Programmazione

Indice:

Lezione 1- Formalizzazione di un Linguaggio

24 settembre 2025

Il linguaggio utilizzato per programmare è un linguaggio formale, cioè segue regole precise e definite. A differenza del linguaggio naturale, non ammette ambiguità e ogni istruzione deve rispettare una sintassi ben stabilita.

Il linguaggio naturale, invece, può essere analizzato su tre livelli:

- il livello pragmatico, che riguarda lo scopo e l'obiettivo del messaggio, cioè ciò che voglio ottenere con ciò che scrivo;
- il livello semantico, che riguarda il significato delle parole o delle istruzioni;
- il livello sintattico, che definisce le regole di combinazione delle parole o, nel caso della programmazione, le conseguenze e la struttura corretta di un programma.

Una stringa è una sequenza finita di simboli appartenenti a un determinato alfabeto. Di una stringa si può determinare la lunghezza, cioè il numero di simboli che contiene. Esiste anche la stringa vuota, che non contiene alcun simbolo e ha lunghezza pari a zero.

La chiusura di Kleene (o Kleene Star) è un'operazione che permette di generare tutte le possibili stringhe che si possono formare a partire da un determinato alfabeto, comprese le stringhe vuote.

Due stringhe possono essere unite tramite l'operazione di concatenazione, che consiste nel mettere una stringa subito dopo l'altra per formarne una nuova. È possibile effettuare anche concatenazioni multiple, unendo più stringhe in sequenza.

Da una stringa è inoltre possibile estrarre una sottostringa, cioè una parte della stringa originale composta da simboli contigui. La sottostringa conserva l'ordine dei simboli presenti nella stringa di partenza.

La chiusura positiva (detta anche Kleene Plus) è simile alla chiusura di Kleene, ma con una differenza importante: non genera la stringa vuota. In altre parole, rappresenta tutte le possibili combinazioni dei simboli dell'alfabeto con lunghezza maggiore di zero (n > 0).

Un linguaggio formale è un insieme di parole (o stringhe) costruite a partire da un determinato alfabeto e che rispettano specifiche regole di formazione.

Non tutti i linguaggi hanno un elenco finito di parole, come invece accade per esempio con le password, che possono generare un insieme potenzialmente infinito di combinazioni.

L'insieme Σ* (Sigma Star) rappresenta tutte le parole possibili, comprese quelle vuote, che si possono formare con i simboli di un determinato alfabeto. Alcuni linguaggi, invece, possono essere vuoti, cioè non contenere alcuna parola.

La cardinalità di un linguaggio indica quante stringhe lo compongono, cioè il numero di elementi presenti nel linguaggio.

Un linguaggio diventa un codice quando è univocamente decifrabile, cioè quando ogni messaggio può essere scomposto in un solo modo, senza ambiguità.

Per esempio, il linguaggio C = {1, 10} è un codice perché le parole sono completamente diverse e non si possono confondere.
Invece, C = {bab, aba, ab} non è un codice, perché esistono frasi che si possono interpretare in più modi.
Ad esempio, la stringa ababab può essere ottenuta sia come aba + bab sia come ab + ab + ab, quindi non è univocamente decifrabile.

Il codice è un linguaggio univocamente decifrabile.

I virus informatici possono combinarsi tra loro, dando origine a nuove varianti più complesse e difficili da riconoscere.

La potenza di un linguaggio rappresenta la sua capacità di riconoscere o generare un insieme di parole (che può anche essere infinito), verificando per ciascuna parola se rispetta le regole di quel linguaggio.

L'approccio riconoscitivo consiste nell'utilizzare una procedura algoritmica per stabilire se una determinata parola appartiene o meno a un linguaggio.
L'approccio generativo, invece, parte dal linguaggio stesso e verifica se una parola può essere generata seguendo le regole che lo definiscono: se non riesco a generarla, significa che non appartiene a quel linguaggio.

Un linguaggio regolare è un linguaggio che segue regole precise e ben definite, che ne descrivono la struttura.

È inoltre possibile tradurre un linguaggio in un altro. Un esempio tipico è quello del compilatore, che traduce il linguaggio di programmazione C in linguaggio macchina (binario), comprensibile dal computer.

Lezione 2 – Automi a Stati Finiti

24 settembre 2025

Un automa a stati finiti è un modello matematico che, dato un ingresso, fornisce una determinata uscita

Un automa a stati finiti può essere descritto formalmente come una quintupla

Un sistema può trovarsi in diverse situazioni, e queste situazioni vengono chiamate stati. Ogni stato rappresenta un insieme di informazioni che descrivono come si trova il sistema in un determinato momento.

Un esempio semplice è quello del semaforo. Il semaforo può trovarsi in tre stati: verde, giallo e rosso. L'ingresso che determina il cambiamento di stato è il passare del tempo, ad esempio ogni trenta secondi. Le transizioni tra gli stati avvengono in modo ciclico: se il semaforo è verde e passa il tempo, diventa giallo; se è giallo e passa il tempo, diventa rosso; se è rosso e passa il tempo, torna verde.

Allo stesso modo, anche un computer quando viene avviato si trova in uno stato iniziale e attende un input. A seconda dell'input ricevuto, può cambiare stato ed entrare in una nuova situazione.

Questo comportamento si può rappresentare attraverso un modello chiamato automa a stati finiti. Gli automi servono per modellare sistemi che cambiano nel tempo in base a determinati ingressi. In una rappresentazione grafica, ogni stato è indicato con un cerchio, mentre le transizioni tra stati sono rappresentate da frecce che indicano il passaggio da uno stato all'altro. Sopra ogni freccia viene scritto l'input che causa quel passaggio. Se un'automa rimane nello stesso stato per un certo input, si rappresenta con una freccia che parte e torna sullo stesso cerchio, chiamata auto-anello.

Un automa può essere visto come una black box (scatola nera): riceve degli input, cambia stato e produce un certo comportamento visibile dall'esterno. Tutti gli stati e le transizioni tra di essi formano un grafo, e questo grafo descrive il linguaggio delle soluzioni che l'automa può rappresentare.

Esistono due tipi principali di automi: deterministici e non deterministici.
Un automa deterministico (DFA) è quello in cui, da ogni stato e per ogni simbolo dell'alfabeto, esiste al massimo una sola transizione possibile.
Un automa non deterministico (NFA), invece, può avere più transizioni possibili dallo stesso stato con lo stesso simbolo. Questo tipo di automa viene usato per rappresentare situazioni in cui il comportamento del sistema non è sempre identico, ma può variare, ad esempio in base a scelte o probabilità.

Gli automi hanno alcune proprietà fondamentali:

- Dinamicità, perché descrivono sistemi che si evolvono nel tempo passando da uno stato all'altro.
- Discretezza, perché sia gli stati che le transizioni sono in numero finito.

Esistono due principali tipi di macchine a stati finiti che generano un output: la macchina di Mealy e la macchina di Moore.

La macchina di Mealy produce l’output durante la transizione da uno stato all’altro, cioè l’output dipende sia dallo stato corrente sia dall’input ricevuto.

La macchina di Moore, invece, genera l’output in base allo stato in cui si trova, quindi l’output dipende solo dallo stato attuale e non direttamente dall’input.

Lezione 3 – Macchina di Turing

30 settembre 2025

La macchina di Turing è un modello teorico ideato da Alan Turing per rappresentare il funzionamento logico di un calcolatore. È una macchina a stati capace di spostarsi da uno stato all’altro in modo automatico, senza l’intervento umano diretto.

Turing la progettò per studiare la calcolabilità e la complessità dei problemi, cioè per capire quali problemi possono essere risolti da una macchina e quanto sono difficili da risolvere. L’unico compito dell’utente è fornire l’input, mentre la macchina esegue il calcolo da sola.

Il suo funzionamento si basa su un nastro infinito, che rappresenta la memoria. Su questo nastro la macchina può leggere e scrivere simboli tramite una testina che si muove a destra o a sinistra. Ogni volta che legge o scrive un simbolo, la macchina può cambiare stato in base alle regole definite.

