# Programmazione I

## Indice:

Ia ai non ha inventiva

Inventiva = ricerca della soluzione, ricerca del miglior programma (implementazione)

DevOps e DevSecOps sono nuovi processi e dice che bisogna seguire tutta la creazione del codice

# Lezione 1- Formalizzazione di un Linguaggio

*24 settembre 2025*

Il linguaggio che si usa per programmare è formale e ha delle regole specifiche

Il linguaggio di parole è formato da 3 livelli

- Pragmatico: quello che ho scritto che mi fa arrivare all'obiettivo
- Semantico: il significato della parola
- Sintattico: quali sono le conseguenze di un certo programma

Una stringa è una sequenza finita di simboli appartenenti all'alfabeto

Di una stringa si può definire quanti simboli ci sono al suo interno

Esiste anche la stringa vuota che non contiene nessun simbolo

Kleene Start è quella funzione che permette di creare tutte le possibili stringhe a partire dall'alfabeto e può generare una stringa vuota

Si possono unire due stringhe con la concatenazione e posso fare concatenazioni multiple

Si può avere una sottostringa

La chiusura positiva è come Kleene Start,  ma non genera la stringa vuota (n > 0)

Un linguaggio formale è un insieme di parole su un alfabeto

Non tutti i linguaggi non hanno un elenco come le password

Il linguaggio delle password possono creare un infinito elenco

Sigma Star sono tutte le parole anche quelle vuote e alcuni litiguaggi possono essere vuoti

La cardinalità è il numero di stringhe che compongono quel linguaggio

Il linguaggio è un codice quando è univocamente decifrabile

C = {1,10} è un codice perché sono totalmente diversi

C = {bab, aba, ab} non è un codice perché posso ottenere la stessa frase con parole diverse

(ababab -> aba + bab

-> ab+ab+ab)

Il codice è un linguaggio univocamente decifrabile

I virus si ammogliano tra di loro

La potenza del linguaggio può riconoscere un linguaggio (che può essere infinito), verificando parola per parola se rispetta le regole di quel linguaggio.

Approccio riconoscitivo: procedura algoritmica per capire se quella parola è appartenente a un linguaggio

Un'altro approccio è quello generativo, mi arriva una parola e vedo se quella parola la riesco a generare per conto mio, se non riesco non appartiene a quel linguaggio

Il linguaggio è regolare perché ci sono delle regole che mi dice com'è fatto

Posso tradurre da un linguaggio ad un'altro ad esempio il compilatore traduce da C a binario

# Lezione 2 – Automi a Stati Finiti

*24 settembre 2025*

Un automa stati finito è un modello matematico che dato un ingresso da un uscita

Un automa stati è finito come una quintupla su un alfabeto

Un sistema può trovarsi in situazioni diverse queste situazioni sono gli stati

Gli stati sono un'insieme di informazioni di un sistema

Esempio di stato nella vita reale: il semaforo

- Stati possibili:
  - Verde
  - Giallo
  - Rosso
- Ingresso: il passare del tempo (ad esempio ogni 30 secondi).
- Transizioni:
  - Se lo stato è **Verde** e passa il tempo → diventa **Giallo**
  - Se lo stato è **Giallo** e passa il tempo → diventa **Rosso**
  - Se lo stato è **Rosso** e passa il tempo → torna a **Verde**

Il computer quando viene avviato è in uno stato iniziale e aspetta un input, a seconda dell'input si sposta in un'altro stato

La rappresentazione grafica è un cerchio e il cambio di stato si indica con una freccia con sopra l'input che si da

Un automa stato modella dei sistemi

Il comportamento esterno è quello di una black box

Passa da uno stato all'altro e rimane in uno stato finchè la macchina non li da un'altro input

Auto anello è una freccia che indica che l'automa sta nello stesso stato

L'automa definisce un grafo e il grafo è il linguaggio delle soluzioni

Deterministico = da ogni stato e per ogni simbolo c'è al massimo una transizione possibile.

