10° lezione programmazione II

**Linguaggi di programmazione**

Si utilizzano per codificare gli algoritmi. Un linguaggio di programmazione è  un insieme di istruzioni che vengono utilizzate per codificare programmi. Distinguiamo due tipologia di linguaggi di programmazione:

-   ​**Linguaggi a basso livello:**

o  **Linguaggio macchina:**

o  **Assembly:**è molto simile al linguaggi macchina con la differenza che si utilizzano forme mnemoniche per le corrispondenti istruzioni macchina.

-   ​**Linguaggi a alto livello(si avvicinano al nostro modo di parlare):**ha i seguenti vantaggi:

**-   ​astrazione**

**- ​semplificazione**

**-  ​similarità con il ragionamento umano**

**Paradigmi di programmazione**

**I paradigmi sono la filosofia con cui si scrivono i programmi.**

Un linguaggio di programmazione è basato su un paradigma di programmazione che è caratterizzato da:

-   ​**metodologia** con cui si scrivono i programmi

-   ​**computazione** istruzioni o funzioni matematiche

Esistono svariati paradigmi di programmazione come:

-   ​**programmazione funzionale:** è costituita da valutazione funzione matematiche. Si ha un assenza di side-effect perché non si fa uso di istruzioni imperative (non si posso fare modifiche alla memoria)

-   ​**programmazione logica:**descrive la struttura logica del programma (logica del primo ordine)

-   ​**programmazione imperativa:**sequenza di istruzioni

-   ​**programmazione strutturata :**è basata sul teorema di Bohm Jacobini (**tre costrutti sequenza, selezione e iterazione)**.L'istruzione  Go to permette di fare dei salti in alcune parti del programma.

-   ​**programmazione procedurale:**blocchi di codice sorgente identificati da un nome ed eventuali argomenti.

-   **programmazione modulare**:  strutturare un programma in moduli ben definiti. mettendo insieme queste funzioni si può pensare di aggregarle logicamente in un modulo. Il modulo è fatto da un certo set di funzioni, alcune di queste saranno dedicate al funzionamento interno del  modulo quindi alle computazioni che avvengono all'interno che non sono visibili all'utente finale. Altre di queste funzioni costituiranno la cosiddetta interfaccia del modulo sono funzioni che si potranno invocare esplicitamente attraverso le quali potremmo attivare certe computazioni.

-   ​**programmazione orientata agli oggetti(OOP):**Introduce una serie di costrutti. la classe è una estrazione della realtà invece l'oggetto è un'istanza della classe. I metodi sono le funzionalità dell'oggetto. con la programmazione ad oggetti sia un legame tra i  dati e i metodi che vengono incapsulate nelle classi. Si ha anche la possibilità di avere dei metodi che non sono visibili all'utente cosiddetti metodi privati che servono solo per l'implementazione dell'oggetto.

Immagine che contiene testo, schermata, Carattere, design

Descrizione generata automaticamente

**I linguaggi di programmazione sono definiti da un insieme di regole formali cioè la sintassi che include tutti i simboli da utilizzare durante la scrittura del programma. Le regole di sintassi definiscono come comporre i simboli e le parole per comporre istruzioni corrette. La semantica di un'istruzione è il significato di questa .  attenzione!!  se in un programma non ci sono errori sintattici non vuol dire che la semantica del programma.**

**Traduzione**Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Grazie al concetto di traduzione si ha:

•   **evoluzione del linguaggio di programmazione**( Maggiormente espressivi)

• **facilità nella scrittura del programma** perché sono vicini al linguaggio parlato

•  **Indipendenza dalla piattaforma:** Il programmatore che sta programmando in c non si deve preoccupare della piattaforma(CPU).  Questo problema se la deve porre chi scrive e progetta i compilatori  per le varie CPU.

**si hanno due tipi di traduttori:**

1. **L'interpretazione**traduce ed esegue ogni singola istruzione del programma. Legge ed esegue il codice sorgente del programma senza creare un file oggetto eseguibile. E' più lenta rispetto alla compilazione**.**

2. **compilatore:** il compilatore prende in pasto un programma scritto con un linguaggio alto livello e lo traduce in linguaggio macchina. Il compilatore opera su una architettura che si dice architettura " target ” se su una macchina si installa il compilatore e si va a compilare un programma si ottiene un programma eseguibile per qualunque altra macchina che abbia la stessa architettura.  Se invece il programmatore Dovesse compilare questo programma per architetture diverse dall'architettura della macchina di sviluppo, allora Dovrebbe effettuare una procedura  chiamata cross compilazione.

Nel processo di traduzione del compilatore si hanno almeno due parti:

1.  **back end**:  è la parte che cambia da architettura ad architettura e si occupa di generare  istruzioni macchina

2. **front end**: ha la funzione di eseguire l'analisi sintattica del programma di fare il parsing.