Il nastro contiene i dati e l’output del calcolo, mentre la testina e lo stato interno rappresentano la logica di controllo della macchina.

Successivamente, è stata ideata anche una versione multinastro e non deterministica, in cui la macchina dispone di più nastri: per esempio, uno per l’input, uno per l’elaborazione e uno per l’output. Questo modello serve per studiare più a fondo le prestazioni teoriche e la complessità computazionale dei problemi.

La macchina di Turing può essere rappresentata come un tupla.

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

La macchina di Turing ha l’output sulla transizione, cioè produce il risultato mentre cambia stato in base all’input letto e alle regole di funzionamento.

Per ogni problema esiste una macchina di Turing specifica, costruita apposta per risolverlo. Tuttavia, la macchina di Turing universale è un caso speciale: è una macchina in grado di simulare qualsiasi altra macchina di Turing, quindi può eseguire qualunque programma rappresentabile.

Un algoritmo è un insieme ordinato di passi logici che permettono di risolvere una determinata classe di problemi. La macchina di Turing serve proprio a definire il concetto di calcolabilità, cioè a stabilire se un problema può essere risolto in modo meccanico da una macchina. Se un problema non può essere risolto nemmeno da una macchina di Turing, allora è considerato non calcolabile.

Il nastro della macchina di Turing può essere paragonato alla RAM di un computer moderno: entrambi rappresentano la memoria di lavoro. Anche oggi, infatti, un calcolatore carica il programma in RAM prima di eseguirlo.

La macchina di Turing universale, inoltre, utilizza un unico nastro per memorizzare sia i dati che il programma.

Lezione 4 – Architettura di un elaboratore

30 settembre 2025

La macchina di Von Neumann rappresenta il modello base dell'architettura dei calcolatori moderni. È l'organizzazione logica di un elaboratore, cioè il modo in cui sono strutturate e collegate tra loro le sue principali componenti.

Questa architettura è composta da cinque elementi fondamentali:

- Unità di controllo (Control Unit): gestisce il flusso delle istruzioni e coordina le varie parti del sistema, decidendo cosa deve essere eseguito e in quale ordine.
- Unità aritmetico-logica (ALU): esegue le operazioni matematiche (somma, sottrazione, moltiplicazione…) e logiche (AND, OR, NOT…) sui dati.
- Memoria: contiene sia i dati sia le istruzioni del programma in esecuzione.
- Dispositivi di input/output (I/O): permettono lo scambio di informazioni tra il computer e l'esterno, come tastiera, monitor, disco, stampante, ecc.
- Bus: sono i canali di comunicazione che collegano tutte le unità, consentendo il trasferimento di dati, indirizzi e segnali di controllo.

Il bus è il canale di comunicazione che collega i vari componenti del computer, e come ogni canale, ha una propria velocità di trasferimento. Tuttavia, i bus non possono raggiungere velocità nell'ordine dei gigahertz come la CPU, perché si trovano all'esterno del chip e sono costituiti da pin fisici che limitano la velocità di trasmissione.

La cache è una memoria molto veloce situata all'interno della CPU. Il suo scopo è ridurre i tempi di accesso ai dati, evitando che la CPU debba continuamente recuperare informazioni dalla RAM, che è più lenta. Quando la CPU trova i dati nella cache, può elaborarli rapidamente; se invece non li trova, li recupera dalla RAM e li memorizza nella cache per i futuri utilizzi.

Finché la CPU lavora all'interno del chip, le operazioni sono estremamente veloci. Deve invece uscire dal chip solo quando deve interagire con dispositivi esterni, come monitor, tastiera o memoria di massa, il che rallenta l'elaborazione.

All'interno della CPU esistono diversi livelli di cache, ognuno con una posizione e una funzione specifica:

- La cache L1 è la più piccola e veloce, ed è direttamente collegata al core del processore.
- La cache L2 è più grande ma leggermente più lenta, e fornisce dati alla L1.
- La cache L3 è ancora più ampia e si trova più vicina alla RAM, fungendo da collegamento tra la memoria principale e le cache interne.

Infine, poiché la CPU non può prevedere quali istruzioni verranno eseguite in futuro, il sistema mantiene le informazioni nella RAM, in modo che possano essere caricate nella cache quando necessario.

Lezione 1 – Sintassi e Semantica

30 settembre 2025

Differenza tra sintassi e semantica:

- Sintassi: riguarda la forma, cioè le regole e la struttura del linguaggio (come devono essere scritte le istruzioni).
- Semantica: riguarda il significato, cioè l'interpretazione e il senso di ciò che è scritto.

L'approccio generativo è molto utilizzato nella programmazione, perché consente di costruire strutture complesse (come espressioni o programmi) a partire da regole semplici e ben definite.

Le espressioni matematiche e i linguaggi di programmazione hanno un proprio linguaggio formale, cioè un insieme di simboli e regole che determinano come devono essere scritte e interpretate.

Nei linguaggi di programmazione si utilizzano diverse notazioni per rappresentare le operazioni:

- Prefissa: l'operatore è scritto prima degli operandi (es. + 3 4);
- Postfissa: l'operatore è scritto dopo gli operandi (es. 3 4 +);
- Infissa: l'operatore è scritto tra gli operandi (es. 3 + 4).

Il calcolatore preferisce le notazioni prefissa e postfissa, perché sono più facili da interpretare senza ambiguità, a differenza di quella infissa che richiede l'uso di parentesi o regole di precedenza.

Il numero di operandi che un operatore utilizza è chiamato arità (ad esempio, l'operatore "+" è binario perché usa due operandi).

L'AST (Abstract Syntax Tree), o albero di sintassi astratta, è una struttura che collega gli operatori ai loro operandi in modo logico e gerarchico, rappresentando la struttura sintattica del codice senza dettagli superflui.

Gli operandi possono a loro volta essere espressioni, e si distinguono dagli operatori, che rappresentano le azioni o le operazioni da eseguire.

Infine, il parse tree (albero di parsing) è la rappresentazione ad albero del risultato dell'analisi sintattica di un testo secondo una determinata grammatica.

Il processo che costruisce questo albero si chiama parsing, e consiste nell'analizzare un testo (ad esempio un file JSON, XML o un sorgente Python) per verificare se rispetta le regole grammaticali del linguaggio e per comprenderne la struttura.

Lezione 2 – Grammatica

30 settembre 2025

La grammatica di un linguaggio definisce l'insieme delle regole che bisogna rispettare per poter scrivere un codice corretto. È una quaterna composta da quattro elementi principali: l'insieme dei simboli terminali, l'insieme dei simboli non terminali, il simbolo iniziale e l'insieme delle regole di produzione.

I simboli terminali, chiamati anche token, rappresentano le unità di base del linguaggio, cioè gli elementi che non possono essere ulteriormente scomposti, come ad esempio parole chiave,

operatori, numeri o identificatori. I simboli non terminali invece indicano concetti o strutture del linguaggio e vengono solitamente rappresentati con lettere maiuscole.

Le regole di produzione stabiliscono come si possono sostituire o combinare i simboli per formare frasi o espressioni corrette. Si parla di produzione quando, partendo da un simbolo, si ottiene una sequenza seguendo una regola della grammatica.

Il parse tree, o albero di derivazione, rappresenta graficamente tutte le derivazioni possibili di un testo secondo la grammatica. Da questo albero si può poi ottenere l'AST (Abstract Syntax Tree), che è una sua versione più astratta e semplificata, utile per rappresentare solo la struttura logica del programma, eliminando i dettagli sintattici superflui. In sintesi, la grammatica impone la struttura del linguaggio e permette di costruire e interpretare correttamente il parse tree e, di conseguenza, l'AST.

Lezione 3 – BNF e Carte sintattiche

30 settembre 2025

Le espressioni regolari rappresentano un tipo di grammatica che può essere risolta attraverso una macchina a stati, cioè un modello in grado di riconoscere sequenze di simboli seguendo determinate regole di transizione.

