\***

In C++, è possibile combinare operazioni di dereferenziazione e incremento per scorrere un array o un'altra struttura di dati. Questo è spesso utilizzato in combinazione con un ciclo per operare su tutti gli elementi di un array.
```

Esempio di codice:

```
int arr[5] = {1, 2, 3, 4, 5};
 int* p = arr;
 for (int i = 0; i < 5; i++) {
    cout << *p << endl;  // Stampa l'elemento corrente
    p++;  // Passa all'elemento successivo
 }
```

****Confronto fra operatore [ ] e dereferenziazione del puntatore “off settato"****

L'operatore [ ] in C++ è un altro modo per accedere agli elementi di un array. L'espressione arr[i] è equivalente a *(arr + i). In entrambi i casi, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
int arr[5] = {1, 2, 3, 4, 5};
 cout << arr[2] << endl;  // Stampa il terzo elemento usando l'operatore [ ]
 cout << *(arr + 2) << endl;  // Stampa il terzo elemento usando la dereferenziazione
```

****Funzioni con argomenti array****

Le funzioni in C++ possono accettare array come argomenti. Quando si passa un array a una funzione, in realtà si sta passando un puntatore al primo elemento dell'array. Questo significa che la funzione può modificare gli elementi dell'array. Quando si passa un array a una funzione, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:
```
void stampa(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << endl;
    }
}

int arr[5] = {1, 2, 3, 4, 5};
stampa(arr, 5);  // Stampa tutti gli elementi
```
di arr
```
```

Funzioni con argomenti puntatori passati by reference

Le funzioni in C++ possono accettare puntatori come argomenti e modificarli. Questo è utile per restituire più di un valore da una funzione o per modificare un puntatore in una funzione. Quando si passa un puntatore a una funzione, viene creata una copia dell'indirizzo del puntatore, ma non dei dati a cui il puntatore punta.

Esempio di codice:
```
void incrementa(int*& p) {
    (*p)++;  // Incrementa il valore a cui p punta
}

int x = 5;
int* p = &x;
incrementa(p);  // x è ora 6
```
```

**Array di puntatori**
```

In C++, è possibile avere un array di puntatori. Questo è utile per creare un array di stringhe o un array di array. Quando si dichiara un array di puntatori, viene allocata una quantità di memoria sufficiente a contenere tutti i puntatori. Ogni puntatore, a sua volta, può puntare a un blocco di memoria separato.

Esempio di codice:

```
int* arr[5];  // Dichiarazione di un array di puntatori a int
for (int i = 0; i < 5; i++) {
    arr[i] = new int(i);  // Ogni puntatore punta a un nuovo int
}
```

Puntatori a costante

Un puntatore a costante in C++ è un puntatore che punta a un valore costante. Questo significa che non è possibile utilizzare il puntatore per modificare il valore a cui punta. Quando si dichiara un puntatore a costante, viene allocata una piccola quantità di memoria per memorizzare l'indirizzo. La dimensione esatta dipende dal sistema, ma è tipicamente 4 byte su un sistema a 32 bit e 8 byte su un sistema a 64 bit.

Esempio di codice:

```
const int x = 5;
const int* p = &x;  // p è un puntatore a un int costante
```

****Funzioni con argomenti costanti trasmessi by value****

Le funzioni in C++ possono accettare argomenti costanti passati per valore. Questo significa che la funzione riceve una copia del valore, e non può modificarlo. Quando si passa un argomento costante per valore a una funzione, viene allocata una quantità di memoria sufficiente a contenere il valore. Questa memoria viene liberata quando la funzione termina.

Esempio di codice:

```
void stampa(const int x) {
    cout << x << endl;  // Stampa x
}
```

****Funzioni con argomenti costanti trasmessi by reference****

Le funzioni in C++ possono accettare argomenti costanti passati per riferimento. Questo significa che la funzione riceve un riferimento al valore originale, e non può modificarlo. Quando si passa un argomento costante per riferimento a una funzione, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
void stampa(const int& x) {
    cout << x << endl;  // Stampa x
}
```

Tipi definiti dall’utente

In C++, è possibile definire tipi di dati personalizzati utilizzando il keyword typedef, le strutture (struct) e le classi (class). Questi tipi di dati definiti dall'utente possono essere utilizzati per creare variabili che possono contenere dati più complessi rispetto ai tipi di dati primitivi.

Concetti di oggetto e istanza

In C++, un oggetto è una particolare istanza di un tipo di dato. Ad esempio, se si definisce una struttura "Persona", ogni persona specifica che si crea nel codice è un oggetto, o un'istanza della struttura "Persona". Quando si crea un oggetto, viene allocata una quantità di memoria sufficiente a contenere tutti i dati dell'oggetto. L'indirizzo di partenza di questo blocco di memoria è l'indirizzo dell'oggetto.

Esempio di codice:

```
struct Persona {
    string nome;
    int eta;
};

Persona p;  // p è un oggetto di tipo Persona
```

Typedef

Il comando typedef in C++ viene utilizzato per creare un alias per un tipo di dato esistente. Questo può rendere il

codice più leggibile e più facile da scrivere. Quando si utilizza typedef, non viene allocata alcuna memoria.

Esempio di codice:
```
typedef int Intero;  // Intero è ora un alias per int
Intero x = 5;  // x è un Intero, che è un int
```

Strutture
Una struttura in C++ è un tipo di dato definito dall'utente che può contenere vari membri con tipi di dati diversi. Quando si dichiara una variabile di tipo struttura, viene allocata una quantità di memoria sufficiente a contenere tutti i membri della struttura.

Esempio di codice:
```
struct Persona {
    string nome;
    int eta;
};

Persona p;  // p è una variabile di tipo Persona
```

Operatore .
L'operatore . in C++ viene utilizzato per accedere ai membri di un oggetto o di una struttura. Quando si utilizza l'operatore ., non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:
```
struct Persona {
    string nome;
```

```
    int eta;
};

Persona p;
p.nome = "Mario";  // Utilizza l'operatore . per accedere al membro nome
```

Puntatori a strutture - Operatore ->

In C++, è possibile avere puntatori a strutture. L'operatore -> viene utilizzato per accedere ai membri di una struttura attraverso un puntatore. Quando si dichiara un puntatore a una struttura, viene allocata una piccola quantità di memoria per memorizzare l'indirizzo.

Esempio di codice:

```
struct Persona {
    string nome;
    int eta;
};

Persona* p = new Persona;  // p è un puntatore a una Persona
p->nome = "Mario";  // Utilizza l'operatore -> per accedere al membro nome
```

Allocazione dinamica della memoria

L'allocazione dinamica della memoria in C++ viene utilizzata per richied

ere memoria a runtime, piuttosto che a tempo di compilazione. Questo è utile quando la quantità di

memoria necessaria non è nota in anticipo. L'operatore new viene utilizzato per allocare memoria, e l'operatore delete viene utilizzato per liberarla. Quando si alloca memoria con new, viene restituito un puntatore all'inizio del blocco di memoria allocato.

Esempio di codice:
```
int* p = new int;  // Alloca memoria per un int
*p = 5;  // Assegna un valore all'int allocato
delete p;  // Libera la memoria allocata
```

Memoria stack e memoria heap

In C++, la memoria stack è utilizzata per le variabili automatiche, mentre la memoria heap è utilizzata per l'allocazione dinamica della memoria. Le variabili stack hanno una durata limitata alla funzione in cui sono dichiarate, mentre la memoria heap rimane allocata fino a quando non viene liberata con l'operatore delete.

Operatore new

L'operatore new in C++ viene utilizzato per allocare memoria dinamicamente. Restituisce un puntatore all'inizio del blocco di memoria allocato. Quando si utilizza new, viene allocata una quantità di memoria sufficiente a contenere il tipo di dato specificato.

Esempio di codice:
```
int* p = new int;  // Alloca memoria per un int
```

Operatore delete

L'operatore delete in C++ viene utilizzato per liberare la memoria che è stata allocata dinamicamente. Quando si utilizza delete, la memoria che era stata allocata viene liberata, rendendola disponibile per future allocazioni.

Esempio di codice:

```
int* p = new int;
delete p;  // Libera la memoria allocata
```

"Potrebbe sembrare tutto semplice, perché ho cercato di spiegare in modo che tutti potessero comprendere. Tuttavia, dietro a questa lezione ci sono anni di studio e dedizione." - Prof. Alemanno Giancarlo

Strutture Dati Dinamiche

Un namespace in C++ è un contenitore che permette di raggruppare entità come classi, oggetti e funzioni sotto un nome unico. L'uso dei namespace è molto utile per evitare conflitti di nomi quando si utilizzano librerie di terze parti. Quando si definisce un namespace, non viene allocata alcuna memoria.

Esempio di codice:

```
namespace MioNamespace {
    int x = 5;
}

int main() {
    cout << MioNamespace::x << endl;  // Accede a x tramite il namespace
}
```

****Programmazione modulare e compilazione separata****

La programmazione modulare è un approccio alla programmazione che enfatizza la suddivisione del codice sorgente in moduli separati, ognuno dei quali è autonomo e indipendente. Questo rende il codice più organizzato, riutilizzabile e facile da gestire. La compilazione separata è un processo in cui ogni modulo viene compilato separatamente, il che può migliorare i tempi di compilazione e facilitare la gestione dei progetti di grandi dimensioni. Quando si compila un modulo, viene allocata memoria per le variabili e le funzioni definite nel modulo.

Definizione di namespace

Un namespace viene definito utilizzando la parola chiave `namespace` seguita dal nome del namespace e un blocco di codice racchiuso tra parentesi graffe. All'interno del blocco di codice, è possibile definire variabili, funzioni, classi e altri namespace.

Esempio di codice:

```
namespace MioNamespace {
    int x = 5;
}
```

Estendibilità della definizione di un namespace

Un namespace in C++ può essere definito in più parti del codice. Questo significa che è possibile aggiungere nuove entità a un namespace esistente semplicemente definendo di nuovo il namespace. Quando si estende un namespace, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
namespace MioNamespace {
    int x = 5;
}

namespace MioNamespace {
    int y = 6;  // Aggiunge y al namespace esistente
}
```

Parola-chiave using

La parola chiave `using` in C++ viene utilizzata per importare un singolo nome da un namespace, o per importare l'intero namespace. Questo può rendere il codice più leggibile eliminando la necessità di qualificare i nomi con il nome del namespace. Quando si utilizza la parola chiave `using`, non viene allocata alcuna memoria.

Esempio di codice:

using namespace std; // Importa tutto il namespace std

using std::cout; // Importa solo cout dal namespace std

```
```

Eccezioni

Eccezioni - Segnalazione e gestione degli errori

Le eccezioni in C++ sono utilizzate per segnalare e gestire errori che si verificano durante l'esecuzione di un programma. Quando si lancia un'eccezione, il flusso di controllo del programma viene interrotto e passa al gestore di eccezioni più vicino. Questo può comportare un cambiamento nella quantità di memoria utilizzata, poiché le variabili locali delle funzioni interrotte possono essere deallocate.

Esempio di codice:

```
try {
    throw "Errore!";  // Lancia un'eccezione
} catch (const char* msg) {
    cerr << msg << endl;  // Gestisce l'eccezione
}
```

Il costrutto try

Il costrutto `try` in C++ viene utilizzato per delimitare un blocco di codice in cui possono verificarsi eccezioni. Non viene allocata alcuna memoria aggiuntiva quando si utilizza il costrutto `try`.

Esempio di codice:

```
try {
    // Codice che può lanciare un'eccezione
}
```

L'istruzione throw

L'istruzione `throw` in C++ viene utilizzata per lanciare un'eccezione. Quando si lancia un'eccezione, il flusso di controllo del programma viene interrotto e passa al gestore di eccezioni più vicino. Non viene allocata alcuna memoria aggiuntiva quando si lancia un'eccezione.

Esempio di codice:

```
throw "Errore!";  // Lancia un'eccezione
```

Il gestore delle eccezioni: costrutto catch

Il costrutto `catch` in C++ viene utilizzato per gestire le eccezioni. Un blocco `catch` segue un blocco `try` e contiene il codice per gestire l'eccezione. Non viene allocata alcuna memoria aggiuntiva quando si utilizza il costrutto `catch`.