Non Deterministico = da uno stato e per un simbolo possono esserci più transizioni possib

Gli stati sono finiti

Le proprietà sono:

- Dinamicità: evoluzione attraverso gli stati
- Discretezza:
-

Ci sono anche gli stati non deterministici che servono a gestire qualcosa che non funziona sempre alla stesso modo, ma funziona a probabilità

Ci sono due tipi di macchine a stati che generano output:

- Macchina di Mearly = genera l'output dalla transazione dello stato
- Macchina di Moore = genera output a seconda dello stato in cui si trova

# Lezione 3 – Macchina di Turing

*30 settembre 2025*

Turing si è inventato una macchina a stati che era in grado di spostarsi da uno stato all'altro in automatico senza bisogno dell'uomo

La sua macchina era molto più veloce di un calcolatore perché aveva memoria infinita

La macchina di Turing servono per calcolare la complessità e la calcolabilità dei problemi quindi se si possono risolvere o meno

L'unica cosa che deve fare l'utente è darli l'input

Questa macchina può scrivere e leggere su nastro, si può spostare a destra e sinistra di questo nastro che è infinito, il nastro è la sua memoria

Legge sul nastro attraverso una testina

Hanno creato anche una versione multinastro non deterministico quindi piuttosto che avere input e output sullo stesso nastro ha input su un nastro mentre output su un'altro

Legge e scrive i simboli dell'alfabeto, ogni volta che legge o scrive un simbolo cambierà stato

Il nastro è la testina hanno uno stato iniziale

La macchina di Turing risolve una funzione (vedere slide prof)

Definizione formale della macchina di Turing (vedere slide prof)

S è lo stato iniziale, I è il simbolo letto, s è lo stato successivo, V è il nasto che scorre

Ha l'output sulla transizione

Per ogni problema ho una macchina di Turing differente

L'algoritmo è l'insieme di passi per risolvere una classe di problemi

Definisce un livello di calolabilità quindi se la macchina di Turing non riesce a risolvere il problema allora è impossibile

Il nastro della macchina di Turing è la RAM infatti il calcolatore odierno carica prima il programma in RAM e poi lo esegue

La macchina di Turing Universale esegue un'altra macchina di Turing

La macchina di Turing Universale memorizza i dati nello stesso posto

# Lezione 4 – Architettura di un elaboratore

*30 settembre 2025*

La macchina di Von Neuman costituisce l'architettura di un elaboratore

La macchina di Von Neuman è composta da unità di controllo, unità aritmetico-logica (ALU), memoria, input/output, bus.

Il bus ha la sua velocità

I bus non possono avere gigaheartz di velocità perché stanno fuori dal chip che contiene la CPU e perché sono dei pin

La cache è la memoria presente nella CPU in modo tale che la CPU non deve andare a recuperare dati fuori

Se non esco dal chip sono molto veloce

Esco dal chip solo quando bisogna interagire con l'uomo

La CPU contiene gli stati quindi recupera le informazioni dalla RAM che non sono presenti nella cache

Cache L1 è vicina al core

Cache L2 passava le informazioni a L1

Cache L3 stava vicina alla RAM

Non sapendo quale istruzioni vengono eseguite si sposta tutto in RAM

# Lezione 1 – Sintassi e Semantica

*30 settembre 2025*

La differenza tra sintassi e semantica è che la sintassi sono la forma quindi le regole e la struttura di un linguaggio mentre semantica è il significato quindi l'interpretazione e il senso del linguaggio

L’approccio generativo è utilizzato molto nella programmazione

L’espressioni matematiche hanno un linguaggio

I linguaggi di programmazione usano dei mix di notazioni differenti:

- Prefissa: l’operatore è indicato prima degli operandi
- Postfissa: l’operatore è indicato dopo degli operandi
- Infissa: l’operatore è indicato tra gli operandi