Una grammatica non contestuale (o libera dal contesto) è una grammatica in cui la produzione di un simbolo non dipende dal contesto in cui si trova, ma solo dalla regola applicata. Al contrario, in una grammatica contestuale, la produzione di un simbolo dipende dai simboli che si trovano vicino a esso: le regole considerano quindi sia ciò che precede che ciò che segue. Questo tipo di grammatica è più complessa, perché l'applicazione di una regola può generare nuove condizioni che attivano altre regole, creando una sorta di "reazione a catena".

La BNF (Backus-Naur Form) è un metodo standard utilizzato per descrivere formalmente la grammatica di un linguaggio. Permette al programmatore di specificare in modo chiaro e rigoroso le regole sintattiche che definiscono come devono essere scritti i comandi o le espressioni di quel linguaggio.

In sintesi, la grammatica definisce la sintassi di un linguaggio, cioè l'insieme dei simboli, delle parole e delle regole che determinano quali frasi o istruzioni sono corrette dal punto di vista formale.

La gerarchia di Chomsky è una classificazione dei linguaggi formali in quattro categorie, ordinate in base alla potenza espressiva delle grammatiche che li generano, cioè alla loro capacità di descrivere linguaggi sempre più complessi.

I quattro tipi di linguaggi sono:

1. Linguaggi regolari, generati da grammatiche regolari, i più semplici e quelli che possono essere riconosciuti da automi a stati finiti.
2. Linguaggi liberi dal contesto, generati da grammatiche non contestuali, che possono essere riconosciuti da automi a pila.
3. Linguaggi contestuali, che richiedono grammatiche dipendenti dal contesto e automi più complessi per essere riconosciuti.
4. Linguaggi ricorsivamente enumerabili, i più potenti, che possono essere riconosciuti da una macchina di Turing.

Una grammatica è ambigua quando una stessa frase o stringa può essere interpretata in più modi diversi, cioè può generare più parse tree (alberi sintattici) differenti. Questo significa che la grammatica non è in grado di determinare un'unica struttura per quella frase, creando quindi ambiguità nel significato o nella struttura sintattica.

Lezione 1 – Nozione di Algoritmo

7 ottobre 2025

Un algoritmo è una sequenza finita di istruzioni che permette di risolvere un problema o di eseguire un calcolo in modo logico e ordinato.

Un algoritmo deve essere descritto utilizzando il linguaggio dell'esecutore, cioè un linguaggio che quest'ultimo sia in grado di comprendere ed eseguire. Ad esempio, se l'esecutore è un computer, l'algoritmo dovrà essere espresso in un linguaggio di programmazione o comunque in una forma formalmente interpretabile dalla macchina.

Inoltre, un algoritmo deve essere in grado di risolvere una classe di problemi, non solo un singolo caso specifico. Deve quindi essere generale, applicabile a diverse istanze dello stesso problema.

Infine, deve essere costituito da una serie finita di istruzioni, cioè un numero limitato di passi che, una volta eseguiti, portano sempre a una conclusione o a un risultato.

Un algoritmo deve possedere alcune proprietà fondamentali per essere considerato corretto ed efficace.

Innanzitutto deve essere non ambiguo, cioè le sue istruzioni devono essere chiare, precise e interpretabili in un solo modo, senza possibilità di confusione. Ogni passo deve essere comprensibile sia per l'uomo che per la macchina.

In secondo luogo, deve essere finito, ovvero deve terminare dopo un numero limitato di operazioni. Un algoritmo che non si conclude mai non può risolvere realmente un problema.

Infine, deve essere efficiente, cioè deve raggiungere il risultato nel minor tempo possibile e con il minimo numero di istruzioni o risorse. Un algoritmo efficiente permette di ottenere la soluzione in modo rapido e con un buon utilizzo delle capacità del calcolatore.

Tra l'esecutore e l'algoritmo si trova il programma, che rappresenta la traduzione dell'algoritmo in un linguaggio comprensibile dalla macchina. In altre parole, l'algoritmo descrive i passi logici per risolvere un problema, mentre il programma li trasforma in istruzioni eseguibili dal calcolatore.

Nel processo di analisi si parte dall'individuazione di un problema e si passa alla fase di soluzione, in cui si progetta una macchina a stati, si sceglie il linguaggio di programmazione più adatto e infine si scrive ed esegue il programma per verificare se effettivamente risolve il problema. Durante questa fase è utile osservare le somiglianze tra i vari problemi, poiché questo aiuta a riconoscere schemi comuni e a riutilizzare soluzioni già note.

Per catalogare un problema si devono seguire tre passaggi fondamentali:

1. Individuare i dati in ingresso, cioè le informazioni di partenza necessarie.

2. Definire i risultati desiderati, ovvero ciò che si vuole ottenere come output.
3. Stabilire il legame tra ingressi e uscite, ossia le regole o operazioni che collegano i dati iniziali ai risultati finali.

Se si trova la soluzione per una singola istanza del problema, si risolve solo quel caso specifico. Al contrario, se si trova la soluzione del problema generale, è possibile applicarla a tutte le sue varianti. Per questo motivo è sempre importante generalizzare il problema, perché ciò garantisce maggiore sicurezza, flessibilità e riutilizzabilità della soluzione.

Lezione 3 – Flusso di un Algoritmo

7 ottobre 2025

Il flusso rappresenta l'ordine con cui vengono eseguiti i passi di un algoritmo. In altre parole, è la sequenza logica delle istruzioni che portano alla risoluzione di un problema. Il flusso può essere lineare, ma spesso è determinato da condizioni che generano ramificazioni, ossia percorsi alternativi che dipendono dal risultato di una verifica logica.

In programmazione, l'incognita matematica assume il nome di variabile, cioè un contenitore che può memorizzare un valore e modificarlo durante l'esecuzione del programma. Nella macchina di Turing, una variabile corrisponde a una cella del nastro, mentre nell'architettura di Von Neumann essa è rappresentata da una cella della memoria RAM. La principale differenza rispetto all'incognita matematica è che la variabile è un elemento fisico, presente in un preciso punto della memoria, mentre l'incognita è un concetto puramente teorico.

L'assegnamento è l'operazione con cui si assegna un valore a una variabile: in termini matematici, equivale all'uguale, ma con il significato di "memorizzare" un valore, non di "confrontarlo".

Nel flusso di un programma, possono esserci condizioni che fanno deviare l'esecuzione, ossia cambiare percorso a seconda del risultato di un test logico. Inoltre, è possibile effettuare salti, cioè passare da un'istruzione a un'altra non in ordine sequenziale, oppure introdurre ripetizioni, facendo eseguire una stessa parte del programma più volte.

La logica di esecuzione si basa sui predicati di verità, cioè le operazioni logiche di AND, OR e NOT, che permettono di verificare condizioni e determinare i percorsi del flusso.

Non tutto ciò che sembra una procedura può essere considerato un algoritmo: per esserlo, deve rispettare determinate proprietà (chiarezza, finitezza, efficienza, ecc.) e risolvere effettivamente un problema.

Durante la fase di analisi, è fondamentale capire se il problema è risolvibile e che tipo di problema si sta affrontando. Le principali tipologie di problemi sono:

- Decisionali, quando si deve stabilire se una condizione è vera o falsa;
- Di ricerca, quando si deve trovare un elemento specifico;
- Di numerazione, quando si devono contare le possibili soluzioni;
- Di ottimizzazione, quando si deve individuare la soluzione migliore tra più alternative.

Il problema dell'arresto è un concetto fondamentale dell'informatica teorica, introdotto da Alan Turing. Esso afferma che non esiste un programma o un algoritmo capace di stabilire, in modo generale e automatico, se un altro programma terminerà la propria esecuzione oppure continuerà a funzionare all'infinito.

Esistono più programmi che linguaggi, perché un linguaggio di programmazione può generare un numero praticamente infinito di programmi diversi. Ogni programma, infatti, è una combinazione unica di istruzioni scritte secondo le regole di un linguaggio.