Esempio di codice:

```
try {
    // Codice che può lanciare un'eccezione
} catch (const char* msg) {
    cerr << msg << endl;  // Gestisce l'eccezione
}
```

Riconoscimento di un'eccezione fra diverse alternative

In C++, è possibile avere più blocchi `catch` per gestire diversi tipi di eccezioni. Quando si lancia un'eccezione, il primo blocco `catch` che può gestire quel tipo di eccezione viene eseguito. Non viene allocata alcuna memoria aggiuntiva quando si utilizzano più blocchi `catch`.

Esempio di codice:

```
try {
    // Codice che può lanciare un'eccezione
} catch (int e) {
    cerr << "Eccezione di tipo int: " << e << endl;
} catch (const char*

msg) {
    cerr << "Eccezione di tipo const char*: " << msg
<< endl;
}
```

Classi e Programmazione ad Oggetti

La programmazione ad oggetti è un paradigma di programmazione che si basa sull'idea di "oggetti", che possono contenere dati e codice: dati sotto forma di campi (spesso noti come attributi o proprietà), e codice, sotto forma di procedure (spesso note come metodi).

****Analogia fra classi e strutture****

In C++, le classi e le strutture sono molto simili. Entrambe possono avere membri dati e funzioni, e entrambe possono essere utilizzate per creare nuovi tipi di dati. La differenza principale è che, di default, i membri di una struttura sono pubblici, mentre i membri di una classe sono privati. Quando si crea un'istanza di una classe o di una struttura, viene allocata una quantità di memoria sufficiente a contenere tutti i membri dati.

Esempio di codice:

```
struct MiaStruttura {
    int x;
};

class MiaClasse {
public:
    int x;
};
```

Specificatori di accesso

Gli specificatori di accesso in C++ determinano la visibilità dei membri di una classe. Ci sono tre specificatori di accesso: `public`, `private` e `protected`. I membri `public` sono accessibili da ovunque, i membri `private` sono accessibili solo all'interno della classe, e i membri `protected` sono accessibili all'interno della classe e dalle sue sottoclassi.

Esempio di codice:

```
class MiaClasse {
public:
    int x;  // Membro pubblico
private:
    int y;  // Membro privato
};
```

Data hiding

Il data hiding è un concetto fondamentale della programmazione orientata agli oggetti. Si riferisce alla pratica di rendere privati i membri dati di una classe per impedire l'accesso diretto da codice esterno alla classe. Questo aiuta a mantenere l'integrità dei dati e a prevenire errori.

Esempio di codice:

```
class MiaClasse {
private:
    int x;  // x è nascosto al codice esterno alla classe
public:
    void setX(int val) { x = val; }  // Funzione membro per modificare x
```

```
    int getX() { return x; }  // Funzione membro per ottenere il valore di x
};
```

Funzioni membro

Le funzioni membro sono funzioni che sono definite all'interno di una classe. Possono accedere direttamente ai membri dati della classe. Quando si chiama una funzione membro, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class MiaClasse {
private:
    int x;
public:
    void setX(int val) { x = val; }  // setX è una funzione membro
};
```

Risoluzione della visibilità

La risoluzione della visibilità in C++ si riferisce al processo di determinare a quale variabile o funzione si riferisce un nome in un dato contesto. Il compilatore utilizza le regole di scope per determinare questo. Non viene allocata alcuna memoria aggiuntiva durante la risoluzione della visibilità.

Funzioni-membro di sola lettura

In C++, è possibile rendere una funzione membro "di sola lettura" utilizzando il qualificatore `const` dopo la dichiarazione della funzione. Questo indica che la funzione non modificherà nessuno dei membri dati della classe. Quando si chiama una funzione di sola lettura, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class MiaClasse {
private:
    int x;
public:
    int getX() const { return x; }  // getX è una funzione di sola lettura
};
```

Classi membro

In C++, una classe può avere altre classi come membri. Questo è noto come composizione di classi. Quando si crea un'istanza di una classe che ha classi membro, viene allocata una quantità di memoria sufficiente a contenere l'istanza della classe principale e tutte le sue classi membro.

Esempio di codice:

```
class ClasseMembro {
public:
    int y;
};

class MiaClasse {
public:
```

ClasseMembro membro; // membro è un'istanza di ClasseMembro
};
```

**\*\*Puntatore nascosto this\*\***

In C++, all'interno di una funzione membro non statica, è possibile utilizzare il puntatore `this` per riferirsi all'oggetto su cui la funzione è stata chiamata. Il puntatore `this` è un puntatore nascosto che viene passato automaticamente a tutte le funzioni membro non statiche. Non viene allocata alcuna memoria aggiuntiva per il puntatore `this`.

Esempio di codice:
**class MiaClasse {**
**public:**
**int x;**
**void setX(int val) { this->x = val; }** // Utilizza il puntatore this
**};**
```

Costruzione e distruzione di un oggetto

In C++, la costruzione di un oggetto si riferisce al processo di inizializzazione di un oggetto quando viene creato. Questo viene fatto tramite un costruttore, che è una funzione speciale definita all'interno della classe. Allo stesso modo, la distruzione di un oggetto si riferisce al processo di pulizia che avviene quando un oggetto viene rimosso. Questo viene fatto tramite un distruttore. Quando si costruisce un oggetto, viene allocata una quantità di memoria sufficiente a contenere l'oggetto. Quando l'oggetto viene distrutto, questa memoria viene liberata.

Esempio di codice:

```
class MiaClasse {
 public:
    MiaClasse() {  // Costruttore
      // Codice di inizializzazione
    }

    ~MiaClasse() {  // Distruttore
      // Codice di pulizia
    }
};
```

Costruttori

Un costruttore in C++ è una funzione speciale che viene chiamata quando viene creato un oggetto di una classe. Il costruttore ha lo stesso nome della classe e non ha un tipo di ritorno. Può accettare parametri, che possono essere utilizzati per inizializzare i membri dati dell'oggetto. Quando si chiama un costruttore, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class MiaClasse {
 public:
    int x;

    MiaClasse(int val) : x(val) {  // Costruttore con un parametro
    }
};
```

Costruttori e conversione implicita

In C++, un costruttore che accetta un solo parametro può essere utilizzato per la conversione implicita da quel tipo di dato al tipo della classe. Questo significa che è possibile utilizzare un valore di quel tipo di dato dove ci si aspetta un oggetto della classe. Quando si esegue una conversione implicita, viene allocata una quantità di memoria sufficiente a contenere un nuovo oggetto della classe.

Esempio di codice:

```
class MiaClasse {
 public:
    int x;
```

```
    MiaClasse(int val) : x(val) {  // Costruttore con un parametro
    }
};

MiaClasse obj = 5;  // Conversione implicita da int a MiaClasse
```

Distruttori

Un distruttore in C++ è una funzione speciale che viene chiamata quando un oggetto viene rimosso. Il distruttore ha lo stesso nome della classe, preceduto da un tilde (~), e non ha parametri o tipo di ritorno. Quando si chiama un distruttore, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class MiaClasse {
public:
    ~MiaClasse() {  // Distruttore
        // Codice di pulizia


    }
};
```

Oggetti allocati dinamicamente

In C++, è possibile allocare dinamicamente oggetti utilizzando l'operatore `new`. Questo crea l'oggetto nella memoria heap piuttosto che nello stack, e restituisce un puntatore all'oggetto. Quando si alloca un oggetto

dinamicamente, viene allocata una quantità di memoria sufficiente a contenere l'oggetto.

Esempio di codice:
```
class MiaClasse {
public:
    int x;
};

MiaClasse* obj = new MiaClasse();  // Crea un oggetto dinamico
```

Membri puntatori

Una classe in C++ può avere membri puntatori, che sono puntatori a oggetti o valori di altri tipi. Quando si crea un oggetto di una classe che ha membri puntatori, viene allocata una quantità di memoria sufficiente a contenere l'oggetto e i suoi membri puntatori, ma non gli oggetti a cui i puntatori puntano.

Esempio di codice:
```
class MiaClasse {
public:
    int* p;  // Membro puntatore
};
```

Costruttori di copia

Un costruttore di copia in C++ è un tipo speciale di costruttore che inizializza un nuovo oggetto come copia di un oggetto esistente. Quando si chiama un costruttore di

copia, viene allocata una quantità di memoria sufficiente a contenere il nuovo oggetto.

Esempio di codice:

```
class MiaClasse {
 public:
    int x;

    MiaClasse(const MiaClasse& other) : x(other.x) { // Costruttore di copia
    }
};
```

Liste di inizializzazione

In C++, è possibile utilizzare liste di inizializzazione per inizializzare i membri dati di un oggetto al momento della creazione. Questo può essere fatto nel costruttore della classe. Quando si inizializza un oggetto con una lista di inizializzazione, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class MiaClasse {
 public:
    int x, y;

    MiaClasse(int a, int b) : x(a), y(b) { // Lista di inizializzazione
    }
};
```

Membri oggetto

Una classe in C++ può avere altri oggetti come membri. Questo è noto come composizione di classi. Quando si crea un oggetto di una classe che ha membri oggetto, viene allocata una quantità di memoria sufficiente a contenere l'oggetto e tutti i suoi membri oggetto.

Esempio di codice:

```
class ClasseMembro {
public:
    int y;
};

class MiaClasse {
public:
    ClasseMembro membro;  // membro è un oggetto di ClasseMembro
};
```

Array di oggetti

In C++, è possibile creare array di oggetti. Quando si crea un array di oggetti, viene allocata una quantità di memoria sufficiente a contenere tutti gli oggetti nell

'array.

Esempio di codice:

```
class MiaClasse {
public:
    int x;
};
```

MiaClasse array[10]; // Crea un array di 10 oggetti di MiaClasse
```

**Utilità dei costruttori e distruttori**

I costruttori e i distruttori sono fondamentali nella programmazione orientata agli oggetti in C++. I costruttori permettono di inizializzare gli oggetti in modo sicuro e controllato, mentre i distruttori permettono di pulire in modo sicuro quando un oggetto non è più necessario. Questo può includere la liberazione di risorse come la memoria o i file aperti. Quando si chiama un costruttore o un distruttore, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class MiaClasse {
public:
    int* p;

    MiaClasse() {
        p = new int; // Alloca memoria nel costruttore
    }

    ~MiaClasse() {
        delete p; // Libera la memoria nel distruttore
    }
};
```
```

Overload degli operatori

L'overload degli operatori è una caratteristica potente del C++ che permette di ridefinire il modo in cui gli operatori standard (come +, -, *, /, ecc.) funzionano con i tipi di dati definiti dall'utente. Questo può rendere il codice più intuitivo e facile da leggere.

****Estendibilità del C++****

C++ è un linguaggio di programmazione estensibile, il che significa che può essere esteso o modificato per adattarsi a nuovi requisiti o contesti. Una delle caratteristiche che rende C++ estensibile è l'overload degli operatori, che permette di ridefinire il comportamento degli operatori per nuovi tipi di dati. Quando si esegue l'overload di un operatore, non viene allocata alcuna memoria aggiuntiva.

****Ridefinizione degli operatori****

L'overload degli operatori in C++ permette di ridefinire il comportamento degli operatori per nuovi tipi di dati. Questo viene fatto definendo una funzione che implementa l'operazione desiderata e utilizzando la parola chiave `operator` seguita dall'operatore da sovraccaricare.

Esempio di codice:

```
class MiaClasse {
 public:
    int x;

    MiaClasse operator+(const MiaClasse& other) {
```

```
        return MiaClasse(x + other.x);
    }
};
```

Metodi della classe o funzioni esterne?

L'overload degli operatori può essere implementato come metodi di classe o come funzioni esterne. Se l'overload viene implementato come metodo di classe, il primo operando è l'oggetto su cui viene chiamato il metodo. Se l'overload viene implementato come funzione esterna, entrambi gli operandi vengono passati come argomenti.

Esempio di codice:

```
class MiaClasse {
public:
    int x;

    MiaClasse operator+(const MiaClasse& other) {  // Metodo di classe
        return MiaClasse(x + other.x);
    }
};

MiaClasse operator+(const MiaClasse& a, const MiaClasse& b) {  // Funzione esterna
    return MiaClasse(a.x + b.x);
}
```

Il ruolo del puntatore nascosto this

All'interno di un metodo di classe non statico in C++, il puntatore `this` è un puntatore nascosto che punta all'oggetto su cui il metodo è stato chiamato. Quando si esegue l'overload di un operatore come metodo di classe, il puntatore `this` può essere utilizzato per accedere ai membri dati dell'oggetto.