I linguaggi interpretati e compilati hanno dei vantaggi degli svantaggi:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Nei linguaggi interpretati il programmatore **non si deve preoccupare nell'architettura su cui verrà eseguito il programma.** infatti il programmatore scrive il programma in Java la compilazione produce un **programma in byte code**  che è un linguaggio simile al linguaggio macchina ma è indipendente dall'architettura e **il  byte code  viene eseguito mediante un interprete Java installato su una macchina.**

**JVM  è la macchina virtuale che interpreta il codice Byte code.**

**Le classi**

**La classe è l’astrazione della realtà**. Formata da metodi e attributi. È da intendersi come un insieme di oggetti che hanno **la stessa struttura e le stesse funzioni**.

Le classe definiscono tipi di dati personalizzati in funzione al problema da risolvere.

È consigliato separare l’interfaccia dall’implementazione (chi progetta la classe deve sapere i metodi implementativi, invece, chi utilizza la classe deve conoscere soltanto l’interfaccia).

**Come si definisce una classe?**

Immagine che contiene testo, ricevuta, Carattere, bianco

Descrizione generata automaticamente

 Si compone di due parti:

1. **Dichiarazioni**: definire dati(attributi) e metodi
2. **Definizione**: implementazione dei metodi (che interagiscono sugli attributi)

Specificatori di accesso

Ne esistono di 3 tipi:

N.B. gli attributi ed i metodi hanno gli specificatori di accesso.

1. **Privati(private):** I metodi e gli attributi sono visibili sono nella classe stessa dove sono stati definiti.
2. **Pubblici(public):** I metodi e gli attributi sono visibili tutte le classi e le sottoclassi.
3. **Protetti(protected):** I metodi e gli attributi sono visibile nelle classi derivate della stessa.

Immagine che contiene testo, ricevuta, Carattere, bianco

Descrizione generata automaticamente

Information hiding

Con la OOP si ha la possibilità di occultare attributi e metodi, così facendo si limitano gli errori.

.Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente

I metodi (intestazione delle funzioni) sono inseriti nella sezione pubblica, invece, gli attributi (caratteristiche dell’oggetto).

L’interfaccia (costruttore […]) deve essere pubblica se non può essere invocata.

**Cosa bisogna sapere per definire una classe?**

* **Nome della classe**
* **La posizione in cui è definita**
* **Che operazioni supporta**

**Il nome della classe è definito in un file header con lo stesso nome della classe dove vengono inserite le intestazioni dei metodi e gli attributi**. **Le implementazioni** dei metodi vengono eseguite in un file con estensione .**cpp** tipicamente con lo stesso nome della classe.

**Cos’è un oggetto?**

Un oggetto è un’istanza della classe. Ci possono essere più istanza della stessa classe con parametri diversi. Una fra le potenza della OOP è **incapsulamento ciò vuol dire che all’interno dell’oggetto ci sono metodi e attributi.** Un ogetto si crea con il nome della classe e un identificatore (solitamente si usa il nome della classe in minuscolo). Gli oggetti si possono associare in parte alle struct in C con la differenza che l’operatore di accesso al campo (.) diventa l’operatore di accesso agli attribuiti e i metodi.

**Come copiare un oggetto in C++?**

In C++ si fa una copia bit a bit di tutti i membri. Tutti i dati dell’oggetto originale vengono copiati nell’oggetto destinatario.

Immagine che contiene testo, Carattere, design

Descrizione generata automaticamente

Gli attributi

Possono essere di qualunque tipo valido(int, float,double) **tranne il tipo della classe che si sta definendo**. **Un attributo è formato da il tipo e da un identificatore**. Il tipo può essere anche un oggetto di un'altra classe. Solitamente **per conversione** si utilizza la **notazione camelCase(nomePersona).**

**I metodi**

Si possono definire e definire all’interno della classe.

La definizione dei metodi si compone di **4 parti**:

1. **Tipo restituito dalla funzione**
2. **Il nome della funzione**
3. **La lista di parametri formali**
4. **Il copro della funzione (quello che fa)**

Le prime 3 parti compongono **la signature o il prototipo della funzione che va definito dentro la classe**, **mentre l’implementazione può essere definita in un file a parte solitamente con estensione .cpp**

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**Come si esegue la chiama un metodo?**

I metodi di una classe si invocano cosi come si accede ai dati di un oggetto tramite l’operatore (.) con la seguente sintassi:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**Quali tipi di metodi ci sono?**

* **Costruttori e distruttori:** vengono invocati automaticamente alla creazione e alla distruzione di oggetti.
* **Selettori(set):** si utilizzano per permettere di restituire gli attributi privati di una classe definita precedentemente in una altra classe.
* **Modificatori:** si utilizzano per permettere di modificare gli attributi privati di una classe definita precedentemente in una altra classe.
* **Operatori:** ridefinizioni di operatori << == >>
* **Iteratori:** elaborazione di collezione di oggetti

**Costruttori**

È un metodo di una classe **PUBBLICO** che viene eseguito quando si crea l’oggetto. Ha un numero variabile di parametri, ha lo stesso nome della classe. **Non restituisce nulla.**Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Quando si crea un oggetto si passano i parametri al costruttore, nel seguente modo:

c Immagine che contiene testo, Carattere

Descrizione generata automaticamente

Si hanno diversi tipi di costruttori:

* **Costruttore di default:** si utilizza per inizializzare gli attributi
* **Costruttore parametrizzato**: si utilizza per dare dei valori all’oggetto
* **Costruttore di copia**: viene creato automaticamente dal compilatore quando viene passato un oggetto per valore

**Come inizializzare gli attributi di un oggetto?**

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamenteImmagine che contiene testo, schermata

Descrizione generata automaticamente

Nella figura di sopra si ha un caso di casting implicito degli attributi.