Inoltre, molti programmi sono autoreferenziali, cioè fanno riferimento a se stessi o si comportano in modo da analizzare o modificare il proprio codice. Questo è un concetto importante in informatica teorica, legato anche alle macchine di Turing universali e alla ricorsione: un programma può leggere, interpretare o persino eseguire se stesso, come accade nei compilatori o negli interpreti scritti nello stesso linguaggio che elaborano.

Lezione 1 – Linguaggi per la descrizione di algoritmi

7 ottobre 2025

Esistono linguaggi che permettono di rappresentare il flusso di esecuzione di un programma attraverso schemi grafici, come i flow chart, che mostrano in modo visivo i passaggi logici di un algoritmo.
Un altro metodo per rappresentare algoritmi e strutture software è l'UML (Unified Modeling Language), utilizzato soprattutto nella progettazione orientata agli oggetti.

Il linguaggio naturale è invece ambiguo, cioè può essere interpretato in più modi diversi. Tuttavia, negli ultimi anni si sta cercando di far comprendere il linguaggio naturale anche ai calcolatori, come avviene con i moderni sistemi di intelligenza artificiale.

Un metodo intermedio tra linguaggio naturale e linguaggio di programmazione è lo pseudo-codice, che serve per descrivere un algoritmo in modo più comprensibile per l'uomo, ma con una struttura simile a quella di un vero linguaggio di programmazione. Il limite dello pseudo-codice è che, pur non essendo un linguaggio formale, richiede comunque conoscenze di programmazione. Nello pseudo-codice, per catturare l'input si usa il comando read, mentre per produrre l'output si utilizza write.

Più un linguaggio si avvicina al modo di pensare del programmatore, più è detto di alto livello (come Python o Java). Viceversa, più si avvicina al linguaggio della macchina, più è di basso livello (come Assembly o linguaggio macchina). Il linguaggio macchina è formato da istruzioni binarie, mentre l'Assembly traduce quasi direttamente in linguaggio binario, fungendo da intermediario. I linguaggi ad alto livello, invece, vengono tradotti prima in Assembly e poi in codice binario eseguibile dalla CPU.

L'esecutore (cioè la CPU) non può operare direttamente sulla RAM: deve prima trasferire i dati dalla memoria alla CPU, dove vengono eseguite le operazioni.

Per rappresentare il controllo del flusso di un programma, lo strumento più chiaro e intuitivo rimane il flow chart. Un algoritmo è tanto più efficiente quanto meno istruzioni servono per risolvere un problema, mantenendo però correttezza e chiarezza.

Quando un problema è di tipo matematico, anche la sua soluzione sarà di natura matematica. Un programma è una sequenza finita di istruzioni che descrive come risolvere un problema.

Un modello utile per studiare la complessità di un programma è la macchina RAM, che simula il comportamento di un calcolatore. Essa utilizza un insieme di istruzioni fondamentali:

- Read per leggere un dato,
- Write per scrivere un risultato,

- Assegnamento,
- Selezione,
- Salto.

Nella macchina RAM non è possibile assegnare nomi alle variabili: si accede direttamente alle celle di memoria tramite il comando mem.

La programmazione non strutturata è quella che lascia la massima libertà nel controllo del flusso, utilizzando salti incondizionati che permettono di passare da un'istruzione a un'altra senza seguire un ordine sequenziale. L'unico linguaggio che oggi mantiene questa caratteristica in modo evidente è l'Assembly.

Il flusso rappresenta il cuore del programma, ovvero l'ordine con cui le istruzioni vengono eseguite. I salti incondizionati (detti anche goto) non dipendono da condizioni logiche e per questo motivo possono rendere il codice disordinato e difficile da seguire. Alcuni linguaggi, come il C, ancora li supportano, ma oggi sono usati raramente.

Infine, i paradigmi di programmazione descrivono i diversi modi in cui è possibile risolvere un problema e organizzare un programma — ad esempio, la programmazione procedurale, quella orientata agli oggetti, funzionale o logica.

Nel paradigma imperativo, il programma viene scritto come una sequenza di istruzioni che restano fisse nel codice, quindi possiamo dire che il programma è statico. Tuttavia, durante l'esecuzione, i dati e lo stato del programma possono cambiare continuamente, rendendo la computazione dinamica. In questo paradigma il programmatore specifica come il calcolatore deve svolgere un compito, indicando passo per passo le operazioni da eseguire.

Il tipo di una variabile, invece, definisce le caratteristiche del dato che essa può contenere. In particolare, stabilisce quale insieme di valori la variabile può assumere, quali operazioni possono essere eseguite su di essa, quanto spazio occupa in memoria e come il valore è rappresentato all'interno della memoria del computer. Ad esempio, una variabile di tipo intero (int) può contenere solo numeri interi, occupa una certa quantità di byte e viene codificata in modo diverso rispetto a una variabile di tipo reale (float) o a una variabile di tipo carattere (char).

La variabile rappresenta un riferimento mnemonico all'indirizzo di una specifica locazione di memoria: in altre parole, è un nome che usiamo per accedere più facilmente a un'area della memoria dove viene conservato un valore.
Quando vogliamo assegnare un valore a una variabile, nei linguaggi di basso livello come l'Assembler viene utilizzato il comando STORE, che serve proprio per memorizzare un dato in una certa posizione di memoria.

Il tipo di una variabile fornisce un'astrazione rispetto alla rappresentazione effettiva dei dati, cioè indica al calcolatore come deve interpretare quel valore in memoria. Tuttavia, nei linguaggi di basso livello come l'Assembler, non esistono i tipi: i dati sono solo sequenze di bit e il significato dipende da come vengono usati.

Nei linguaggi di programmazione, tutte le variabili che si vogliono utilizzare devono essere dichiarate prima del loro utilizzo. Questa dichiarazione può avvenire in due modi:

- Esplicita, come nel linguaggio C, dove bisogna indicare anche il tipo (es. int numero;);
- Implicita, come in Python, dove il tipo viene determinato automaticamente dal valore assegnato (es. numero = 5).

Alcuni linguaggi, come Pascal, richiedono di dichiarare tutte le variabili all'inizio del programma, mentre altri, come Java, permettono di dichiararle in qualunque punto, purché siano dichiarate prima del loro utilizzo.

I vantaggi della dichiarazione delle variabili sono diversi:

- migliora la leggibilità del programma, rendendo più chiaro il significato dei dati;
- riduce gli errori, poiché il compilatore può controllare il tipo e l'uso delle variabili;
- facilita il lavoro del compilatore, che può tradurre il programma in modo più efficiente.

Infine, quando vogliamo evitare che una variabile modifichi il proprio valore, possiamo utilizzare le costanti, che rappresentano dati fissi e immutabili durante l'esecuzione del programma.

Un letterale è un valore fissato direttamente nel programma, cioè un dato scritto esplicitamente nel codice. Ad esempio, i numeri come 5, i caratteri come 'a' o i valori booleani come true sono tutti letterali, perché rappresentano valori costanti già definiti e non legati a nessuna variabile.

Una costante, invece, è una variabile il cui valore non può essere modificato durante l'esecuzione del programma. In pratica, una costante viene dichiarata una sola volta e mantiene sempre lo stesso valore. Per definirla si usa un modificatore specifico, come const (in C o JavaScript) oppure final (in Java).

In sintesi, mentre il letterale è semplicemente un valore scritto nel codice, la costante è una variabile immutabile che conserva quel valore per tutta la durata del programma.

La costante è un modificatore di tipo che serve a rendere una variabile immutabile, cioè a impedire che il suo valore venga modificato durante l'esecuzione del programma. In questo modo, una volta assegnato il valore iniziale, esso rimane fisso per tutta la durata del programma, garantendo maggiore sicurezza e stabilità nel codice.

Un valore booleano, invece, rappresenta un'espressione logica e può assumere solo due stati possibili: vero (true) oppure falso (false). Questo tipo di dato è molto importante nei controlli di flusso, perché permette di prendere decisioni nei programmi, ad esempio nelle condizioni if o nei cicli while, dove il comportamento del programma dipende dal valore logico valutato.