Esempio di codice:

```
class MiaClasse {
 public:
    int x;

    MiaClasse operator+(const MiaClasse& other) {
        return MiaClasse(this->x + other.x);  // Utilizza il puntatore this
    }
};
```

Overload degli operatori di flusso di I/O

In C++, è possibile eseguire l'overload degli operatori di flusso di input/output (`<<` e `>>`) per permettere l'input e l'output di oggetti di classi personalizzate. Questo viene solitamente fatto come funzione esterna, in modo da avere accesso sia all'og

getto della classe che allo stream di I/O come operandi.

Esempio di codice:

```
class MiaClasse {
 public:
    int x;
```

```
    friend std::ostream& operator<<(std::ostream&
os, const MiaClasse& obj);
};

std::ostream& operator<<(std::ostream& os, const
MiaClasse& obj) {
    os << obj.x;
    return os;
}
```

Operatori binari e conversioni

Quando si esegue l'overload di un operatore binario in C++, è possibile che sia necessario convertire uno degli operandi in un tipo diverso. Questo può essere fatto utilizzando un costruttore di conversione, che è un costruttore che accetta un solo argomento di un tipo diverso.

Esempio di codice:

```
class MiaClasse {
public:
    int x;

    MiaClasse(int val) : x(val) {}  // Costruttore di
conversione

    MiaClasse operator+(int val) {
        return MiaClasse(x + val);
    }
};
```

Operatori unari e casting a tipo nativo

L'overload degli operatori unari in C++ funziona in modo simile all'overload degli operatori binari. Inoltre, è possibile eseguire l'overload di operatori di casting per permettere la conversione di un oggetto di una classe personalizzata in un tipo nativo.

Esempio di codice:

```
class MiaClasse {
public:
    int x;

    MiaClasse operator-() {  // Overload dell'operatore unario -
        return MiaClasse(-x);
    }

    operator int() {  // Overload dell'operatore di casting a int
        return x;
    }
};
```

Ereditarietà

L'ereditarietà è uno dei concetti fondamentali della programmazione orientata agli oggetti. Permette di creare una nuova classe che eredita le proprietà e i metodi di una classe esistente. La classe esistente è chiamata classe base, e la nuova classe è chiamata classe derivata.

****L'eredità in C++****

L'ereditarietà è un concetto fondamentale della programmazione orientata agli oggetti che permette di creare una nuova classe basata su una classe esistente. La nuova classe eredita tutti i membri pubblici e protetti della classe base. Quando si crea un oggetto di una classe derivata, viene allocata una quantità di memoria sufficiente a contenere l'oggetto, inclusi i membri dati ereditati dalla classe base.

Esempio di codice:

```
class ClasseBase {
public:
    int x;
};

class ClasseDerivata : public ClasseBase {
    // ClasseDerivata eredita x da ClasseBase
};
```

****Classi base e derivata****

In C++, una classe base è una classe da cui altre classi possono ereditare. Una classe derivata è una classe che eredita da una classe base. Quando si crea un oggetto di una classe derivata, viene allocata una quantità di memoria sufficiente a contenere l'oggetto, inclusi i membri dati ereditati dalla classe base.

Esempio di codice:

```
class ClasseBase {
 public:
    int x;
};

class ClasseDerivata : public ClasseBase {
    // ClasseDerivata eredita x da ClasseBase
};
```

Accesso ai membri della classe base

In C++, una classe derivata può accedere a tutti i membri pubblici e protetti della sua classe base. Non può accedere direttamente ai membri privati della classe base, ma può accedere a essi tramite metodi pubblici o protetti della classe base. Quando si accede a un membro della classe base, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class ClasseBase {
 public:
    int x;
};

class ClasseDerivata : public ClasseBase {
```

```
    public:
        int getX() { return x; }  // Accede al membro x della classe base
    };
```

Conversioni fra classi base e derivata

In C++, è possibile convertire un puntatore o un riferimento a un oggetto di una classe derivata in un puntatore o un riferimento a un oggetto della sua classe base. Questo è noto come upcasting e è sempre sicuro. Il downcasting, ovvero la conversione da un puntatore o un riferimento alla classe base a un puntatore o un riferimento alla classe derivata, è anche possibile, ma può essere sicuro solo se l'oggetto originale è effettivamente un oggetto della classe derivata.

Esempio di codice:

```
    class ClasseBase {
    public:
        int x;
    };

    class ClasseDerivata : public ClasseBase {
        // ClasseDerivata eredita x da ClasseBase
    };

    ClasseDerivata d;
    ClasseBase* b = &d;  // Upcasting
    ClasseDerivata* d2 = static_cast<ClasseDerivata*>(b);  // Downcasting
```

Costruzione della classe base

In C++, quando si crea un oggetto di una classe derivata, il costruttore della classe base viene chiamato prima del costruttore della classe derivata. Questo assicura che tutti i membri dati ereditati dalla classe base siano inizializzati correttamente. Quando si chiama il costruttore della classe base, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class ClasseBase {
public:
    int x;

    ClasseBase(int val) : x(val) {  // Costruttore della classe base
    }
};

class ClasseDerivata : public ClasseBase {
public:
    int y;

    ClasseDerivata(int val1, int val2) : ClasseBase(val1), y(val2) {  // Costruttore della classe derivata
    }
};
```

Funzioni virtuali

In C++, una funzione virtuale è una funzione membro che può essere sovrascritta in una classe derivata. Quando si chiama una funzione virtuale tramite un puntatore o un

riferimento a un oggetto della classe base, viene chiamata la versione della funzione che corrisponde al tipo effettivo dell'oggetto. Quando si dichiara una funzione virtuale, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
class ClasseBase {
public:
    virtual void funzione() {  // Funzione virtuale
        // Implementazione della classe base
    }
};

class ClasseDerivata : public ClasseBase {
public:
    void funzione() override {  // Sovrascrive la funzione virtuale
        // Implementazione della classe derivata
    }
};
```

Tabelle delle funzioni virtuali

In C++, la tabella delle funzioni virtuali (vtable) è una struttura dati che viene utilizzata per supportare il polimorfismo dinamico. Ogni classe con almeno una funzione virtuale ha una vtable associata. Quando si crea un oggetto di una classe con funzioni virtuali, viene allocata una quantità di memoria sufficiente a contenere l'oggetto e il suo puntatore alla vtable.

Esempio di codice:

```
class ClasseBase {
 public:
    virtual void funzione() {  // Funzione virtuale
      // Implementazione della classe base
    }
};

// La vtable per ClasseBase contiene un puntatore alla funzione membro ClasseBase::funzione
```

Classi astratte

In C++, una classe astratta è una classe che ha almeno una funzione membro virtuale pura. Una funzione virtuale pura è una funzione virtuale che è dichiarata nella classe base ma non ha un'implementazione. Non è possibile creare oggetti di una classe astratta, ma è possibile creare puntatori e riferimenti a una classe astratta.

Esempio di codice:

```
class ClasseAstratta {
 public:
    virtual void funzionePura() = 0;  // Funzione virtuale pura
};

// Non è possibile creare oggetti di ClasseAstratta
ClasseAstratta* p;  // Ma è possibile creare puntatori a ClasseAstratta
```

I template

in C++ sono un potente strumento per la programmazione generica, che permette di scrivere codice che può funzionare con vari tipi di dati. Questo può aumentare la riutilizzabilità del codice e aiutare a rendere i programmi più efficienti e sicuri.

****Programmazione generica****

La programmazione generica è un paradigma di programmazione che mira a scrivere codice che può essere riutilizzato per diversi tipi di dati. In C++, i template sono il principale strumento per la programmazione generica. Quando si definisce un template, non viene allocata alcuna memoria. La memoria viene allocata solo quando si crea un'istanza del template.

****Definizione di una classe template****

Una classe template in C++ è una "formula" che può essere utilizzata per creare una famiglia di classi. Quando si definisce una classe template, non viene allocata alcuna memoria. La memoria viene allocata solo quando si crea un'istanza della classe template.

Esempio di codice:

```
template <typename T>
class MiaClasse {
public:
    T x;
};
```

Istanza di un template

Un'istanza di un template in C++ è una versione specifica di una funzione o una classe template per un particolare tipo. Quando si crea un'istanza di un template, viene allocata una quantità di memoria sufficiente a contenere l'oggetto.

Esempio di codice:

```
template <typename T>
class MiaClasse {
public:
    T x;
};

MiaClasse<int> obj;  // Crea un'istanza di MiaClasse per il tipo int
```

Parametri di default

In C++, è possibile specificare parametri di default per i template. Questo significa che se un parametro del template non viene specificato quando si crea un'istanza del template, viene utilizzato il valore di default. Questo non richiede alcuna allocazione di memoria aggiuntiva.

Esempio di codice:

```
template <typename T = int>
class MiaClasse {
public:
    T x;
};
```

MiaClasse<> obj; // Crea un'istanza di MiaClasse con T impostato su int
```

**Funzioni template**

Una funzione template in C++ è una "formula" per creare una famiglia di funzioni. Quando si definisce una funzione template, non viene allocata alcuna memoria. La memoria viene allocata solo quando si chiama la funzione template.

Esempio di codice:

```
template <typename T>
T somma(T a, T b) {
    return a + b;
}
```

Differenze fra funzioni e classi template

Le funzioni template e le classi template in C++ sono simili nel senso che entrambe permettono la programmazione generica. Tuttavia, ci sono alcune differenze chiave. Una funzione template è una "formula" per creare una famiglia di funzioni, mentre una classe template è una "formula" per creare una famiglia di classi. Inoltre, quando si chiama una funzione template, C++ può spesso dedurre il tipo dei parametri del template dai parametri della funzione. Tuttavia, quando si crea un'istanza di una

classe template, è necessario specificare esplicitamente i parametri del template.
```

Esempio di codice:

```
template <typename T>
T somma(T a, T b) {  // Funzione template
    return a + b;
}

somma(1, 2);  // C++ deduce che T è int

template <typename T>
class MiaClasse {  // Classe template
public:
    T x;
};

MiaClasse<int> obj;  // Deve specificare che T è int
```

Generalità sulla Libreria Standard del C++

La Libreria Standard del C++ è una collezione di classi, funzioni, costanti e template predefiniti che forniscono funzionalità comuni, come l'input/output, la manipolazione delle stringhe, la gestione della memoria, e molto altro. Quando si utilizza un elemento della Libreria Standard, non viene allocata alcuna memoria fino a quando non si crea un'istanza di una classe o si chiama una funzione che alloca memoria.

****Campi di applicazione****

La Libreria Standard del C++ può essere utilizzata in una vasta gamma di applicazioni, tra cui lo sviluppo di software di sistema, applicazioni desktop, giochi, applicazioni web, e molto altro. Non viene allocata alcuna memoria specifica per l'utilizzo della Libreria Standard in sé; la memoria viene allocata solo quando si utilizzano specifiche funzioni o classi che richiedono l'allocazione di memoria.

****Header files****

Gli header files in C++ sono file di testo che contengono dichiarazioni di funzioni, classi, costanti e altri elementi che possono essere utilizzati in più file di codice sorgente. Quando si include un header file con la direttiva `#include`, il contenuto dell'header file viene copiato nel file di codice sorgente. Questo non richiede alcuna allocazione di memoria.

Esempio di codice:
#include <iostream> // Include l'header file iostream della Libreria Standard del C++
```

**Il namespace std**
In C++, `std` è il namespace che contiene tutti gli elementi della Libreria Standard. Quando si utilizza un elemento del namespace `std`, è necessario precedere il nome dell'elemento con `std::`, oppure si può utilizzare la direttiva `using namespace std;` per evitare di doverlo fare. L'utilizzo di un namespace non richiede alcuna allocazione di memoria.