**Costanti(const)**

Ad una variabile si può cambiare il contenuto. Ad una costante no. Infatti bisogna inizializzarla quando si definisce.

**Puntatori costanti e puntatori a costanti**

Prima di definire che cos'è il puntatore a  costante definiamo Che cos'è una costante. Una costante è una variabile di un certo tipo **che non può cambiare durante il runtime del programma.**

**Puntatori a costanti**

**int num=10;**

**int\* const ptr=&num;**

**Mediante il puntatore ptr non si potrà modificare il valore puntato dal puntatore stesso** (come una costante). **Si deduce pertanto che va inizializzato contestualmente alla sua dichiarazione.**È irrilevante (in questo caso) se il dato è costante o variabile.

**Puntatori costanti**

**int num=20;**

**const int \*ptr=&num;**

Si usa quando attraversa il puntatore ptr non si vuole modificare il valore alla locazione di memoria puntata dal   puntatore.**Si deduce pertanto che va inizializzato contestualmente alla sua dichiarazione**

**Riassumendo**

**Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**

**Parametri di funzioni costanti**

Ciò vuol dire che il **parametro non deve essere modificato**.

**I metodi constanti**

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

**Non si può modificare il valore degli attributi dell’oggetto di appartenenza**.

**Valori di ritorno dei metodi costanti**

Immagine che contiene testo, schermata, Carattere, bianco

Descrizione generata automaticamente

Non si può modificare il valore del metodo di ritorno.

**Distruttore**

Immagine che contiene testo, ricevuta, Carattere, bianco

Descrizione generata automaticamente

Quando viene distrutto un oggetto viene chiamato questo metodo, non accetta parametri. Ha lo stesso nome della classe preceduto dal simbolo ~. Se non sene definisce uno C++ lo crea vuoto. Si utilizza per liberare la memoria assegnata dal costruttore.

**Metodi statici**

Immagine che contiene testo, schermata, ricevuta, Carattere

Descrizione generata automaticamente

È associato alla classe ma non all’oggetto ciò vuol dire che si ha una copia di questo in tutte le istanze che si creano. È possibile invocare questo metodo senza aver creato l’oggetto.

**Friend ed incapsulamento**

**Funzioni friend**: Una funzione può essere dichiarata come amica di una classe. Ciò significa che la funzione può accedere direttamente ai membri privati e protetti della classe come se fosse un membro della classe stessa. Per dichiarare una funzione come amica di una classe, la dichiarazione della funzione deve precedere la dichiarazione della classe e la parola chiave friend deve essere utilizzata nella dichiarazione della classe.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente

**La funzione operator ==**

Non è un metodo della classe ma nonostante questo riesce ad accedere agli attributi privati della classe dove si sta definendo tramite la keyword friend. Questa consente di uniformare I/O della classe in questione senza violare il principio di incapsulamento. Infatti, i metodi con friend devono essere inseriti nella definizione della classe. Ciò ci consente di rimanere in pieno controllo dell’interfaccia.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

**Over loading dei metodi e degli operatori**

I metodi possono essere sovraccaricati ma solo nella classe stessa. L’over loading permette di utilizzare lo stesso nome di un metodo per inserire diverso numero di parametri ad ognuno di questi. **Due metodi non possono avere lo stesso numero di parametri e egual nome.** Il compilatore riesce a capire quale metodo eseguire in base al numero di token(parametri) che si passano al metod quando si esegue la chimata a funzione.Immagine che contiene testo, ricevuta, Carattere, schermata

Descrizione generata automaticamente

**Gestioni delle eccezioni**

**Si utilizzano:**

* **try:** La keyword try viene utilizzata per incapsulare il codice che potrebbe generare un'eccezione. All'interno di un blocco try, il programma monitora la possibile generazione di eccezioni.
* **throw:** La keyword throw viene utilizzata per segnalare una condizione eccezionale all'interno di un blocco try. Quando si verifica una situazione eccezionale, un'istruzione throw viene eseguita per lanciare un oggetto eccezione, che può essere di qualsiasi tipo.
* **catch:** La keyword catch viene utilizzata per catturare e gestire un'eccezione lanciata all'interno di un blocco try. Un blocco catch specifica il tipo di eccezione che può essere gestito e il codice da eseguire per gestire l'eccezione.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

In molti casi possiamo prevenire gli errori a runtime dei nostri programmi

quando si verificano situazioni anomale relativamente alla logica del

problema