Lezione 4 – Programmazione Strutturata

8 ottobre 2025

La programmazione strutturata nasce come critica all'uso del salto incondizionale (come il comando goto), che rendeva i programmi difficili da leggere e da mantenere. Il primo a mettere in evidenza questo problema fu Edsger Dijkstra, uno dei padri dell'informatica moderna.

L'obiettivo principale della programmazione strutturata è quello di avere un maggiore controllo sui flussi di esecuzione e di rendere il codice più chiaro e leggibile. In questo paradigma, l'esecuzione del programma avviene attraverso blocchi strutturati e non tramite salti arbitrari.

La sequenza rappresenta il flusso "perfetto" di esecuzione, ossia l'esecuzione ordinata delle istruzioni una dopo l'altra: una volta terminata un'istruzione, l'esecutore passa automaticamente a quella successiva. Le sequenze possono anche collegare più blocchi logici tra loro.

Tutti i programmi, secondo la teoria della programmazione strutturata, possono essere costruiti combinando tre strutture fondamentali:

- Selezione, che permette di ramificare il flusso a seconda di una condizione (ad esempio con istruzioni if o switch);
- Sequenza, cioè l'esecuzione lineare e ordinata delle istruzioni;
- Iterazione, che consente di ripetere una parte di codice più volte, finché una condizione è verificata.

Le iterazioni possono essere di due tipi:

- A pre-condizione, dove la condizione viene controllata prima di entrare nel ciclo (es. while);
- A post-condizione, dove la condizione viene controllata dopo l'esecuzione del blocco (es. do-while).

In sintesi, la programmazione strutturata punta a creare programmi più ordinati, modulari e facili da comprendere, eliminando la confusione causata dai salti incondizionati.

Lezione 5 – Eliminazione dei salti

8 ottobre 2025

Un programma proprio è un programma che presenta un solo ingresso e una sola uscita. Due programmi si dicono equivalenti quando, a parità di input, producono lo stesso output.

Il teorema di Böhm-Jacopini afferma che, dato un qualsiasi programma non strutturato (P), è sempre possibile costruire un programma strutturato (S(P)) che sia equivalente a (P), cioè che produca gli stessi risultati. In altre parole, questo teorema dimostra che un programma strutturato ha la stessa potenza di calcolo di un programma non strutturato.

La dimostrazione del teorema si basa sull'idea che qualunque programma può essere riscritto utilizzando solo tre costrutti fondamentali: la sequenza, la selezione e l'iterazione (in particolare il ciclo while). Tuttavia, la trasformazione completa di un programma non strutturato in uno strutturato può comportare un aumento del numero di variabili e di istruzioni, rendendo il programma più pesante dal punto di vista delle risorse.

Successivamente, il teorema di Ashcroft e Manna ha introdotto un approccio più efficiente: invece di riscrivere completamente il programma, come proposto da Böhm e Jacopini, si interviene solo sulle parti non strutturate, sostituendole con blocchi strutturati. In questo modo si ottiene un codice più leggibile e corretto, ma con meno modifiche complessive.

Nella programmazione strutturata, l'uso di istruzioni come break all'interno di condizioni o cicli non è considerato corretto, poiché interrompe il flusso logico del programma; l'unico break ammesso è quello utilizzato nello switch per terminare un caso.

La correttezza del codice viene garantita attraverso due concetti fondamentali: invariante e asserzione.

- L'invariante è una condizione che rimane sempre vera durante l'esecuzione di una parte del programma, ad esempio all'interno di un ciclo, e serve a dimostrare che il ciclo funziona come previsto.
- L'asserzione, invece, è una condizione che si verifica in un punto specifico del programma (prima o dopo un'istruzione) per controllare che i valori delle variabili siano corretti.

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

Se durante l'esecuzione di un ciclo è necessario verificare esplicitamente perché si è usciti da esso, significa che probabilmente il flusso logico non è ben strutturato.
Infine, è importante notare che, sebbene i programmi strutturati siano più chiari e affidabili, i programmi non strutturati (come quelli scritti in linguaggio assembler) risultano spesso più efficienti, poiché permettono un controllo diretto e immediato sull'hardware. Tuttavia, questa efficienza si paga in termini di leggibilità e manutenzione del codice.

Lezione 5 – Introduzione al Linguaggio C

14 ottobre 2025

Il linguaggio C è considerato un linguaggio di programmazione a basso livello, in quanto permette un controllo diretto sull'hardware e sulla memoria del computer, pur mantenendo una certa leggibilità. Questo linguaggio è progettato per "ragionare come l'esecutore", cioè per operare in modo molto vicino al funzionamento reale della macchina.

In C, l'intero programma è racchiuso all'interno di una funzione principale chiamata main. È proprio da questa funzione che inizia l'esecuzione: il sistema operativo, quando avvia il programma, cerca automaticamente la funzione main e inizia a eseguire le istruzioni che contiene, partendo dalla prima riga.

I file che contengono il codice scritto dal programmatore vengono detti file sorgenti (o sorgenti). Essi vengono elaborati dal compilatore per essere trasformati in un linguaggio comprensibile alla macchina. Tuttavia, la traduzione del C non avviene direttamente in linguaggio binario: prima il codice C viene convertito in linguaggio assembly, e solo in seguito l'assembly viene tradotto in linguaggio macchina, che può essere eseguito direttamente dalla CPU.

In C è molto comune utilizzare la direttiva #include, che serve per includere il contenuto di un altro file (come una libreria o un header) nel punto del programma in cui è scritta. Questo processo avviene prima della compilazione, durante la cosiddetta fase di preprocessing.

Una caratteristica importante del C è che non effettua controlli durante l'esecuzione del programma: tutti i controlli (come la verifica dei tipi, la corretta dichiarazione delle variabili o la sintassi) vengono fatti durante la compilazione, e non a runtime. Questo lo rende un linguaggio molto veloce, ma anche più soggetto a errori difficili da individuare.

Per quanto riguarda l'input, la funzione getchar() serve per leggere un singolo carattere inserito da tastiera. Quando l'utente digita un carattere e preme Invio, nel buffer di input vengono salvati due caratteri: quello digitato e il carattere di newline \n generato dal tasto Invio.

Infine, in C vige una convenzione logica: il valore 0 rappresenta "falso", mentre qualsiasi altro numero rappresenta "vero". Questa regola viene utilizzata in tutte le espressioni condizionali e nei cicli, rendendo il linguaggio semplice ma molto efficiente dal punto di vista computazionale.

Lezione 5 – Programmazione in Linguaggio C

14 ottobre 2025

In un programma, una variabile ha un'esistenza limitata, cioè vive solo all'interno del suo scope, che rappresenta l'ambito di visibilità della variabile. In altre parole, lo scope indica la parte di codice in cui quella variabile può essere usata. Quando lo scope termina, la variabile viene distrutta e la memoria che occupava viene liberata.

Esistono due tipi principali di scope:

- Globale, quando la variabile è dichiarata all'esterno di tutte le funzioni e può essere utilizzata in tutto il programma.
- Locale, quando la variabile è dichiarata all'interno di una funzione o di un blocco di codice, e quindi esiste solo in quella porzione specifica.

Lo scope è molto importante, perché permette di tenere sotto controllo le modifiche ai dati del programma ed evitare conflitti tra variabili con lo stesso nome.

Molti operatori in C si comportano in modo simile tra tipi numerici, ma alcuni cambiano comportamento a seconda del tipo della variabile.
Ad esempio, l'operatore + tra interi esegue una somma numerica, mentre tra stringhe (in altri linguaggi, non in C puro) può servire per concatenarle.

Quando si esegue un'operazione tra tipi diversi, il compilatore effettua una promozione dei tipi, cioè converte automaticamente il tipo più piccolo nel tipo più grande. Questo serve a prevenire errori come overflow o underflow e a garantire un risultato corretto.
Anche il tipo char, infatti, è considerato un numero molto piccolo (8 bit), quindi se partecipa a un'operazione con un tipo più grande, viene automaticamente convertito.