Esempio di codice:
**std::cout << "Ciao, mondo!";** // Utilizza l'elemento cout del namespace std

**using namespace std;**
**cout << "Ciao, mondo!";** // Ora è possibile utilizzare cout senza precederlo con std::
```

La Standard Template Library (STL)

La STL è una parte della Libreria Standard del C++ che fornisce un insieme di template di classi per gestire gruppi di dati di tipo omogeneo. Questi gruppi di dati sono chiamati contenitori e includono strutture come vettori, liste, code, pile e mappe. Quando si crea un'istanza di un contenitore STL, viene allocata una quantità di memoria sufficiente a contenere gli elementi del contenitore.

****Iteratori****

Gli iteratori in C++ sono oggetti che possono essere utilizzati per accedere agli elementi di un contenitore. Gli iteratori funzionano in modo simile ai puntatori, ma sono più sicuri e più facili da usare. Quando si crea un iteratore, viene allocata una piccola quantità di memoria per l'iteratore stesso.

Esempio di codice:

```
#include <vector>

std::vector<int> v = {1, 2, 3, 4, 5};
for (std::vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
    std::cout << *it << std::endl;
}
```

****Contenitori Standard****

I contenitori standard in C++ sono classi template che forniscono strutture di dati per memorizzare gruppi di dati di tipo omogeneo. Quando si crea un'istanza di un contenitore standard, viene allocata una quantità di memoria sufficiente a contenere gli elementi del contenitore.

Esempio di codice:
#include <vector>

std::vector<int> v = {1, 2, 3, 4, 5}; // Crea un vettore di interi
```

**Algoritmi e oggetti-funzione**
La STL fornisce una serie di algoritmi che possono essere utilizzati con i contenitori per eseguire operazioni comuni, come la ricerca, l'ordinamento e la trasformazione degli elementi. Un oggetto-funzione, o funtore, è un oggetto che può essere chiamato come se fosse una funzione. Quando si chiama un algoritmo STL o si crea un oggetto-funzione, viene allocata una piccola quantità di memoria per l'algoritmo o l'oggetto-funzione stesso.

Esempio di codice:
**#include <vector>**
**#include <algorithm>**

**std::vector<int> v = {1, 2, 3, 4, 5};**
**std::sort(v.begin(), v.end());** // Ordina gli elementi del vettore
```

Le classi `string` e `vector` sono due delle classi più utilizzate nella libreria standard del C++.

****La classe string****

La classe `string` in C++ è un contenitore standard che può essere utilizzato per memorizzare e manipolare stringhe di caratteri. Quando si crea un oggetto `string`, viene allocata una quantità di memoria sufficiente a contenere i caratteri della stringa.

Esempio di codice:

```
#include <string>

std::string s = "Ciao, mondo!";  // Crea una stringa
```

****Confronto fra string e vector<char>****

La classe `string` e il contenitore `vector<char>` possono entrambi essere utilizzati per memorizzare stringhe di caratteri in C++. Tuttavia, la classe `string` fornisce molte funzioni utili per la manipolazione delle stringhe che non sono disponibili con `vector<char>`. Quando si crea un `string` o un `vector<char>`, viene allocata una quantità di memoria sufficiente a contenere i caratteri.

Esempio di codice:

```
#include <string>
```

```
#include <vector>

std::string s = "Ciao, mondo!";  // Crea una stringa
std::vector<char> v = {'C', 'i', 'a', 'o', ',', ' ', 'm', 'o', 'n', 'd', 'o', '!'};  // Crea un vettore di caratteri
```

Confronti fra stringhe

In C++, è possibile utilizzare gli operatori di confronto standard (`==`, `!=`, `<`, `<=`, `>`, `>=`) per confrontare le stringhe. Quando si confrontano le stringhe, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
#include <string>

std::string s1 = "Ciao";
std::string s2 = "Mondo";

if (s1 == s2) {
    // ...
}
```

Concatenazioni e inserimenti

In C++, è possibile utilizzare l'operatore `+` per concatenare le stringhe e l'operatore `+=` per inserire una stringa in un'altra. Quando si concatenano o si inseriscono stringhe, viene allocata una nuova quantità di memoria per contenere la stringa risultante.

Esempio di codice:

```
#include <string>
```

```
std::string s1 = "Ciao";
std::string s2 = "Mondo";

std::string s3 = s1 + ", " + s2;  // Concatena le stringhe
s1 += ", " + s2;  // Inserisce s2 in s1
```

Ricerca di sotto-stringhe

La classe `string` in C++ fornisce la funzione `find` per cercare una sotto-stringa all'interno di una stringa. Quando si cerca una sotto-stringa, non viene allocata alcuna memoria aggiuntiva.

Esempio di codice:

```
#include <string>

std::string s = "Ciao, mondo!";
size_t pos =

s.find("mondo");  // Cerca la sotto-stringa "mondo"
```

Estrazione e sostituzione di sotto-stringhe

La classe `string` in C++ fornisce le funzioni `substr` per estrarre una sotto-stringa da una stringa e `replace` per sostituire una sotto-stringa con un'altra. Quando si estrae o si sostituisce una sotto-stringa, viene allocata una nuova quantità di memoria per contenere la stringa risultante.

Esempio di codice:

```
#include <string>
```

```
std::string s = "Ciao, mondo!";
std::string sub = s.substr(6, 5);  // Estrae la sotto-stringa "mondo"

s.replace(6, 5, "pianeta");  // Sostituisce "mondo" con "pianeta"
```

"Non mettere troppa carne al fuoco, altrimenti finirai per fare solo 'cose' senza cogliere il significato profondo di ciò che stai facendo." - Prof. Alemanno Giancarlo

Il polimorfismo

è uno dei quattro principi fondamentali della programmazione orientata agli oggetti (OOP), insieme all'incapsulamento, all'ereditarietà e all'astrazione. Il termine "polimorfismo" deriva dal greco e significa "molte forme". In termini di programmazione, si riferisce alla capacità di un'entità, come una variabile, una funzione o un oggetto, di assumere molte forme.

In C++, il polimorfismo si realizza principalmente attraverso l'uso di funzioni virtuali e classi astratte. Una funzione virtuale è una funzione membro che viene dichiarata nella classe base e ridefinita (ovvero sovrascritta) da una o più classi derivate. Una classe astratta è una classe che contiene almeno una funzione virtuale pura (una funzione virtuale che è dichiarata ma non implementata nella classe base).

Quando si utilizza il polimorfismo in C++, la memoria viene allocata per l'oggetto della classe base o derivata che si sta utilizzando. L'indirizzo di partenza di questa memoria dipende da come il sistema operativo gestisce la memoria per il tuo programma.

Ecco un esempio di codice che illustra il polimorfismo in C++:

```
#include <iostream>

// Classe base astratta
```

```
class Forma {
public:
    virtual void disegna() = 0; // Funzione virtuale pura
};

// Classe derivata
class Cerchio : public Forma {
public:
    void disegna() override {
        std::cout << "Disegno un cerchio." << std::endl;
    }
};

// Classe derivata
class Rettangolo : public Forma {
public:
    void disegna() override {
        std::cout << "Disegno un rettangolo." <<
std::endl;
    }
};

int main() {
    // Creazione di oggetti
    Cerchio cerchio;
    Rettangolo rettangolo;

    // Puntatore alla classe base
    Forma* forma;

    // Polimorfismo in azione
    forma = &cerchio;
    forma->disegna(); // Outputs: "Disegno un cerchio."
```

```
    forma = &rettangolo;
    forma->disegna(); // Outputs: "Disegno un rettangolo."

    return 0;
}
```

In questo esempio, la funzione `disegna` è dichiarata come una funzione virtuale pura nella classe base `Forma` e quindi viene sovrascritta nelle classi derivate `Cerchio` e `Rettangolo`. Quando la funzione `disegna` viene chiamata su un oggetto della classe base `Forma` che punta a un oggetto della classe derivata, la versione sovrascritta della funzione viene chiamata. Questo è il polimorfismo in azione.

Introduzione alla programmazione Client/Server

La programmazione è un processo creativo che istruisce un computer su come eseguire un compito. È il modo in cui gli umani comunicano con le macchine per eseguire funzioni specifiche. Questo viene fatto utilizzando linguaggi di programmazione, che sono un insieme di istruzioni sintattiche e semantiche utilizzate per definire vari tipi di output. Ci sono molti linguaggi di programmazione, tra cui C++, Java, Python, ecc., ognuno con le sue particolari caratteristiche e utilizzi.

****Client/Server****

Il modello client/server è un paradigma di calcolo distribuito in cui le funzioni del sistema sono suddivise tra server, che forniscono risorse o servizi, e client, che accedono e utilizzano queste risorse. I server ospitano e gestiscono le risorse, mentre i client inviano richieste ai server per accedere o utilizzare le risorse. Ad esempio, un server web ospita pagine web e un client web (browser) richiede queste pagine dal server per visualizzarle all'utente.

****Concetto di Applicazione Client****

Un'applicazione client è un software che viene eseguito sul dispositivo dell'utente finale e interagisce con il server per accedere a determinate funzionalità o servizi. Queste applicazioni inviano richieste al server e ricevono risposte. Ad esempio, un browser web è un'applicazione client che invia richieste a un server web per recuperare e visualizzare pagine web.

****Concetto di Applicazione Server****

Un'applicazione server è un software che riceve richieste dall'applicazione client, elabora queste richieste e restituisce i risultati al client. Le applicazioni server gestiscono la logica di business e l'accesso ai dati di base dell'applicazione. Ad esempio, un server web è un'applicazione server che elabora le richieste HTTP dai client web, recupera le pagine web richieste e le invia al client.

****Applicazioni WEB****

Le applicazioni web sono programmi software che utilizzano tecnologie web per eseguire compiti su Internet. Sono accessibili tramite un browser web e non richiedono l'installazione sul dispositivo dell'utente. Le applicazioni web possono includere qualsiasi cosa, dai negozi online ai servizi di posta elettronica, dai sistemi di gestione dei contenuti ai giochi online.

****HTTP un esempio di protocollo di rete****

HTTP (Hypertext Transfer Protocol) è un protocollo di rete utilizzato per trasferire dati su Internet. È il protocollo di base per la trasmissione di documenti ipertestuali, come le pagine web. Quando un utente accede a una pagina web, il browser invia una richiesta HTTP al server web. Il server elabora la richiesta e risponde con i dati della pagina web, che vengono inviati al browser dell'utente tramite HTTP. HTTP è un protocollo senza stato, il che significa che ogni richiesta è indipendente dalle precedenti.

Fondamenti di HTML

HTML (HyperText Markup Language) è il linguaggio di markup standard per la creazione di pagine web. È il blocco di costruzione più fondamentale del web.

****Fondamenti di HTML****

HTML utilizza "tag" per definire gli elementi all'interno di una pagina web. Questi tag indicano al browser come visualizzare il contenuto. Ad esempio, il tag `<h1>` viene utilizzato per il titolo principale, mentre `<p>` è utilizzato per un paragrafo di testo.

****Struttura di una pagina Web****

Una pagina web HTML di base ha una struttura specifica. Inizia con un tipo di documento (`<!DOCTYPE html>`) e ha elementi radice come `<html>`, `<head>` e `<body>`. L'elemento `<head>` contiene metadati sulla pagina, mentre `<body>` contiene il contenuto principale che viene visualizzato ai visitatori del sito web.

Esempio di codice:

```html
<!DOCTYPE html>
<html>
<head>
<title>Titolo della pagina</title>
</head>
<body>
<h1>Il mio primo titolo</h1>
<p>Il mio primo paragrafo.</p>
</body>
```

</**html**>
```

**Paragrafi**
In HTML, i paragrafi sono definiti dal tag `<p>`. Tutto ciò che è compreso tra l'apertura `<p>` e la chiusura `</p>` viene visualizzato come un paragrafo sul sito web.

Esempio di codice:
```html
<**p**>**Questo è un esempio di paragrafo.**</**p**>
```

Liste
HTML supporta vari tipi di liste, tra cui liste ordinate (`<ol>`), liste non ordinate (`<ul>`) e liste di definizione (`<dl>`). Gli elementi della lista sono definiti dal tag `<li>`.

Esempio di codice:
```html
<**ul**>
<**li**>**Elemento 1**</**li**>
<**li**>**Elemento 2**</**li**>
<**li**>**Elemento 3**</**li**>
</**ul**>
```

Tabelle
Le tabelle in HTML sono definite utilizzando il tag `<table>`, con righe di tabella (`<tr>`) e celle di tabella (`<td>` o `<th>` per le celle dell'intestazione).

Esempio di codice:
```

```html
<table>
<tr>
<th>Intestazione 1</th>
<th>Intestazione 2</th>
</tr>
<tr>
<td>Riga 1, Cellula 1</td>
<td>Riga 1, Cellula 2</td>
</tr>
</table>
```

Link

I link in HTML sono definiti dal tag `<a>`. L'attributo `href` specifica l'URL della pagina a cui il link dovrebbe puntare.