Il calcolatore preferisce prefissa e postfissa

Il numero di operandi di un operatore è detto arità

Il AST lega gli operandi ai operatori in modo astratto

Gli operandi possono essere espressioni

L’operatore differisce dai operandi

Il parse tree è la rappresentazione ad albero del risultato del parsing di un testo secondo una grammatica

Parsing è il processo di analizzare un testo (es. un file JSON, XML, sorgente Python) secondo una grammatica

# Lezione 2 – Grammatica

*30 settembre 2025*

La grammatica ci dice quali regole devo rispettare per costruire un codice

I terminali (token) sono i simboli di base del linguaggio.

La grammatica è una quanterna di 4 elementi

Le regole sono definite nel token o terminale

Ho una produzione quando da delta produco alfa seguendo una regola

La grammatica impone il parse tree quindi la grammatica è composta da quelle regole

Il parse tree mostra tutte le derivazioni, mentre l’AST è una sua versione più astratta e semplificata

I simboli non terminali sono le iniziali dei concetti

# Lezione 3 – BNF e Carte sintattiche

*30 settembre 2025*

Le espressioni regolari sono grammatiche che possono essere risolte con la macchina a stati

Si dice grammatica non contestuale (o libera dal contesto) perché la produzione di un simbolo non dipende dal contesto in cui si trova

Se nella produzione incominciano a produrre a destra e a sinistra è contestuale, perché lì contano i simboli vicini

In una grammatica contestuale ci sono più regole, e spesso l'applicazione di una regola porta a nuove condizioni che fanno scattare altre regole, come una reazione a catena

BNF è il metodo in cui il programmatore scrive la grammatica di quel linguaggio

La grammatica impone la sintassi

La sintassi è l'insieme dei simboli di base, cioè le parole ammesse dal linguaggio

La gerarchia di Chomsky classifica i linguaggi formali in quattro tipi (regolari, liberi dal contesto, contestuali e ricorsivamente enumerabili) in base alla potenza espressiva delle grammatiche che li generano

La grammatica è ambigua quando si può avere parse tree quindi che la stessa cosa la posso interpretare in due modi diversi

# Lezione 1 – Nozione di Algoritmo

*7 ottobre 2025*

Un algoritmo è una sequenza di istruzioni per risolvere un problema matematico in maniera interrativa

L'algoritmo deve essere descritto con il linguaggio dell'esecutore

L'algoritmo deve risolvere una classe di problemi e deve essere fatto con una serie finita di istruzioni

Proprietà dell'algoritmo:

- Non ambiguo = istruzioni chiare e con un solo significato;
- Finito = termina dopo un numero limitato di passi;
- Efficiente = risolve il problema con poche istruzioni e tempo ridotto;

Tra esecutore e algoritmo c'è il programma, cioè la traduzione dell'algoritmo in linguaggio comprensibile alla macchina

Nella fase di analisi abbiamo un problema dopo di che abbiamo la fase di soluzione quindi devo disegnare la macchina a stati, penso a che linguaggio utilizzare  e lo scrivo. Subito dopo lo eseguo e vedo se risolve il problema

Durante l'analisi trovo la somiglianza tra i vari problemi

Per catalogare i problemi:

- Individuazione dei dati in ingresso;
- Individuazione dei risultati desiderati;
- Individuazione di cosa lega ingressi e uscite;

Se trovo la soluzione di una sola istanza del problema, risolvo solo quel caso specifico.

Se invece trovo la soluzione del problema generale, posso risolvere tutte le possibili istanze di quel problema.