Il tipo di ritorno (return) di una funzione deve sempre corrispondere al tipo dichiarato nella sua intestazione, altrimenti si genera un errore o un comportamento indefinito.

Il tipo di una variabile definisce:

- che cosa può contenere (es. numero, carattere, valore logico);
- quanta memoria occupa;
- come viene codificato il valore in memoria;
- quali operazioni può subire;
- e quali valori può assumere.

Gli identificatori, cioè i nomi di variabili e funzioni, non possono mai iniziare con una cifra, ma solo con una lettera o un underscore _.

L'assegnamento (=) è un vero e proprio operatore: oltre a restituire un valore (quello assegnato), modifica anche la variabile posta a sinistra dell'uguale. Questa modifica viene chiamata side effect (effetto collaterale), perché cambia lo stato del programma.

Una funzione utile del linguaggio C è isdigit(), che si trova nella libreria <ctype.h> e serve per controllare se un carattere è una cifra numerica.
Le funzioni, inoltre, possono essere richiamate anche dentro le condizioni, ad esempio all'interno di un if.

I file header (.h) contengono le dichiarazioni delle funzioni, dette prototipi. Queste dichiarazioni servono al compilatore per sapere che una funzione esiste, come si chiama, che tipo di valore restituisce e quali parametri accetta.

In sintesi, lo scope definisce la “vita” di una variabile, mentre gli operatori e le promozioni di tipo regolano come i dati vengono manipolati in modo sicuro ed efficiente all’interno del programma.

Lezione 6 – Programmazione in Linguaggio C

15 ottobre 2025

Quando l’utente digita qualcosa, i dati inseriti vengono temporaneamente salvati in un’area di memoria chiamata buffer, fino a quando non vengono letti o rimossi da un’istruzione del programma.
L’istruzione scanf in C di solito elimina gli spazi bianchi presenti nel buffer e cerca al suo interno un pattern, cioè una sequenza di elementi (numeri, caratteri, ecc.) corrispondenti al formato richiesto. Quando trova ciò che riconosce, lo preleva dal buffer.

L’operatore ternario è un operatore condizionale che permette di scrivere in forma compatta una struttura if–else, rendendo il codice più sintetico.

Nel costrutto switch, l’uso del break non è obbligatorio dal punto di vista sintattico: il programma compila anche se non viene inserito. Tuttavia, se si omette, il flusso di esecuzione prosegue automaticamente nel caso successivo, fenomeno noto come fall-through.

Nel ciclo for, se non si inserisce una condizione (cioè la seconda istruzione tra le parentesi tonde), questa viene considerata sempre vera, e ciò può causare un ciclo infinito.
Se invece si dichiara la variabile di controllo (ad esempio int i) direttamente all’interno del for, essa sarà visibile solo all’interno del ciclo e non nel resto del programma.

L’operatore virgola (,) in C serve per combinare più espressioni in un’unica istruzione: le valuta da sinistra verso destra e restituisce il valore dell’ultima.

L’istruzione continue permette di saltare il resto del corpo del ciclo e passare subito all’iterazione successiva. È meno drastica del break, che invece interrompe completamente il ciclo. Tuttavia, anche continue è un’istruzione che si consiglia di usare con moderazione per mantenere il codice chiaro.

Una stringa in C è un array di caratteri che termina con un carattere speciale chiamato terminatore di stringa ('\0'), mentre un vettore è una struttura con dimensione nota a priori, cioè definita al momento della dichiarazione.

Lezione 8 – Funzioni in C

28 ottobre 2025

L’effetto shadowing si verifica quando due variabili locali con lo stesso nome vengono dichiarate in blocchi di codice (scope) diversi, uno all’interno dell’altro. In questo caso, la variabile più interna “oscura” quella esterna, rendendo difficile capire quale delle due venga effettivamente utilizzata. Questo comportamento può causare ambiguità e rendere il codice meno leggibile o più soggetto a errori.

I parametri formali sono i nomi dei parametri definiti nel prototipo di una funzione, cioè nella sua dichiarazione. I parametri attuali, invece, sono i valori o variabili che vengono passati alla funzione nel momento della chiamata. In altre parole, i parametri formali rappresentano le "variabili di riferimento" della funzione, mentre quelli attuali sono i dati concreti con cui essa lavora.

Le macro sono delle definizioni simboliche create tramite la direttiva #define e gestite dal preprocessore, cioè prima che il codice venga effettivamente compilato. Servono per sostituire automaticamente un nome o un'espressione con un valore o una sequenza di istruzioni, permettendo di scrivere codice più compatto e flessibile.

Lezione 8 – Call and Return

28 ottobre 2025

Lo stack è una zona di memoria associata a ciascun processo, utilizzata principalmente per salvare le variabili locali e le informazioni relative agli ambiti di visibilità (scope) delle funzioni. La struttura dello stack funziona secondo la logica LIFO (Last In, First Out): l'ultimo elemento inserito è il primo a essere rimosso. Ad esempio, il primo blocco nello stack corrisponde solitamente alla funzione main.

Tutti i dati locali, come le variabili dichiarate all'interno di una funzione e i parametri passati, vengono allocati nello stack. Tuttavia, dallo stack è possibile accedere soltanto all'ultimo blocco aggiunto, che rappresenta la funzione attualmente in esecuzione. L'insieme di oggetti caricati nello stack ad ogni chiamata di funzione prende il nome di record di attivazione.

In generale, lo stack contiene tutte le strutture dati statiche di un programma e, se utilizzato in modo eccessivo o in modo errato, può generare un stack overflow, cioè un superamento dello spazio di memoria disponibile.

Una funzione è detta polimorfica quando è in grado di operare su diversi tipi di dati, adattando il proprio comportamento in base al tipo degli argomenti.

Riguardo alle variabili, il binding indica l'associazione tra una variabile e il suo tipo. Se il tipo è fissato in modo stabile e non cambia durante l'esecuzione del programma, si parla di binding statico. Al contrario, se il tipo può cambiare durante l'esecuzione, come avviene in alcuni linguaggi dinamici, si parla di binding dinamico.

Anche il controllo dei tipi può essere statico o dinamico. Il controllo statico avviene prima della compilazione, come in C o Java, mentre il controllo dinamico avviene durante l'esecuzione del programma, come in Python. Quando vogliamo assicurarci che una variabile non generi errori di tipo o overflow a runtime, è necessario utilizzare un controllo dinamico del tipo.

In C, ogni variabile e ogni valore ha un tipo, come int, float, char, ecc., che determina sia la quantità di memoria occupata sia le operazioni che possono essere eseguite su di essa. Il linguaggio C utilizza un controllo dei tipi statico, il che significa che il tipo di ogni variabile è stabilito durante la compilazione e non può cambiare durante l'esecuzione del programma.

Il C permette alcune operazioni avanzate, come il casting (convertire un tipo in un altro) e la creazione di puntatori, cioè variabili che contengono l'indirizzo di memoria di un'altra variabile. Inoltre, il costrutto union consente di creare una struttura dati con più campi, ma a ogni momento può contenere solo un campo alla volta, perché tutti condividono la stessa area di memoria.

Per confronto, l'assembler è un linguaggio ancora meno tipizzato, dove il tipo delle variabili è quasi del tutto assente.

Ogni volta che una funzione viene chiamata, nello stack viene creato un record di attivazione, che rappresenta lo scope della funzione in quel momento e quindi lo stato dello stack relativo alla funzione attiva.

Passaggio dei parametri in C:

- Call by value: viene passata una copia del parametro alla funzione.
  - Si possono usare i parametri formali come variabili locali.
  - Il passaggio per valore non ha effetti collaterali, cioè la variabile originale non viene modificata al di fuori dello scope della funzione.
- Call by reference: viene passato l'indirizzo di memoria della variabile.
  - Il passaggio per riferimento genera un effetto collaterale, perché la variabile viene modificata anche al di fuori del suo scope.

In C, i parametri formali possono essere trattati come variabili locali della funzione. Ad esempio, quando si usa scanf, si passa sempre il riferimento della variabile (tramite l'operatore &), perché la funzione deve modificare direttamente il valore della variabile.