Esempio di codice:

```html
<a href="https://www.example.com">Questo è un link</a>
```

Header

HTML fornisce

sei tag di intestazione, da `<h1>` a `<h6>`. `<h1>` è utilizzato per l'intestazione principale, mentre `<h6>` è l'intestazione di livello più basso.

Esempio di codice:

```html
```

```
<h1>Intestazione 1</h1>
<h2>Intestazione 2</h2>
<h3>Intestazione 3</h3>
<h4>Intestazione 4</h4>
<h5>Intestazione 5</h5>
<h6>Intestazione 6</h6>
```

Form

I form HTML sono utilizzati per raccogliere input dall'utente. Un form è definito dal tag `<form>`. All'interno di un form, è possibile utilizzare diversi tipi di elementi di input, come campi di testo (`<input type="text">`), caselle di controllo (`<input type="checkbox">`), pulsanti radio (`<input type="radio">`), e pulsanti di invio (`<input type="submit">`).

Esempio di codice:

```html
<form action="/submit_form" method="post">
<label for="name">Nome:</label><br>
<input type="text" id="name" name="name"><br>
<input type="submit" value="Invia">
</form>
```

Nel contesto della RAM e del processore, quando un browser web interpreta il codice HTML per visualizzare una pagina web, il codice viene letto dal disco rigido, caricato nella RAM e quindi eseguito dal processore. La quantità di memoria RAM utilizzata dipende dalla complessità del codice HTML. Ad esempio, una pagina web con molti

elementi e contenuti potrebbe richiedere più RAM rispetto a una pagina web più semplice.

Fondamenti di PHP

PHP (Hypertext Preprocessor) è un linguaggio di scripting lato server utilizzato principalmente per lo sviluppo web. È incorporato in HTML e può essere utilizzato per creare contenuti dinamici, gestire dati di form, tracciare sessioni utente e costruire siti web interattivi.

****Confronto C++/PHP****

C++ è un linguaggio di programmazione di livello medio utilizzato per lo sviluppo di software di sistema, driver, server e giochi. PHP, d'altra parte, è un linguaggio di scripting lato server utilizzato per lo sviluppo web. Mentre C++ è più potente e offre un controllo più granulare sul sistema, PHP è più facile da usare per lo sviluppo web grazie alla sua sintassi semplice e alle sue funzionalità integrate per la gestione dei database e delle sessioni HTTP.

****Installazione XAMPP****

XAMPP è un pacchetto software gratuito che include Apache, MySQL, PHP e Perl. È utilizzato per creare un ambiente di sviluppo web locale. L'installazione di XAMPP varia a seconda del sistema operativo, ma generalmente include il download del pacchetto software da https://www.apachefriends.org/index.html e l'esecuzione dell'installatore.

****Sintassi PHP****

La sintassi PHP è simile a quella di altri linguaggi di programmazione C-style. Un script PHP inizia con `<?php`

e termina con `?>`. Ogni istruzione PHP termina con un punto e virgola (;).

Esempio di codice:
```php
<?php
echo "Ciao, mondo!";
?>
```

Variabili PHP
In PHP, le variabili iniziano con un segno di dollaro ($). PHP è un linguaggio non tipizzato, il che significa che non è necessario dichiarare il tipo di una variabile quando la si crea.

Esempio di codice:
```php
<?php
$txt = "Ciao, mondo!";
$x = 5;
$y = 10.5;
?>
```

devo scrivere un libro con molti argomenti riguardante la programmazione, ora ti passero un po di argomenti ,per ogni argomento scrivi spiegazione esaustiva e fai un esempio di codice commentato `echo` e `print` sono due modi per output di testo in PHP. `echo` ha un po' di prestazioni migliori, ma la differenza è così piccola che non è generalmente un fattore decisivo. `echo` può prendere più parametri, mentre `print` può prendere uno solo.

Esempio di codice:

```php
<?php
echo "Ciao, mondo!";
print "Ciao, mondo!";
?>
```

Tipi di dati PHP

PHP supporta otto tipi di dati primitivi: quattro tipi scalari (boolean, integer, float e string), due tipi composti (array e object) e due tipi speciali (resource e NULL).

8. **Stringhe PHP**

Una stringa è una sequenza di caratteri. In PHP, una stringa può essere definita utilizzando singoli apici o doppi apici.

Esempio di codice:

```php
<?php
$x = "Ciao, mondo!";
$y = 'Ciao, mondo!';
echo $x;
echo $y;
?>
```

Numeri PHP

PHP supporta due tipi di numeri: interi e float (o numeri a virgola mobile). Un intero è un numero senza una virgola o una parte decimale. Un float è un numero con una virgola o una parte decimale.

Esempio di codice:
```php
<?php
$x = 5985; // un intero
$y = 59.85; // un float
echo $x;
echo $y;
?>
```

PHP Matematica: FUNZIONE RAND()

La funzione `rand()` in PHP genera un numero casuale. Puoi passare due argomenti a `rand()`: il numero minimo e il numero massimo del range.

Esempio di codice:
```php
<?php
echo rand(10, 100);
?>
```

Costanti PHP

Una costante è un identificatore (nome) per un valore unico. Come suggerisce il nome, il valore non può cambiare durante lo script. Una costante è case-sensitive per impostazione predefinita. Per definire una costante, è necessario utilizzare la funzione `define()`.

Esempio di codice:

```php
<?php
define("GREETING", "Benvenuto su W3Schools.com!");
echo GREETING;
?>
```

Operatori PHP

PHP divide gli operatori nelle seguenti categorie: operatori aritmetici, operatori di assegnazione, operatori di confronto, operatori di incremento/decremento, operatori logici, operatori di stringa, operatori di array.

Costrutti di Selezione

PHP supporta diversi costrutti di selezione, tra cui `if`, `else`, `elseif/else if`, `switch`.

Costrutti Iterativi

PHP supporta diversi costrutti iterativi, tra cui `while`, `do...while`, `for`, `foreach`.

Funzioni PHP

Una funzione è un blocco di codice riutilizzabile. PHP ha molte funzioni integrate, ma è anche possibile creare le proprie funzioni.

Array PHP

Un array in PHP è una variabile speciale, che può contenere più di un valore alla volta. Ci sono tre tipi di array in PHP: indicizzati, associativi e multidimensionali.

Concetto di Array Associativi

Gli array associativi sono array che utilizzano chiavi nominate che assegnate ai loro valori. Questo contrasta con gli array indicizzati, che utilizzano indici numerici.

Nel contesto della RAM e del processore, quando un script PHP viene eseguito, il codice viene letto dal disco rigido, caricato nella RAM e quindi eseguito dal processore. La quantità di memoria RAM utilizzata dipende dalla complessità dello script PHP. Ad esempio, uno script che gestisce grandi quantità di dati potrebbe richiedere più RAM rispetto a uno script più semplice.

Gli Array Associativi Super globali PHP

In PHP, le variabili superglobali sono variabili incorporate che sono sempre disponibili in tutti gli ambiti. Queste variabili superglobali sono di tipo array associativo.

Ecco alcuni esempi:

$_GET

`$_GET` è un array associativo di variabili passate allo script corrente tramite i parametri URL. È utilizzato per raccogliere dati di form inviati con il metodo GET.

Esempio di codice:

```php
<?php
echo 'Ciao, ' . htmlspecialchars($_GET["name"]) . '!';
?>
```

In questo esempio, il valore del parametro "name" nell'URL viene stampato. Ad esempio, se l'URL è "script.php?name=John", verrà stampato "Ciao, John!".

$_POST

`$_POST` è un array associativo di variabili passate allo script corrente tramite il metodo HTTP POST. È utilizzato per raccogliere dati di form inviati con il metodo POST.

Esempio di codice:

```php
```

```
<?php
echo 'Ciao, ' . htmlspecialchars($_POST["name"]) . "!";
?>
```

In questo esempio, il valore del campo "name" nel form inviato viene stampato.

$_COOKIE

`$_COOKIE` è un array associativo di variabili passate allo script corrente tramite cookie HTTP.

Esempio di codice:

```php
<?php
echo 'Ciao, ' . htmlspecialchars($_COOKIE["name"]) . "!";
?>
```

In questo esempio, il valore del cookie "name" viene stampato.

$_SESSION

`$_SESSION` è un array associativo contenente le variabili di sessione disponibili per lo script corrente.

Esempio di codice:

```php
<?php
session_start();
$_SESSION["favcolor"] = "verde";
echo 'Il tuo colore preferito è ' . $_SESSION["favcolor"];
```

?>
```

In questo esempio, viene impostata una variabile di sessione "favcolor" e quindi stampata.

**$_SERVER**

`$_SERVER` è un array contenente informazioni come intestazioni, percorsi e posizioni dello script. Le voci in questo array sono create dal server web.

Esempio di codice:
```php
<**?php**
**echo $_SERVER['SERVER_NAME'];**
**?>**
```
In questo esempio, viene stampato il nome del server.
```

"Ogni lezione è come un post-it, non dovresti ritardare troppo a mettere in pratica ciò che hai appreso, perché prima o poi il post-it potrebbe volare via." - Prof. Alemanno Giancarlo

LABORATORIO E LAVORI PIU' RILEVANTI SVOLTI E PROPOSTI

Costruzione della Classe Frazione e creazione di un nuovo tipo di variabile

In C++, è possibile creare un nuovo tipo di variabile definendo una classe. Ad esempio, si può creare una classe "Frazione" per rappresentare una frazione matematica.

```
class Frazione {
    private:
        int numeratore;
        int denominatore;
    public:
        Frazione(int num, int den) :
numeratore(num), denominatore(den) {}
        int getNumeratore() { return numeratore; }
        int getDenominatore() { return
denominatore; }
};
```

In questo esempio, la classe "Frazione" ha due membri privati: numeratore e denominatore. Ha un costruttore che accetta due argomenti e due metodi pubblici per ottenere il numeratore e il denominatore.

Quando si crea un'istanza di "Frazione", viene allocata una certa quantità di memoria nella RAM per conservare i valori del numeratore e del denominatore. L'indirizzo di

partenza di questa memoria dipende da dove il sistema operativo decide di posizionare l'oggetto nella RAM.

Uso del Puntatore THIS

In C++, `this` è un puntatore che punta all'oggetto corrente. È utilizzato per accedere ai membri dell'oggetto corrente.

```
class Frazione {
    private:
        int numeratore;
        int denominatore;
    public:
        Frazione(int num, int den) :
numeratore(num), denominatore(den) {}
        Frazione& setNumeratore(int num) { this-
>numeratore = num; return *this; }
        Frazione& setDenominatore(int den) { this-
>denominatore = den; return *this; }
};
```

In questo esempio, i metodi `setNumeratore` e `setDenominatore` utilizzano il puntatore `this` per accedere ai membri dell'oggetto corrente. Ritornano un riferimento all'oggetto corrente, permettendo le chiamate in cascata.

Applicazione del principio di sovrapposizione delle funzioni

In C++, è possibile avere più funzioni con lo stesso nome ma con diversi parametri. Questo è noto come sovraccarico delle funzioni.

```
    class Frazione {
        private:
            int numeratore;
            int denominatore;
        public:
            Frazione(int num, int den) :
numeratore(num), denominatore(den) {}
            Frazione(int num) : numeratore(num),
denominatore(1) {}
    };
    ```
```

In questo esempio, la classe "Frazione" ha due costruttori: uno che accetta due argomenti e uno che ne accetta solo uno. Questo è un esempio di sovraccarico delle funzioni.