Bisogna sempre generalizzare il problema anche perché così facendo ho più sicurezza

# Lezione 3 – Flusso di un Algoritmo

*7 ottobre 2025*

Il flusso è l'ordine della sequenza dei passi di un algoritmo

Il flusso di un algoritmo è determinato da condizioni che possono generare ramificazioni (cioè percorsi diversi a seconda del risultato della condizione)

L'incognita matematica viene chiamata variabile in programmazione

La variabile è un contenitore di un valore che cambia

La variabile nella macchina di Turing è una cella

Nell'architettura di Von Neuman la variabile è la cella di una memoria RAM

Durante l'esecuzione la variabile può cambiare il suo valore

L'incognita matematica è teorica mentre la variabile è un oggetto fisico presente in un determinato posto

L'assegnamento è l'uguale matematico quindi assegna ad una variabile il valore di un risultato di una operazione

Ci sono delle condizioni che mi permettono di deviare il flusso

In un programma c'è il "salto" cioè permette di non eseguire in modo ordinato quindi dall'istruzione 1 passa all'istruzione 3, c'è anche la ripetizione quindi li faccio eseguire la stesso flusso per tot volte

Abbiamo anche il predicato di verità quindi logica di primo ordine (OR, AND e NOT)

Non tutto ciò che sembra una procedura è un algoritmo

Nella fase di analisi devo capire se il problema è risolvibile o meno

Le tipologie dei problemi sono:

- Decisionali;
- Ricerca;
- Numerazioni;
- Ottimizzazioni;

Il problema dell’arresto consiste nel fatto che non esiste un programma in grado di stabilire, in anticipo, se un altro programma terminerà o resterà in esecuzione all’infinito

Ho più programmi che linguaggi infatti tanti programma sono autoreferenziali

# Lezione 1 – Linguaggi per la descrizione di algoritmi

*7 ottobre 2025*

Ci sono dei linguaggi che disegnano il flusso come flow chart

UML è un altro metodo per scrivere un algoritmo

Il linguaggio naturale è ambiguo e in questo presente si sta tentando di far capire all’esecutore il linguaggio naturale

Il problema dello pseudo-codice ha il problema che per scriverlo bisogna sapere programmare e non è un linguaggio naturale. È simile a un programma

Nello pseudo-codice per catturare l’input utilizzo il comando read mentre per scrivere l’ouput utilizzo il comando write

Più il linguaggio si avvicina al programmatore più diventa di alto livello mentre più si avvicina alla macchina più diventa basso

Il linguaggio macchina è un linguaggio assembler che traduce quasi immediatamente in linguaggio binario

I linguaggi ad alto livello vengono convertiti in assembler e poi in linguaggio binario

L’esecutore non è in grado di fare le operazioni in RAM quindi trasporta i dati dalla RAM nella CPU e poi con quei numeri fa le operazioni

Per il controllo di flusso lo strumento migliore è il flow chart

La caratteristica più efficiente dell’algoritmo vuol dire che bisogna accorciare quell’algoritmo per risolvere il problema

Finché il problema è matematico allora la soluzione è matematico

Il programma è la sequenza finita di istruzioni

La macchina RAM è un modello di macchina è utilizzata per calcolare la complessità di un programma

La macchina RAM utilizza le seguenti istruzioni:

- Read
- Write
- Assegnamento
- Selezione
- Salto

Non si possono dare nomi alle variabili nella macchina RAM, ma si accede direttamente alla cella della RAM tramite il comando mem

La programmazione non strutturata è una programmazione che lascia la massima libertà sul flusso quindi usa i salti incondizionati

L'unico programma non strutturato al giorno d'oggi è l'assembler

Il flusso è il cuore del programma

I salti incondizionati sono quei salti che sono più evidenti perché non hanno una condizione

l salto può essere indicato anche come goto nello pseudo-codice

Il goto è ancora presente in alcuni linguaggi come il C, ma oggi è poco usato perché può rendere il codice disordinato e difficile da seguire

I paradigmi di programmazione descrivono come risolvere un problema e come organizzare il programma

Nel paradigma imperativo, il programma è statico (le istruzioni sono fissate nel codice), mentre la computazione è dinamica (cioè i dati e lo stato del programma cambiano durante l'esecuzione)

Il tipo di una variabile è:

- la classe di valori che questa può assumere
- L'insieme dell'operazioni che possono essere effettuate su di essa
- Quanto spazio di memoria occupata
- Come viene codificata il dato all'interno della memoria

La variabile è il riferimento mnemonico all'indirizzo della locazione di memoria

Per assegnare il valore a una variabile viene utilizzato il comadno stone in assembler

Il tipo fornisce un'astrazione rispetto alla rappresentazione effettiva dei dati quindi dice all'elaboratore come interpretare quel dato

Nell'assembler non è presente il tipo

Tutto quello che voglio utilizzare a livello di variabili lo devo dichiarare prima

La variabile si può dichiarare in modo esplicita (come C) o implicita (come Python)

In alcuni linguaggi come Pascal devi dichiarare le variabili all'inizio del programma mentre altri come Java l'importante è dichiararli, non dove

I vantaggi della dichiarazione sono:

- Leggibilità del programma;
- Diminuisce la possibilità di errori;
- Facilita la traduzione efficiente per il compilatore

Per evitare che la variabile cambi si usano le costanti

Un letterale è un valore fissato nel programma (es. 5, 'a', true)

Una costante è una variabile il cui valore non può essere modificato durante l'esecuzione del programma; si dichiara con un modificatore come const o final

La costante è un modificatore di tipo

Un valore booleano rappresenta un'espressione logica e può assumere solo due valori: vero o falso

# Lezione 4 – Programmazione Strutturata

*8 ottobre 2025*

La programmazione strutturata critica il salto incondizionale

Il primo a evidenziare questa cosa fu Dijkstra

L'obiettivo della programmazione strutturata è avere maggiore controllo sui flussi di esecuzione e rendere il codice più leggibile.

La sequenza rappresenta il flusso "perfetto", cioè l'esecuzione ordinata delle istruzioni

La programmazione strutturata utilizza blocchi strutturati e non salti (goto)

Tutti i programmi si possono scrivere combinando tre blocchi fondamentali:

- Selezione quindi l'unico modo per ramificare un flusso:
- Sequenza quindi mettere un flusso uno dietro l'altro
- Iterazione è quando devo ripetere tante volte una determinata soluzione

La sequenza è che quando l'esecutore ha eseguito un'istruzione deve passare a quella dopo

Una sequenza può essere anche tra blocchi

Le iterazioni possono essere post-condizione e pre-condizione

# Lezione 5 – Eliminazione dei salti

*8 ottobre 2025*

Un programma proprio è un programma che ha solo un ingresso e sono un'uscita

Se dati gli stessi input danno lo stesso output si dicono equivalenti

Con il teorema di Böhm-Jacopini si dice che, dato un programma qualsiasi P, è possibile costruire un programma strutturato S(P) equivalente a P.
In poche parole, hanno dimostrato che un programma strutturato ha la stessa potenza di calcolo di un programma non strutturato

Il teorema di Böhm-Jacopini è stato dimostrato grazie al fatto che hanno preso un programma non strutturato e lo hanno riscritto usando solo tre costrutti: sequenza, selezione e iterazione (while)

Grazie al teorema di Böhm-Jacopini abbiamo capito che un programma non strutturato può essere trasformato in un programma strutturato, anche se questo comporta un aumento del numero di variabili e di istruzioni, quindi il programma peserà di più

Il teorema di Ashcroft e Manna sostiene che bisogna analizzare il flusso del programma e modificare solo le parti non strutturate, aggiungendo in quei punti dei pezzi strutturati

La differenza è che per Böhm-Jacopini bisognava trasformare tutto il programma, mentre per Ashcroft e Manna era sufficiente cambiare solo i pezzi non strutturati

I break nelle condizioni e nei cicli non fanno parte della programmazione strutturata, mentre fa parte solo il break dello switch

La correttezza del codice si basa sui concetti di invariante e asserzione, che servono per verificare che il programma funzioni correttamente

L'invariante è una condizione che rimane sempre vera durante l'esecuzione di una parte del programma, come ad esempio dentro un ciclo e serve per dimostrare che il ciclo si comporta come previsto.