Per lavorare con gli indirizzi di memoria, C mette a disposizione due operatori:

- & = restituisce l'indirizzo di memoria di una variabile;
- * = consente di accedere al valore memorizzato all'indirizzo indicato.

Le funzioni in C possono restituire un solo valore. Se vogliamo restituire più valori, dobbiamo farlo attraverso i parametri passati per indirizzo.

Oltre a call by value e call by reference, esistono altri metodi di passaggio dei parametri in alcuni linguaggi:

- Call by result: il parametro viene usato solo per restituire un valore. Le modifiche all'interno della funzione vengono copiate all'esterno solo quando la funzione termina.
- Call by value-result: combina call by value e call by result. All'inizio viene copiata una copia del valore originale all'interno della funzione, e alla fine le modifiche vengono copiate all'esterno.
- Call by name: si passa il nome del parametro, e il suo valore viene ricalcolato ogni volta che viene usato nella funzione.

La differenza tra passaggio per indirizzo e call by result è che nel primo la variabile viene modificata immediatamente, sia all'interno che all'esterno della funzione, mentre nel secondo le modifiche avvengono solo all'interno e vengono copiate all'esterno alla fine.

In C, però, è presente solo il passaggio per valore, e ogni effetto di modifica diretta deve essere realizzato usando puntatori.

Lezione 9 - Puntatori e Strutture

29 ottobre 2025

Un puntatore è un tipo di variabile speciale il cui valore rappresenta l'indirizzo di memoria di un'altra variabile. In memoria, un puntatore viene rappresentato come un numero intero e, di

conseguenza, molte delle operazioni valide sui numeri possono essere applicate anche ai puntatori.

In genere, un puntatore occupa un numero fisso di celle di memoria, spesso 4 byte nei sistemi a 32 bit. L'operatore * permette di accedere al valore memorizzato all'indirizzo a cui il puntatore fa riferimento, consentendo anche di modificarlo direttamente.

L'aritmetica dei puntatori consiste nel fare operazioni sugli indirizzi contenuti nei puntatori, come sommare o sottrarre un offset per spostarsi tra celle di memoria adiacenti. Un puntatore può essere anche inizializzato a un valore specifico, come ad esempio l'indirizzo di una variabile esistente.

Il costrutto union è simile a una struct, ma con una differenza fondamentale: tutti i suoi campi condividono lo stesso spazio di memoria, quindi a ogni istante può contenere un solo valore. Il compilatore alloca uno spazio pari a quello del campo più grande e gli altri campi si sovrappongono a esso.

Le union sono utilizzate principalmente per risparmiare memoria, quando vogliamo memorizzare diversi tipi di dato nella stessa locazione di memoria. Dal punto di vista pratico, si utilizzano come le struct, accedendo ai campi con lo stesso nome, ma bisogna ricordare che solo un campo alla volta può contenere un valore valido.

Esempio pratico: in un gioco, un personaggio può scegliere di sparare o camminare, ma non può fare entrambe le azioni contemporaneamente. In questo caso, si può usare una union per rappresentare questa scelta, memorizzando solo l'azione attiva in quel momento.

Lezione 10 – Memoria strutture dinamiche e puntatori

5 novembre 2025

Quando si alloca un'area di memoria dinamicamente, non si utilizza lo stack, ma lo heap, che è una zona di memoria separata, gestita a runtime. Questo permette di decidere quanto spazio riservare durante l'esecuzione del programma.

Il puntatore void * è un puntatore generico, cioè può puntare a qualunque tipo di dato. Per gestire la memoria dinamica, il linguaggio C mette a disposizione alcune funzioni fondamentali:

- malloc(): alloca un blocco di memoria della dimensione specificata (in byte), ma non inizializza il contenuto.
- calloc(): alloca un blocco di memoria e azzera tutto il contenuto.
- realloc(): ridimensiona un'area di memoria precedentemente allocata; se la nuova dimensione è minore, l'area viene ridotta; se è maggiore, l'area viene estesa, ma la nuova parte non viene azzerata automaticamente.
- free(): dealloca la memoria precedentemente allocata.

Quando si usa una funzione di allocazione, spesso è necessario fare un casting per indicare il tipo di dato che vogliamo gestire.

Ogni volta che si alloca memoria dinamica, è importante deallocarla dopo l'uso per evitare perdite di memoria (memory leak). Il tipo size_t viene utilizzato per rappresentare la dimensione in byte di un oggetto in memoria.

Alcune regole pratiche:

- Il puntatore passato a free() deve indicare la prima cella dell'area da liberare.
- Dopo aver chiamato free(), è buona norma inizializzare il puntatore a NULL per evitare errori di accesso a memoria non valida.
- Con free() si dealloca la memoria puntata, non il puntatore stesso.

Un puntatore a puntatore è un puntatore che punta a un altro puntatore, il quale a sua volta punta a una variabile.

In C non è possibile fare l'overloading delle funzioni, cioè non si possono avere più funzioni con lo stesso nome e parametri diversi, come invece accade in C++.

Lezione 14 – I file

5 novembre 2025

I file sono flussi di bit (sequenze di dati memorizzati su disco).

I file si dividono in due categorie principali:

- File binari = sono flussi di bit, contengono dati in formato binario (non leggibili direttamente).
- File di testo = sono flussi di caratteri, contengono dati leggibili (come lettere, numeri e simboli).

Quando lavoro con i file in C, utilizzo un puntatore a struttura di tipo FILE che punta alle informazioni del file aperto.
(Esempio: FILE *fp;)

Esistono tre stream standard gestiti automaticamente dal sistema:

- Standard input (stdin) = è il flusso di ingresso, cioè una sequenza di caratteri proveniente dalla tastiera.
- Standard output (stdout) = è il flusso di uscita, cioè l'output inviato al monitor (quando scrivo con printf(), i caratteri appaiono sullo schermo).
- Standard error (stderr) = è il flusso di errore, usato dal programma per stampare i messaggi di errore sullo schermo, separatamente dall'output normale.

Gli stream standard (stdin, stdout e stderr) vengono gestiti direttamente dal sistema operativo. Quando si vuole aprire un file, è necessario sapere se si tratta di un file binario o di un file di testo, perché il comando di apertura cambia a seconda del tipo. Inoltre, bisogna conoscere il nome del file e la modalità con cui lo si vuole aprire.

Un file può essere aperto in sola lettura con la modalità "r", in scrittura con "w" oppure in accodamento con "a". Nel caso di file binari, dopo la lettera della modalità si aggiunge la lettera "b", ad esempio "rb" per leggere un file binario, "wb" per scrivere e "ab" per accodare.

Per aprire un file si utilizza la funzione fopen, che restituisce un puntatore alla struttura di tipo FILE. Se la funzione non riesce ad aprire il file, restituisce NULL. È importante chiudere sempre i file con la funzione fclose, in modo da liberare la struttura associata in memoria e aggiornare correttamente il file system. Se un file non viene chiuso correttamente, potrebbero verificarsi problemi di scrittura o perdita di dati.

La scrittura e la lettura dei file avvengono tramite le funzioni fprintf e fscanf. La prima serve per scrivere dati all'interno del file, mentre la seconda permette di leggerli. Quando il programma raggiunge la fine di un file, viene restituito il valore EOF (End Of File), che indica che non ci sono più dati da leggere. La funzione feof serve per controllare se la lettura è terminata correttamente o se si è arrivati effettivamente alla fine del file.

Durante le operazioni di lettura e scrittura, il sistema utilizza una zona di memoria chiamata buffer, dove i dati vengono temporaneamente memorizzati prima di essere scritti sul file. Per svuotare il buffer e forzare la scrittura immediata dei dati sul disco, si usa la funzione fflush. In alcune situazioni, tuttavia, il fflush viene eseguito automaticamente, ad esempio quando si scrive un carattere di nuova linea (\n) oppure quando il file viene chiuso con la funzione fclose.