**Ridefinizione degli operatori**

In C++, è possibile ridefinire la maggior parte degli operatori incorporati in modo che funzionino con gli oggetti delle classi definite dall'utente. Questo è noto come sovraccarico degli operatori.

```
    class Frazione {
        private:
            int numeratore;
            int denominatore;
        public:
            Frazione(int num, int den) :
numeratore(num), denominatore(den) {}
```
```

Costruzione della classe vettore

In C++, la classe `vector` è una delle classi di contenitori più utilizzate. Permette di gestire array di dati in modo dinamico, ovvero la sua dimensione può cambiare durante l'esecuzione del programma.

#include <vector>

int main() {
std::vector<int> v; // Crea un vettore vuoto di interi
v.push_back(1); // Aggiunge un elemento alla fine del vettore
v.push_back(2);
v.push_back(3);
return 0;
}
```

In questo esempio, viene creato un vettore di interi vuoto. Poi, vengono aggiunti tre elementi al vettore utilizzando il metodo `push_back`.

**\*\*Puntatori\*\***

Un puntatore è una variabile che contiene l'indirizzo di un'altra variabile. In C++, i puntatori sono utilizzati per molte operazioni, come la gestione dinamica della memoria, la manipolazione di array e la creazione di strutture dati complesse.
```

```
int main() {
    int x = 10;
    int* p = &x; // p è un puntatore all'intero x
    return 0;
}
```

In questo esempio, `p` è un puntatore all'intero `x`. L'operatore `&` viene utilizzato per ottenere l'indirizzo di `x`.

Aritmetica dei Puntatori

L'aritmetica dei puntatori in C++ permette di manipolare i puntatori utilizzando gli operatori aritmetici. Questo è particolarmente utile quando si lavora con array.

```
int main() {
    int array[3] = {1, 2, 3};
    int* p = array; // p punta al primo elemento dell'array
    p++;          // p ora punta al secondo elemento dell'array
    return 0;
}
```

In questo esempio, `p` è un puntatore al primo elemento dell'array. L'operatore `++` viene utilizzato per far puntare `p` al secondo elemento dell'array.

Liste

In C++, le liste sono strutture dati che contengono una sequenza di elementi. A differenza degli array e dei vettori,

le liste permettono l'inserimento e la rimozione di elementi in qualsiasi posizione in tempo costante.

```
#include <list>

int main() {
    std::list<int> l; // Crea una lista vuota di interi
    l.push_back(1);   // Aggiunge un elemento alla fine della lista
    l.push_back(2);
    l.push_back(3);
    return 0;
}
```

In questo esempio, viene creata una lista di interi vuota. Poi, vengono aggiunti tre elementi alla lista utilizzando il metodo `push_back`.

Classe e Funzioni Template

Le classi e le funzioni template in C++ permettono di creare codice che può operare su diversi tipi di dati. Questo è un concetto fondamentale della programmazione generica, che mira a scrivere codice che può essere riutilizzato in diversi contesti, riducendo la duplicazione del codice.

Un esempio di funzione template è una funzione `max` che può operare su qualsiasi tipo di dati per cui l'operatore di confronto `>` è definito. Ecco un esempio:

```
template <typename T>
T max(T a, T b) {
```

```
    return (a > b) ? a : b;
}

int main() {
    int max_i = max(1, 2); // Uso della funzione template con interi
    double max_d = max(1.2, 2.3); // Uso della funzione template con numeri in virgola mobile
    return 0;
}
```

In questo esempio, la funzione `max` è definita come una funzione template che accetta due argomenti di un tipo generico `T` e restituisce il maggiore dei due. La funzione `max` può quindi essere utilizzata con qualsiasi tipo di dati per cui l'operatore `>` è definito, come dimostrato nel `main` dove viene utilizzata sia con interi che con numeri in virgola mobile.

“Io alla mi età sto ancora studiando, tu cosa pensi di fare da grande?” - Prof. Alemanno Giancarlo

Costruzione della classe Stack

La classe Stack è un tipo di contenitore adattatore con un comportamento di tipo LIFO (Last In First Out), dove un nuovo elemento viene aggiunto da un'estremità (in cima) e un elemento viene rimosso da quella stessa estremità. Stack utilizza un oggetto incapsulato di vector, deque (per impostazione predefinita) o list come suo contenitore sottostante, fornendo un insieme specifico di funzioni membro per accedere ai suoi elementi.

Ecco un esempio di come si può implementare e utilizzare una stack in C++:

```
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<int> stack;
    stack.push(21);
    stack.push(22);
    stack.push(24);
    stack.push(25);
    int num = 0;
    stack.push(num);
    stack.pop();
```

```
    stack.pop();
    stack.pop();
    while (!stack.empty()) {
        cout << stack.top() << " ";
        stack.pop();
    }
}
```

In questo esempio, abbiamo creato una stack di interi e abbiamo inserito alcuni valori utilizzando la funzione `push()`. Successivamente, abbiamo rimosso alcuni elementi dalla cima della stack utilizzando la funzione `pop()`. Infine, abbiamo stampato tutti gli elementi rimanenti nella stack.

Complessità temporale e spaziale

La complessità temporale di questo programma è O(N), dove N è il numero totale di elementi nella stack. Il ciclo while itera N volte, rimuovendo gli elementi dalla stack e stampandoli.

La complessità spaziale di questo programma è O(N), dove N è il numero totale di elementi nella stack. La struttura dati stack utilizza spazio proporzionale al numero di elementi memorizzati in essa. In questo caso, la dimensione massima della stack è 5, quindi la complessità spaziale è costante e può essere considerata come O(1).

Concetti chiave

Puntatori

I puntatori sono uno degli aspetti più distintivi e potenti del C++. Un puntatore è una variabile che contiene l'indirizzo di un'altra variabile. Questo significa che un puntatore "punta" a un'area di memoria in cui i dati sono memorizzati. I puntatori sono utilizzati in C++ per gestire dinamicamente la memoria, per supportare strutture dati complesse come le liste collegate e gli alberi, e per costruire strutture dati dinamiche come le stack e le code.

Ecco un esempio di come si utilizzano i puntatori in C++:

int x = 10;
int* p = &x;
cout << *p; // Stampa il valore di x (10)
```

In questo esempio, `p` è un puntatore a un intero. L'operatore `&` viene utilizzato per ottenere l'indirizzo di `x`, e l'operatore `*` viene utilizzato per dereferenziare il puntatore `p`, cioè per accedere al valore a cui `p` punta.

**\*\*Array a dimensione variabile\*\***

In C++, gli array sono strutture dati statiche, il che significa che la loro dimensione deve essere nota al momento della compilazione. Tuttavia, è possibile simulare il comportamento di un array a dimensione variabile utilizzando la memoria dinamica e i puntatori. Questo è particolarmente utile quando si lavora con strutture dati come le stack, dove il numero di elementi può cambiare durante l'esecuzione del programma.
```

Ecco un esempio di come si può creare un array a dimensione variabile in C++:

```
int* arr = new int[10];  // Crea un array di 10 interi
delete[] arr;  // Libera la memoria allocata per l'array
arr = new int[20];  // Crea un nuovo array di 20 interi
```

In questo esempio, `arr` è un puntatore a un intero che viene utilizzato per creare un array di interi. L'operatore `new` viene utilizzato per allocare memoria per l'array, e l'operatore `delete[]` viene utilizzato per liberare la memoria quando l'array non è più necessario.

Liste

Le liste sono un tipo di struttura dati lineare che consiste in una serie di nodi, ognuno dei quali contiene un valore e un puntatore al nodo successivo nella lista. Le liste possono essere utilizzate per implementare altre strutture dati come stack e code, e sono particolarmente utili quando il numero di elementi può cambiare durante l'esecuzione del programma.

Ecco un esempio di come si può creare una lista in C++:

```
#include <list>

std::list<int> lst;
lst.push_back(1);
lst.push_back(2);
lst.push_back(3);
```

```
for (int x : lst) {
    std::cout << x << " ";
}
```

In questo esempio, `lst` è un oggetto della classe `std::list` che contiene interi. La funzione `push_back()` viene utilizzata per aggiungere elementi alla fine della lista, e il ciclo for viene utilizzato per iterare su tutti gli elementi della lista.

Classe e funzioni Template

Le classi e le funzioni template

in C++ sono un potente strumento che permette di scrivere codice generico, ovvero codice che può lavorare con diversi tipi di dati senza dover essere riscritto per ciascuno di essi. Questo è particolarmente utile quando si crea una struttura dati come una stack, che potrebbe dover gestire elementi di qualsiasi tipo.

Un template di classe è una classe blu che può essere utilizzata per creare oggetti di qualsiasi tipo. Un template di funzione è una funzione che può essere utilizzata con argomenti di qualsiasi tipo. In entrambi i casi, il tipo di dati viene specificato come un parametro al template.

Ecco un esempio di come si possono utilizzare i template in C++:

```
template <typename T>
class Stack {
```

```
    private:
        std::vector<T> elements;
    public:
        void push(const T& element) {
            elements.push_back(element);
        }
        T pop() {
            T top = elements.back();
            elements.pop_back();
            return top;
        }
    };

    int main() {
        Stack<int> intStack;
        intStack.push(1);
        int top = intStack.pop();
        std::cout << top << std::endl;

        Stack<std::string> stringStack;
        stringStack.push("C++");
        std::string topString = stringStack.pop();
        std::cout << topString << std::endl;
    }
    ```

In questo esempio, `Stack` è un template di classe che può essere utilizzato per creare stack di qualsiasi tipo. La funzione `push()` aggiunge un elemento alla cima della stack, e la funzione `pop()` rimuove e restituisce l'elemento in cima alla stack. Nel `main()`, creiamo una stack di interi e una stack di stringhe utilizzando il template di classe `Stack`.

La costruzione di una classe Coda

in C++ è un esercizio fondamentale per comprendere i concetti chiave della programmazione orientata agli oggetti e della gestione della memoria.

****Puntatori****

I puntatori sono strumenti potenti in C++ che permettono di fare riferimento direttamente alla memoria del computer. In una classe Coda, i puntatori possono essere utilizzati per tenere traccia dell'inizio e della fine della coda, permettendo operazioni efficienti di inserimento e rimozione.

****Array a dimensione variabile****

Gli array a dimensione variabile sono strumenti utili per gestire collezioni di dati che possono cambiare dimensione durante l'esecuzione del programma. In una classe Coda, un array a dimensione variabile può essere utilizzato per memorizzare gli elementi della coda.

****Liste****

Le liste sono un tipo di struttura dati che può essere utilizzata per implementare una coda. Una lista consente di inserire e rimuovere elementi in modo efficiente, il che è ideale per una coda dove gli elementi vengono aggiunti alla fine e rimossi dall'inizio.

****Classe e funzioni Template****

Le classi e le funzioni template in C++ permettono di creare codice che può lavorare con diversi tipi di dati. Questo è particolarmente utile quando si crea una classe Coda, in quanto permette di creare code che possono contenere qualsiasi tipo di dati.

Ecco un esempio di come potrebbe apparire una classe Coda in C++:

```
template <typename T>
class Queue {
  private:
    T* arr;
    int capacity;
    int front;
    int rear;
    int count;

  public:
    Queue(int size = DEFAULT_SIZE);
    ~Queue();
    void dequeue();
    void enqueue(T x);
    T peek();
    int size();
    bool isEmpty();
    bool isFull();
};
```

In questo esempio, `Queue` è una classe template che può contenere elementi di qualsiasi tipo `T`. L'array `arr`

viene utilizzato per memorizzare gli elementi della coda, mentre `front` e `rear` sono puntatori che tengono traccia dell'inizio e della fine della coda. Le funzioni membro permettono di aggiungere e rimuovere elementi dalla coda, controllare se la coda è vuota o piena, e ottenere l'elemento in cima alla coda senza rimuoverlo.

Costruzione della classe Lista

La costruzione di una classe Lista in C++ è un esercizio fondamentale per comprendere come funzionano le strutture dati dinamiche. Una lista è una collezione di elementi, dove ogni elemento ha un riferimento al successivo, formando una catena. Questo tipo di struttura dati è particolarmente utile quando il numero di elementi non è noto a priori e può variare nel tempo.

****Concetti chiave:****

****Puntatori:****
I puntatori sono utilizzati per creare le liste in C++. Ogni elemento della lista (chiamato nodo) contiene un puntatore al nodo successivo. Questo permette di navigare la lista seguendo i puntatori da un nodo all'altro.