L'asserzione è una condizione che si verifica in un punto preciso del programma (prima o dopo una certa istruzione) per controllare che il valore di una variabile sia corretto.

Il teorema di Böhm-Jacopini mostra anche che i linguaggi di programmazione sono Turing-equivalenti, cioè hanno la stessa potenza espressiva della macchina di Turing.

Se nel codice devo testare perché in quel ciclo sono uscito allora c'è qualcosa che non quadra

C'è un teorema che dice che il programma non strutturato è più efficiente di quello strutturato

Il linguaggio per eccelenza non strutturato è l'assembler ed è il più efficiente di tutti

# Lezione 5 – Introduzione al Linguaggio C

*14 ottobre 2025*

C possiede molte librerie a basso livello e ragiona come l'esecutore

Tutto il programma viene visto come un'intera funzione di nome main

Nella funzione principale si chiama main perché come prima cosa viene lanciato il primo comando che si trova dentro al main

I file che contengo il programma si chiamano anche sorgenti

La traduzione non è da C a binario, ma è una traduzione intermedia

I comandi in assembly vengono tradotti in linguaggio macchina, che è eseguito direttamente dalla CPU

La direttiva #include in C serve a includere il contenuto di un altro file nel punto in cui è scritta, prima della compilazione

C non fa controlli durante l'esecuzione, ma solo durante l'esecuzione

Getchar() legge un singolo carattere da tastiera. Dopo aver digitato un carattere e premuto Invio, nel buffer ci sono due caratteri: quello inserito dall'utente e il newline \n generato dall'Invio

Nel C c'è la convenzione che il numero 0 è falso e qualsiasi altro numero è vero

# Lezione 5 – Programmazione in Linguaggio C

*14 ottobre 2025*

Una variabile ha un'esistenza limitata: esiste solo all'interno del suo scope e viene distrutta quando lo scope termina. Lo scope è l'ambito di visibilità della variabile, cioè la parte di codice in cui può essere utilizzata

Esistono due tipi di scope:

- Globale quindi esiste in tutto il programma;
- Locale quindi esiste solo in quel blocco;

Lo scope è importante per tenere controllato cosa cambia il programma

Molti operatori hanno comportamento simile tra tipi numerici, ma alcuni operatori hanno comportamenti diversi a seconda del tipo della variabile

Esempio: + tra interi fa somma numerica, + tra stringhe può fare concatenazione

Gli operatori aritmetici possono comportarsi diversamente a seconda dei tipi. Quando si esegue un'operazione tra tipi diversi, il compilatore effettua una promozione dei tipi, convertendo il tipo più piccolo nel tipo più grande. Il tipo del risultato dell'operazione è determinato dai tipi degli operandi e dall'operatore

Anche char è un numero più piccolo di 8 bit

Promuovo sempre il più piccolo nel più grosso in modo tale da non avere problemi di overflow e underflow

l tipo del return deve corrispondere al tipo dichiarato della funzione

Un tipo di una variabile ci dice cosa contiene, l'occupazione in memoria, come viene codificato quel numero in memoria, l'operazioni che si possono fare e i valori che può assumere

Un identificatore non può mai iniziare con una cifra

L'assegnamento è un'operatore e ha l'effetto di generare un valore

L'assegnamento ha anche un effetto secondario (side effect) ovvero modifica il valore dell'operando a sinistra e questo effetto si manifesta subito dopo la valutazione del valore di =

Isdigit è una funzione della libreria (ctype.h) e serve a controllare se l'input dell'utente è una cifra o no

Dentro alle condizioni posso inserire anche la chiamata alle funzioni

I file header (.h) contengono le dichiarazioni delle funzioni (prototipi), cioè indicano che esistono e come si chiamano (tipo di ritorno, parametri, ecc.)