Lezione 13 – Ricorsione

5 novembre 2025

Per ottenere un effetto di ripetizione in informatica si possono utilizzare i cicli oppure la ricorsione. Entrambi permettono di eseguire più volte le stesse istruzioni, ma con valori che cambiano a ogni iterazione o chiamata.

La ricorsione consiste in una funzione che chiama sé stessa per risolvere un problema più grande suddividendolo in sottoproblemi più semplici. È l'analogo, in programmazione, dell'induzione matematica, poiché si basa su due concetti fondamentali:

- un caso base, che interrompe la ricorsione;
- e un passo ricorsivo, in cui la funzione si richiama con un parametro ridotto per avvicinarsi al caso base.

Nelle funzioni ricorsive, il processo di winding (avvolgimento) si riferisce alla fase in cui vengono effettuate le chiamate ricorsive: ogni nuova chiamata genera un nuovo record di attivazione che viene aggiunto allo stack. In questa fase, la funzione continua a richiamare sé stessa fino a raggiungere il caso base.

La fase di unwinding (svolgimento), invece, avviene quando le chiamate ricorsive iniziano a terminare: i record di attivazione vengono progressivamente rimossi dallo stack, e la funzione restituisce i risultati parziali fino a completare l'esecuzione.

L'iterazione si basa su strutture statiche, come variabili di controllo e cicli (for, while), mentre la ricorsione è più dinamica, perché sfrutta lo stack per creare nuovi record di attivazione a ogni chiamata.

Se una funzione ricorsiva non si avvicina mai al caso base, il programma entra in un ciclo infinito di chiamate e non terminerà mai, provocando un overflow dello stack.

In generale, gli array si gestiscono più facilmente con i cicli iterativi, mentre le liste (soprattutto le liste concatenate) si prestano meglio alla ricorsione, grazie alla loro struttura dinamica e ripetitiva.

Lezione 13 – command line

11 novembre 2025

In C, una stringa è una sequenza di caratteri immutabili, terminata dal carattere speciale '\0' (null). Ciò significa che il contenuto di una stringa non può essere modificato direttamente come se fosse una semplice variabile, ma occorre gestirla tramite funzioni o puntatori.

La funzione main può essere definita con dei parametri formali, che vengono poi sostituiti dai parametri attuali quando il programma viene eseguito dalla linea di comando.

I parametri formali sono:

- int argc: rappresenta il numero di argomenti passati al programma (incluso il nome del programma stesso);
- char *argv[ ]: è un array di puntatori a caratteri, cioè un vettore di stringhe che contiene gli argomenti passati al programma.

Quindi, int argc, char *argv[ ] servono per gestire i valori forniti all'avvio del programma

Il risultato restituito da un programma in C è sempre un numero intero, che rappresenta il codice di uscita del programma. Esistono due costanti predefinite:

- EXIT_SUCCESS = indica che il programma è terminato correttamente;
- EXIT_FAILURE = indica che si è verificato un errore durante l'esecuzione.

Quando si passa un array a una funzione, in realtà viene passato solo l'indirizzo del primo elemento dell'array (un puntatore), e non la sua dimensione. Per questo motivo, se si vuole conoscere la lunghezza dell'array all'interno della funzione, bisogna passarla come parametro aggiuntivo.

Lezione 15 – Puntatori a funzione

11 novembre 2025

Le macro sono direttive per il preprocessore (non per il processore) del linguaggio C. Vengono definite tramite la parola chiave #define e servono per sostituire parti di codice con un testo specificato prima della compilazione.

La differenza tra macro e funzioni è che:

- le macro vengono elaborate dal preprocessore, non hanno veri e propri parametri (anche se si possono definire macro con argomenti testuali), sono più veloci perché non generano chiamate a funzione ma semplici sostituzioni di testo;
- le funzioni, invece, vengono gestite dal compilatore, hanno parametri con tipi definiti, permettono il controllo dei tipi, e generano un vero e proprio salto di esecuzione.

Le celle di memoria possono contenere due tipi di informazioni:

- dati (numeri, caratteri, variabili, ecc.);
- istruzioni (cioè le operazioni che il programma deve eseguire).

Per questo motivo, un puntatore può puntare sia a un dato, sia a un'istruzione.

Un puntatore a funzione è un particolare tipo di puntatore che punta all'indirizzo di una funzione. Con un puntatore a funzione è possibile:

- assegnarlo a una variabile;
- inserirlo in un vettore o in una struttura;
- passarlo come parametro a un'altra funzione;
- restituirlo come valore di ritorno di una funzione;
- richiamare la funzione a cui punta, eseguendola indirettamente.

Infine, è importante distinguere tra queste due dichiarazioni:

- int *f(int); = dichiara una funzione che restituisce un puntatore a intero.
- int (*f)(int); = dichiara un puntatore a funzione che punta a una funzione che accetta un intero come parametro e restituisce un intero.

I puntatori a funzione si utilizzano quando si vuole parametrizzare il comportamento di una funzione passando come argomento un'altra funzione da applicare. Sono molto utili anche per definire quali funzioni devono essere usate per un certo tipo di dato rappresentato da una struct (ad esempio per costruire manualmente una tabella di metodi o una «vtable»). In pratica permettono di cambiare l'algoritmo o l'azione eseguita dal codice senza modificarne la struttura: basta passare o assegnare un diverso puntatore a funzione.

Dal punto di vista della sicurezza, i puntatori a funzione vanno trattati con attenzione. Se un attaccante riesce a sovrascrivere o a manipolare il contenuto di memoria che contiene un puntatore a funzione, può far sì che il programma esegua codice non previsto. Questo è il concetto generale dietro tecniche come l'esecuzione di shellcode o l'hijacking del flusso di controllo.

In particolare, una forma di attacco consiste nel dirottare il flusso di controllo del programma — ad esempio modificando il valore di un puntatore a funzione o l'indirizzo di ritorno sullo stack — in modo che il processore esegua istruzioni arbitrarie. Anche questa è una descrizione ad alto livello.

Lezione 16 – Qualificatore const

12 novembre 2025

const è un qualificatore che non alloca memoria, ma modifica il comportamento di una variabile indicandone la natura costante.
Serve ad avvisare il compilatore (e anche il programmatore) che il contenuto della variabile associata non può essere modificato dopo l'inizializzazione.

La principale differenza tra #define e const è che:

- #define è una direttiva del preprocessore e sostituisce il valore letteralmente prima della compilazione;
- const, invece, definisce una vera e propria variabile costante, gestita dal compilatore, che possiede un tipo e un indirizzo di memoria.

Inoltre, quando si usa un puntatore con const, si può specificare se:

- il valore puntato non può essere modificato, oppure
- se l'indirizzo del puntatore stesso non deve cambiare.

Lezione 16 – Qualificatore static

12 novembre 2025

Una variabile dichiarata con il qualificatore static viene inizializzata una sola volta e mantiene il proprio valore anche dopo l'uscita dalla funzione in cui è definita.
In altre parole, il suo valore persiste tra una chiamata e l'altra della funzione, pur rimanendo visibile solo all'interno di quello scope.

Questo significa che, se all'interno di una funzione assegno a una variabile static il valore 5, alla successiva chiamata della funzione quella variabile non verrà reinizializzata, ma conserverà ancora il valore 5.

Le variabili static hanno durata di vita globale, ma visibilità limitata:

- se dichiarate dentro una funzione, mantengono il valore tra le chiamate ma sono accessibili solo da quella funzione;
- se dichiarate fuori da una funzione (a livello di file), sono visibili solo all'interno di quel file sorgente e non possono essere utilizzate da altri file del programma.

Lo stesso principio vale anche per le funzioni dichiarate static: esse possono essere richiamate solo nel file in cui sono definite, evitando conflitti di nomi con funzioni omonime in altri file.

La differenza principale tra variabili globali e variabili statiche è quindi la seguente:

- le variabili globali sono accessibili da qualsiasi file del programma (tramite extern);
- le variabili statiche hanno visibilità limitata al file in cui sono dichiarate, anche se il loro valore persiste per tutta l'esecuzione del programma.