****Array a dimensione variabile:****
Anche se le liste sono più flessibili e non richiedono la conoscenza a priori del numero di elementi, a volte può essere utile utilizzare array a dimensione variabile. Questi array possono essere ridimensionati dinamicamente, il che può essere utile per implementare strutture dati come le liste.

****Liste:****

Le liste sono una delle strutture dati fondamentali in programmazione. Esistono vari tipi di liste (liste singolarmente collegate, liste doppiamente collegate, liste circolari, ecc.) e ognuna ha i suoi specifici vantaggi e svantaggi. In C++, le liste possono essere implementate utilizzando classi e puntatori.

Classe e funzioni Template:
Le classi e le funzioni template sono un potente strumento in C++ che permette di scrivere codice in modo più generale e riutilizzabile. Ad esempio, potresti creare una classe Lista template che può essere utilizzata per creare liste di qualsiasi tipo di dati.

Esempio di codice:

```
template <typename T>
class Nodo {
public:
    T dato;
    Nodo<T>* successivo;
};

template <typename T>
class Lista {
private:
    Nodo<T>* testa;
public:
    Lista() : testa(nullptr) {}
    // altri metodi della lista (inserimento, cancellazione, ricerca, ecc.)
};
```

In questo esempio, `Nodo` è una classe template che rappresenta un nodo della lista, con un dato di un certo tipo `T` e un puntatore al nodo successivo. `Lista` è una classe template che rappresenta una lista, con un puntatore alla testa della lista.

****Cosa succede nella RAM:****

Quando crei una lista, viene allocata memoria per ogni nodo che aggiungi. Questa memoria viene liberata quando il nodo viene rimosso. L'indirizzo di partenza della lista è l'indirizzo della testa della lista. Quando navighi la lista, segui i puntatori da un nodo all’altro.

"Se non commetti errori, significa che non hai afferrato pienamente il concetto. Tuttavia, evita di inviarmi codice non funzionante, altrimenti potrei doverti inviare fattura." - Prof. Alemanno Giancarlo

Programmazione di Client/Server

Concetti:

HTTP

HTTP (Hypertext Transfer Protocol) è un protocollo di rete utilizzato per trasferire dati su Internet. È il protocollo di base utilizzato per la comunicazione dati nel World Wide Web. HTTP utilizza metodi come GET e POST per inviare e ricevere dati.

Form con campi nascosti

Le form HTML possono contenere campi nascosti, che consentono di inviare dati che non sono visibili o modificabili dagli utenti. Questi possono essere utilizzati per mantenere lo stato tra diverse pagine o per inviare informazioni specifiche al server che non dovrebbero essere modificate dall'utente.

Esempio di codice:

```html
<form action="submit.php" method="post">
  <input type="hidden" name="id" value="123">
  <input type="submit" value="Submit">
</form>
```

Cookie

I cookie sono piccoli file di dati che vengono memorizzati sul computer dell'utente da un sito web. Sono utilizzati per tenere traccia delle informazioni dell'utente, come le preferenze dell'utente, il contenuto del carrello o l'ID di sessione. I cookie possono essere impostati dal server con l'intestazione Set-Cookie e possono essere inviati indietro al server con l'intestazione Cookie.

Esempio di codice:

```php
setcookie("user", "John Doe", time() + 3600);
```

Sessioni

Le sessioni sono un modo per memorizzare le informazioni (in variabili) da utilizzare in più pagine. A differenza dei cookie, le informazioni non sono memorizzate sul computer dell'utente, ma sul server. Le sessioni sono utilizzate per mantenere lo stato dell'utente tra diverse pagine, come l'ID dell'utente o il contenuto del carrello.

Esempio di codice:

```php
session_start();
$_SESSION["username"] = "John Doe";
```

Invio dati Script PHP utilizzando i Metodi GET e POST sia da form che da link

I dati possono essere inviati a uno script PHP utilizzando sia il metodo GET che POST. Il metodo GET aggiunge i dati alla URL, mentre il metodo POST li invia come un flusso di dati separato. Questo può essere fatto sia da un form HTML (utilizzando l'attributo method del form) che da un link (aggiungendo i dati alla URL del link).

Esempio di codice:

```html
<!-- Utilizzo del metodo GET -->
<a href="process.php?name=John&age=30">Invia dati con GET</a>

<!-- Utilizzo del metodo POST -->
<form action="process.php" method="post">
    <input type="text" name="name" value="John">
    <input type="text" name="age" value="30">
    <input type="submit" value="Invia dati con POST">
</form>
```

Costruzione di un Carrello per un Sito di Commercio Elettronico

La programmazione client/server è un modello di programmazione in cui il software del client e del server lavorano insieme per completare le operazioni. In un sito di commercio elettronico, il client potrebbe essere il browser web dell'utente, mentre il server sarebbe il sistema che gestisce il database del negozio, gestisce le transazioni e fornisce le pagine web al client.

Costruire un carrello per un sito di commercio elettronico è un compito che richiede una comprensione di vari concetti di programmazione web. Questi includono l'uso del protocollo HTTP, la creazione di form con campi nascosti, l'uso di cookie e sessioni, e l'invio di dati a script PHP utilizzando i metodi GET e POST.

Il protocollo HTTP è la base della comunicazione dati su World Wide Web. Diverse tecniche HTTP come GET e POST sono utilizzate per inviare e ricevere dati.

Un esempio di script PHP che utilizza il metodo POST per ricevere dati da un form potrebbe essere il seguente:

```php
```

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $nome = $_POST["nome"];
    $prodotto = $_POST["prodotto"];
    $quantita = $_POST["quantita"];
    // qui potrebbe andare il codice per aggiungere il prodotto al carrello
}
?>
```

In questo esempio, il form invia i dati allo stesso script PHP che lo contiene, utilizzando il metodo POST. Il script quindi controlla se il metodo di richiesta è POST e, in caso affermativo, recupera i dati inviati.

I campi nascosti nelle form consentono di inviare dati che non sono visibili o modificabili dagli utenti. Questi possono essere utilizzati per mantenere lo stato tra diverse pagine.

```html
<form method="post" action="aggiungi_al_carrello.php">
    <input type="hidden" name="id_prodotto" value="123">
    <input type="submit" value="Aggiungi al carrello">
</form>
```

In questo esempio, l'ID del prodotto viene inviato come un campo nascosto quando l'utente fa clic sul pulsante "Aggiungi al carrello".

I cookie sono dati memorizzati sul computer dell'utente e sono utilizzati per tenere traccia delle informazioni dell'utente, come le preferenze dell'utente o il contenuto del carrello.

```php
<?php
setcookie("carrello", serialize($carrello), time() + (86400 * 30), "/"); // 86400 = 1 giorno
?>
```

In questo esempio, il contenuto del carrello viene salvato in un cookie che dura un giorno.

Le sessioni sono un modo per memorizzare le informazioni (in variabili) da utilizzare in più pagine. A differenza dei cookie, le informazioni non sono memorizzate sul computer dell'utente, ma sul server.

```php
<?php
session_start();
$_SESSION["carrello"] = $carrello;
?>
```

In questo esempio, il contenuto del carrello viene salvato in una variabile di sessione.

Registrazione ad aree riservate

La costruzione di script e pagine per la registrazione ad aree riservate è un aspetto fondamentale nello sviluppo di qualsiasi applicazione web che richiede un'autenticazione dell'utente. Questo processo coinvolge la creazione di un form di registrazione che raccoglie i dati dell'utente, come nome utente, email e password, e un script PHP che elabora questi dati e li memorizza in un database.

Ecco un esempio di come potrebbe apparire un form di registrazione:

```html
<form action="registrazione.php" method="post">
    Nome utente: <input type="text" name="username" required><br>
    Email: <input type="email" name="email" required><br>
    Password: <input type="password" name="password" required><br>
    <input type="submit" value="Registrati">
</form>
```

E qui c'è un esempio di come potrebbe apparire lo script PHP che elabora i dati del form:

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST["username"];
    $email = $_POST["email"];
    $password = $_POST["password"];

    // Qui dovresti aggiungere il codice per memorizzare questi dati nel tuo database
    // Assicurati di proteggere la password prima di memorizzarla, ad esempio utilizzando la funzione password_hash di PHP
}
?>
```

Ricorda, è molto importante proteggere i dati sensibili degli utenti. Non dovresti mai memorizzare le password in chiaro nel tuo database. Invece, dovresti utilizzare una funzione di hashing, come `password_hash` in PHP, per memorizzare una versione protetta della password. Quando un utente si autentica, puoi utilizzare la funzione `password_verify` per confrontare la password inserita dall'utente con la versione hash memorizzata nel tuo database.

Inoltre, dovresti sempre validare e sanificare i dati del form prima di utilizzarli per prevenire attacchi come l'iniezione SQL e lo scripting cross-site (XSS). PHP offre molte funzioni utili per questo, come `filter_var` e `htmlspecialchars`.

Accedere ad Aree Riservate

La costruzione di script e pagine per l'accesso a aree riservate è un compito comune nella programmazione web. Questo processo coinvolge la creazione di un sistema di autenticazione che consente agli utenti di accedere a determinate parti di un sito web solo dopo aver fornito le credenziali corrette.

****HTTP:****
Il protocollo HTTP è la base della comunicazione dati su World Wide Web. Diverse tecniche HTTP come GET e POST sono utilizzate per inviare e ricevere dati.

****Sessioni:****
Le sessioni sono un modo per memorizzare le informazioni (in variabili) da utilizzare in più pagine. A differenza dei cookie, le informazioni non sono memorizzate sul computer dell'utente, ma sul server. Le sessioni sono utilizzate per mantenere lo stato dell'utente tra diverse pagine, come l'ID dell'utente o il contenuto del carrello.

Esempio di codice:

```php
<?php
// Inizia la sessione
session_start();
```

```
    // Controlla se l'utente è già loggato
    if(isset($_SESSION["loggedin"]) &&
$_SESSION["loggedin"] === true){
        header("location: welcome.php");
        exit;
    }

    // Include il file di configurazione del database
    require_once "config.php";

    // Definisci le variabili e inizializza con valori vuoti
    $username = $password = "";
    $username_err = $password_err = "";

    // Elabora i dati del modulo quando il modulo viene
inviato
    if($_SERVER["REQUEST_METHOD"] == "POST"){

        // Controlla se il nome utente è vuoto
        if(empty(trim($_POST["username"]))){
            $username_err = "Inserisci il nome utente.";
        } else{
            $username = trim($_POST["username"]);
        }

        // Controlla se la password è vuota
        if(empty(trim($_POST["password"]))){
            $password_err = "Inserisci la tua password.";
        } else{
            $password = trim($_POST["password"]);
        }
```

```
    // Valida le credenziali
    if(empty($username_err) &&
empty($password_err)){
        // Prepara una dichiarazione di selezione
        $sql = "SELECT id, username, password FROM
users WHERE username = ?";

        if($stmt = mysqli_prepare($link, $sql)){
            // Collega le variabili alla dichiarazione preparata
come parametri
            mysqli_stmt_bind_param($stmt, "s",
$param_username);

            // Imposta i parametri
            $param_username = $username;

            // Tenta di eseguire la dichiarazione preparata
            if(mysqli_stmt_execute($stmt)){
                // Memorizza il risultato
                mysqli_stmt_store_result($stmt);

                // Controlla se esiste il nome utente, se esiste,
verifica la password
                if(mysqli_stmt_num_rows($stmt) == 1){
                    // Collega le variabili al risultato
                    mysqli_stmt_bind_result($stmt, $id,
$username, $hashed_password);
                    if(mysqli_stmt_fetch($stmt)){
                        if(password_verify($password,
$hashed_password)){
                            // Se la password è corretta, inizia una
nuova sessione
```

Approfondimenti proposti ad Alcuni Studenti

"Non fidatevi mai ciecamente di ciò che vi viene detto, cercate sempre la verità e verificate anche le mie parole. Ricordatevi che anch'io sono solo umano e posso commettere errori." - Prof. Alemanno Giancarlo

Utilizzo di GitHub per la condivisione del codice e la gestione delle versioni

GitHub è una piattaforma di hosting di codice sorgente che utilizza il sistema di controllo delle versioni Git. È utilizzato per la condivisione del codice con altri sviluppatori e per la gestione delle versioni del codice. Puoi creare repository per organizzare il tuo codice e invitare altri sviluppatori a collaborare. Inoltre, GitHub offre funzionalità come la gestione delle issue e dei pull request per migliorare la collaborazione e la gestione del codice.