Gli assegnamenti composti servono a velocizzare la scrittura del codice

Esempio di assegnamento composto:

+=  -=  *=  /=

Ci sono altri comandi per velocizzare la scrittura del codice e sono incremento (++) e decremento (- -), possono essere prefissi e postfissi. Se sono postfissi vengono eseguiti per ultimi prima di passare alla nuova istruzione mentre se sono prefissi verranno eseguiti subito

# Lezione 6 – Programmazione in Linguaggio C

*15 ottobre 2025*

Quando l'utente digita qualcosa viene salvato in un buffer fino a quando nessuno lo rimuove

Lo scanf di solito elimina gli spazi bianchi presenti nel buffer

Lo scanf cerca nel buffer un pattern quindi dei numeri, caratteri, etc... E si prende tutto quello che riconosce

L'operatore ternario è un operatore condizionale che permette di scrivere in modo compatto una struttura if–else

Il break nello switch non è obbligatoria dal punto di vista sintattico: il programma compila anche senza. Tuttavia, se si omette, il flusso continua automaticamente nel caso successivo (fall-through).

Se nel for non inserisco la condizione quindi la seconda istruzione presente nelle paratensi tonde verrà assunta come vera e questo potrebbe causare un ciclo infinito

Se io nella condizione del for inserisco la dichiarazione di i verrà visto solo nel for e non in tutto il programma

L'operatore virgola in C serve per combinare più espressioni in una sola istruzione; le valuta da sinistra a destra e restituisce il valore dell'ultima

L'istruzione continue provoca il passaggio all'iterazione successiva, ignorando il resto del corpo e come istruzione andrebbe evitata

È meno grave del break perché break fa uscire totalmente dalla struttura mentre il continue no

La stringa è un array di caratteri con il terminatore di stringa

Il vettore è la dimensione nota a priori

# Lezione 8 – Funzioni in C

*28 ottobre 2025*

L'effetto shadowing si verifica quando due variabili locali, con lo stesso nome, sono dichiarate in scope (blocco di codice) diversi, uno all'interno dell'altro. Questo può causare ambiguità, in quanto la variabile più interna "oscurerà" quella esterna, rendendo difficile distinguere quale delle due variabili venga effettivamente utilizzata

Parametri formali sono i nomi dei parametri indicati nel prototipo mentre i parametri attuali sono i valori indicati nelle chiamate

Le macro sono definizioni che vengono fatte con il comando #define e sono gestite dal pre-processore prima che il codice venga effettivamente compilato.

# Lezione 8 – Call and Return

*28 ottobre 2025*

Lo stack è una zona di memoria associata a un processo, utilizzata per salvare le variabili locali e le informazioni relative agli scope (ambiti di visibilità) delle funzioni.

La struttura stack agisce come LIFO (Last In First Out) quindi il primo elemento è il main

Tutti i dati locali (come le variabili locali e i parametri delle funzioni) vengono allocati nello stack.

Nello stack posso vedere solo l'ultimo blocco aggiunto

L'insieme di oggetti che vengono caricati nello stack ad ogni chiamata si chiama record di attivazione

Lo stack contiene quindi tutte le strutture dati statiche di un programma

Si può generare uno stack overflow

Una funzione polimorfica è una funzione in grado di operare con tipi di dati differenti.

Se una variabile è associata a un tipo in modo stabile (cioè non cambia durante l'esecuzione), si parla di binding statico.
Al contrario, quando il tipo di una variabile può cambiare durante l'esecuzione del programma (come in alcuni linguaggi dinamici), si parla di binding dinamico.

Il controllo del tipo può essere:

- Statico, se avviene prima della compilazione (ad esempio in C o Java);
- Dinamico, se avviene durante l'esecuzione del programma (ad esempio in Python).

Se voglio controllare a runtime (durante l'esecuzione) che una variabile non vada in overflow o non provochi errori di tipo, devo utilizzare un controllo dinamico del tipo.