Esempio di codice: Creazione di un nuovo repository su GitHub utilizzando la riga di comando.

```
# Naviga nella directory del tuo progetto
cd my_project

# Inizializza un nuovo repository Git
git init

# Aggiungi tutti i file al repository
git add .

# Effettua il commit dei file
git commit -m "Primo commit"
```

Aggiungi l'URL del tuo repository GitHub
git remote add origin https://github.com/username/my_project.git

Spinge i tuoi commit su GitHub
git push -u origin master
```

Ecco un esempio di come si potrebbe utilizzare GitHub:

```bash
# Clona un repository esistente
**git clone https://github.com/username/repository.git**

# Naviga nel repository clonato
**cd repository**

# Crea un nuovo branch
**git branch new-feature**

# Passa al nuovo branch
**git checkout new-feature**

# Modifica i file, quindi aggiungi le modifiche al commit
**git add .**

# Esegui il commit delle modifiche
**git commit -m "Aggiunta nuova funzionalità"**

# Spinge il branch e le modifiche al repository remoto su GitHub
```
```

git push origin new-feature
```

Questo codice clona un repository esistente, crea un nuovo branch, apporta delle modifiche, le commit e le spinge su GitHub. Da lì, le modifiche possono essere riviste e unite nel branch principale.
```

Approfondimento sull'installazione e l'utilizzo di Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft. È gratuito, open source e disponibile su Windows, macOS e Linux. Supporta una varietà di linguaggi di programmazione come JavaScript, TypeScript, Python, C++, Java, e molti altri. VS Code offre funzionalità come evidenziazione della sintassi, completamento automatico del codice, refactoring del codice, visualizzazione del debug e molte altre funzionalità.

Per installare Visual Studio Code, puoi scaricarlo dal [sito ufficiale]**(https://code.visualstudio.com/Download)** e seguire le istruzioni di installazione per il tuo sistema operativo.

Utilizzo di API ChatGPT per la realizzazione di semplici app in Python

ChatGPT è un modello di linguaggio sviluppato da OpenAI che può generare risposte umanoidi a un testo di input. Puoi utilizzare l'API di ChatGPT per integrare questo modello di linguaggio nelle tue applicazioni.

Esempio di codice: Creazione di un semplice bot di chat utilizzando l'API di ChatGPT.

```python
import openai

# Imposta la tua chiave segreta di OpenAI
openai.api_key = 'your-api-key'

# Invia una richiesta all'API di ChatGPT
response = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "Sei un assistente utile."},
    {"role": "user", "content": "Chi ha vinto la serie mondiale nel 2020?"}
  ]
)
```

Stampa la risposta del modello
print(response['choices'][0]['message']['content'])
```

**Nota: sostituisci `'your-api-key'` con la tua chiave**
```

Utilizzo del software: Curl e della libreria utilizzata con PHP per la gestione della sicurezza.

Curl è uno strumento da riga di comando utilizzato per trasferire dati da o verso un server, utilizzando uno dei protocolli supportati (HTTP, HTTPS, FTP, IMAP, POP3, SCP, SFTP, SMTP, TFTP, TELNET, LDAP o FILE). È molto utile per testare, scaricare da URL, o anche per l'interazione con un'API.

Ecco un esempio di come si potrebbe utilizzare Curl dalla riga di comando:

```
# Esempio di richiesta GET
curl http://www.example.com

# Esempio di richiesta POST
curl -d "param1=value1&param2=value2" -X POST http://www.example.com

# Esempio di richiesta POST con dati JSON
curl -d '{"key1":"value1", "key2":"value2"}' -H "Content-Type: application/json" -X POST http://www.example.com

# Esempio di invio di un header
```

curl -H "Custom-Header: value" http://www.example.com

Esempio di richiesta PUT
curl -d "param1=value1" -X PUT http://www.example.com

Esempio di richiesta DELETE
curl -X DELETE http://www.example.com
```

Nell'esempio, `-d` specifica i dati da inviare, `-X` specifica il metodo HTTP da utilizzare, `-H` specifica un header da inviare con la richiesta.

**Ricorda che Curl è molto potente e queste sono solo alcune delle cose che puoi fare con esso. Per un elenco completo delle opzioni disponibili, puoi sempre consultare la documentazione di Curl o utilizzare il comando `man curl` in un terminale Unix/Linux.**
```

Curl è uno strumento da riga di comando e una libreria per trasferire dati con URL. È un potente strumento utilizzato per testare API, scaricare file e molto altro. PHP supporta libcurl, una libreria creata da Daniel Stenberg, che consente di connettersi e comunicare con diversi tipi di server con diversi tipi di protocolli.

Ecco un esempio di come si potrebbe utilizzare Curl con PHP:

```php
<?php
// crea una nuova risorsa curl
$ch = curl_init();

// imposta l'URL e altre opzioni appropriate
curl_setopt($ch, CURLOPT_URL, "http://www.example.com/");
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_HEADER, 0);

// esegui la richiesta
$output = curl_exec($ch);

// controlla se ci sono errori
if ($output === FALSE) {
    echo "cURL Error: " . curl_error($ch);
}

// chiudi la sessione curl
curl_close($ch);
```

```
// stampa l'output
echo $output;
?>
```

Questo script PHP utilizza Curl per fare una richiesta GET a "http://www.example.com/", quindi stampa l'output. Curl è molto versatile e può essere utilizzato per una vasta gamma di operazioni HTTP, tra cui POST, PUT, DELETE e molto altro.

Ambiente per lo sviluppo delle esercitazioni e delle applicazioni: GDB Online

Siti di Riferimento per lo studio della disciplina:
- **https://www.onlinegdb.com/**
- **https://cplusplus.com/**
- **https://www.php.net/**
- **https://www.w3schools.com/**

- **http://www-old.bo.cnr.it/corsi-di-informatica/corsoCstandard/Lezioni/01Indice.html**

- **Appunti del Docente**

Lavori personali svolti durante l'anno Caricati in una repository su git hub

https://github.com/Alexinfotech/molinari.git

Lavori extra corso

Blog costruito grazie ai consigli del Prof. Alemanno Giancarlo

https://www.alexinfotech.it/

Sito web per esercitazione su Raspbarry
Contenuto in un Container Docker

http://minchino.ddns.net/

SIAMO QUELLI DELLA NOTTE!!!

Lista esaustiva argomenti

Funzioni
Definizione di una funzione
Dichiarazione di una funzione
Istruzione return
Comunicazioni fra programma chiamante e funzione
Argomenti di default
Funzioni con overload
Direttive al Preprocessore
Direttiva #include
Direttiva #define di una costante
Indirizzi e Puntatori
Operatore di indirizzo &
Cosa sono i puntatori ?
Dichiarazione di una variabile di tipo puntatore
Assegnazione di un valore a un puntatore
Aritmetica dei puntatori
Operatore di dereferenziazione *
Funzioni con argomenti puntatori
Puntatori ed Array
Analogia fra puntatori ed array
Combinazione fra operazioni di deref. e di incremento
Confronto fra operatore [ ] e deref. del puntatore "offsettato"
Funzioni con argomenti array
Funzioni con argomenti puntatori passati by reference
Array di puntatori

Puntatori e Costanti
Puntatori a costante
Funzioni con argomenti costanti trasmessi by value

Funzioni con argomenti costanti trasmessi by reference
Tipi definiti dall'utente
Concetti di oggetto e istanza
Typedef
Strutture
Operatore .
Puntatori a strutture - Operatore ->
Allocazione dinamica della memoria
Memoria stack e memoria heap
Operatore new
Operatore delete
Strutture Dati Dinamiche
Namespace
Programmazione modulare e compilazione separata
Definizione di namespace
Estendibilità della definizione di un namespace
Parola-chiave using
Eccezioni
Segnalazione e gestione degli errori
Il costrutto try
L'istruzione throw
Il gestore delle eccezioni: costrutto catch
Riconoscimento di un'eccezione fra diverse alternative
Classi e Programmazione ad Oggetti

Analogia fra classi e strutture
Specificatori di accesso
Data hiding
Funzioni membro
Risoluzione della visibilità
Funzioni-membro di sola lettura

Classi membro
Puntatore nascosto this
Costruttori e distruttori degli oggetti
Costruzione e distruzione di un oggetto
Costruttori
Costruttori e conversione implicita
Distruttori
Oggetti allocati dinamicamente
Membri puntatori
Costruttori di copia
Liste di inizializzazione
Membri oggetto
Array di oggetti
Utilità dei costruttori e distruttori
Overload degli operatori
Estendibilità del C++
Ridefinizione degli operatori
Metodi della classe o funzioni esterne ?
Il ruolo del puntatore nascosto this
Overload degli operatori di flusso di I/O
Operatori binari e conversioni
Operatori unari e casting a tipo nativo
Ereditarietà
L'eredità in C++
Classi base e derivata
Accesso ai membri della classe base

Conversioni fra classi base e derivata
Costruzione della classe base
Polimorfismo

Funzioni virtuali
Tabelle delle funzioni virtuali
Classi astratte
Template
Programmazione generica
Definizione di una classe template
Istanza di un template
Parametri di default
Funzioni template
Differenze fra funzioni e classi template
Generalità sulla Libreria Standard del C++
Campi di applicazione
Header files
Il namespace std
La Standard Template Library
Generalità
Iteratori
Contenitori Standard
Algoritmi e oggetti-funzione
Le classi string e vector
La classe string
Confronto fra string e vector<char>
Confronti fra stringhe
Concatenazioni e inserimenti
Ricerca di sotto-stringhe
Estrazione e sostituzione di sotto-stringhe

Introduzione alla programmazione

Client/Server
Concetto di Applicazione Client
Concetto di Applicazione Server
Applicazioni WEB
HTTP un esempio di protocollo di rete
Fondamenti di HTML
Struttura di una pagina Web
Paragrafi
Liste
Tabelle
Link
Header
Form

Fondamenti di PHP
Introduzione PHP
Confronto C++/PHP
Installazione XAMPP
Sintassi PHP
Variabili PHP
PHP Echo / Stampa
Tipi di dati PHP
Stringhe PHP
Numeri PHP
PHP Matematica: FUNZIONE RAND()
Costanti PHP
Operatori PHP
Costrutti di Selezione
Costrutti Iterativi
Funzioni PHP

Array PHP
Concetto di Array Associativi

Gli Array Associativi Super globali PHP
$_GET
$_POST
$_COOKIE

$_SESSION
$_SERVER

LABORATORIO E LAVORI PIU' RILEVANTI SVOLTI E PROPOSTI

Costruzione della Classe Frazione e creazione di un nuovo tipo di variabile
Concetti:
Uso del Puntatore THIS
Applicazione del principio di sovrapposizione delle funzioni
Ridefinizione degli operatori
Costruzione della classe vettore
Concetti:
Puntatori
Aritmetica dei Puntatori
Liste
Classe e funzioni Template
Costruzione della classe Stack
Concetti:
Puntatori
Array a dimensione Variabile
Liste
Classe e funzioni Template
Costruzione della classe Coda
Concetti:
Puntatori

Array a dimensione Variabile
Liste
Classe e funzioni Template
Costruzione della classe Lista
Concetti:
Puntatori
Array a dimensione Variabile
Liste
Classe e funzioni Template

Programmazione di Client/Server
Costruzione di Carrello per sito di commercio elettronico
Concetti:
HTTP
Form con campi nascosti
Cookie
Sessioni
Invio dati Script PHP utilizzando i Metodi GET e POST sia da form che da link
Costruzione Script e Pagina per la Registrazione ad aree riservate
Concetti:
HTTP
Form con campi nascosti
Cookie
Sessioni
Invio dati Script PHP utilizzando i Metodi GET e POST sia da form che da link
Costruzione Script e Pagina Accedere ad aree riservate
Concetti:
HTTP

Form con campi nascosti
Cookie
Sessioni
Invio dati Script PHP utilizzando i Metodi GET e POST sia da form che da link