In C, ogni variabile e ogni valore ha un tipo (come int, float, char, ecc.) che determina la quantità di memoria occupata e le operazioni consentite.
Il linguaggio C usa un controllo dei tipi statico, cioè il tipo di ogni variabile è stabilito durante la compilazione, non a programma in esecuzione.

Il C:
– permette il casting
– permette di generare i puntatori

Il costrutto union permette di creare una struttura dati di tipo strutturato, ma è costituito da un solo campo alla volta (tutti i campi condividono la stessa area di memoria).

Assembler è il linguaggio meno tipizzato di tutti.

Il record di attivazione rappresenta lo scope in quel momento, cioè lo stato dello stack della funzione attiva.

Il passaggio dei parametri avviene in due modi in C:
– Call by value: viene passata una copia del parametro, quindi la funzione non modifica il valore originale.

– Call by reference: la funzione lavora sui valori originali perché gli viene passato l'indirizzo di memoria.

Si possono usare i parametri formali come variabili locali

Il passaggio per valore non ha effetti collaterali, cioè la variabile non viene modificata al di fuori del suo scope.

Il passaggio per riferimento vuol dire passare il riferimento (indirizzo di memoria) della variabile.

Con lo scanf si passa il riferimento (non il valore), perché la funzione deve modificare direttamente la variabile.

Il passaggio per riferimento genera quindi un effetto collaterale, cioè la variabile viene modificata anche al di fuori del suo scope.

Per ottenere l'indirizzo di memoria sono stati introdotti due operatori:

- & = estrae l'indirizzo di memoria di una variabile.
- * = permette di accedere al valore contenuto in quell'indirizzo di memoria.

Nelle funzioni può ritornare solo un elemento, se voglio ritornare più elementi devo farlo attraverso i parametri passati per indirizzo

Altri passaggi di parametro sono:

- Call by result: il parametro viene usato solo per restituire un valore.
  Tutto quello che viene fatto dentro la funzione viene copiato all'esterno solo dopo che la funzione termina.
- Call by value-result: è una combinazione tra call by value e call by result.
  All'inizio viene copiato il valore originale all'interno della funzione, e alla fine il risultato modificato viene copiato di nuovo all'esterno.
- Call by name: si passa il nome del parametro, e il suo valore viene ricalcolato ogni volta che viene usato all'interno della funzione.

La differenza tra passaggio per indirizzo e call by result è che:

- nel passaggio per indirizzo la variabile viene modificata subito, sia all'interno che all'esterno della funzione (effetto immediato);
- nel call by result le modifiche avvengono solo all'interno, e alla fine vengono copiate all'esterno.

Il C ha solo call by value

# Lezione 9 - Puntatori e Strutture

*29 ottobre 2025*

Un puntatore è un tipo

Il puntatore è grosso come un indirizzo

In memoria, un puntatore è rappresentato come un numero intero; di conseguenza, molte delle operazioni valide per i tipi numerici sono applicabili anche ai puntatori.

Il puntatore è sempre grande 4 celle di memoria

L’operatore * permette di accedere al valore memorizzato nell’indirizzo a cui il puntatore fa riferimento, consentendo anche di modificarlo.

L’aritmetica dei puntatori consiste nel fare operazioni sugli indirizzi contenuti nei puntatori.

Un puntatore si può inizializzare ad un valore

Una union è simile a una struct, ma tutti i suoi campi condividono lo stesso spazio di memoria, quindi può contenere un solo valore alla volta.

Con l’union il compilatore alloca solo lo spazio al più grosso e gli altri si sovrappongono

L’union si utilizza principalmente per risparmiare spazio perché ha diversi tipi di dato nella stessa locazione di memoria

L’union si utilizza come le struct

Esempio: in un gioco, un personaggio può sparare o camminare, ma non fare entrambe le azioni contemporaneamente. Si può usare una union per rappresentare questa scelta, memorizzando solo l’azione attiva.