Orale

Algoritmo:Un algoritmo è una sequenza finita di istruzioni che, quando eseguite, svolgono un ben determinato compito. deve essere chiaro, avere un numero di istruzioni finito ed elementari.

Linguaggio di programmazione: una serie di strumenti che ci permettono di convertire l'algoritmo in linguaggio macchina interpretabile dalla macchina.

L'interprete è un programma che traduce, istruzione per istruzione, il programma sorgente. Ogni istruzione è sottomessa alla CPU appena è stata tradotta. Anche se una istruzione è contenuta 10 volte in un programma verrà tradotta ogni volta che si presenterà al traduttore.
Il compilatore, invece, traduce tutto il programma in una sola volta e poi lo passa alla CPU. Se una stessa istruzione compare 10 volte, viene tradotta solo la prima volta e poi memorizzata in maniera tale che possa essere riusata.

Una variabile è una entità che può assumere un valore qualunque all'interno di un insieme di valori, avente un tipo semplice o strutturato.

I **sottoprogrammi** o **funzioni** sono usati per semplificare la soluzione di problemi complessi attraverso il noto principio del divide et impera:
dividi il problema in sottoproblemi
risolvi separatamente i sottoproblemi
ricombina i risultati.
ogni funzione ha bisogno di **parametri** in input per eseguire il compito richiesto, e si definiscono **formali** al momento della creazione della funzione, e **attuali** al momento della chiamata. i parametri attuali sostituiranno al momento della chiamata di funzione, i parametri formali. I parametri possono essere passati o per **valore**, cioè corrispondono solo ad input e non possono subire modifiche dato che viene passato una copia di tale variabile; o per **riferimento** come avviene con i vettori cioè passando l'indirizzo a cui è salvata tale variabile e quindi modificabile.

Gli **operatori relazionali** in C sono: <,<=,>,>= e producono valore 0 o 1 (sono associativi a sinistra)

operatori logici: ! && ||

operatore condizionale: k = i > j ? i : j;

istruzioni di Controllo
Condizionali
If, if/else, switch
Iterative
while, do/while, for
Salti
break, continue, goto, return

break: Causa un'uscita immediata da una struttura while, for, do/while o switch e L'esecuzione del programma continua con la prima istruzione successiva dopo la struttura

continue: Salta le istruzioni che restano in un corpo di un while, for o do/while
Esegue la successiva iterazione del loop

un **ARRAY** è una collezione di variabili omogenee accessibili attraverso un indice indicante la posizione dello stesso. l'ARRAY bidimensionale ha bisogno di essere specificato almeno il numero

di colonne, necessario affinchè ogni elemento possa essere ritrovato, dato che ogni array è memorizzato per riga.

puntatore: è una variabile che contiene l'indirizzo di un oggetto o variabile. il tipo di dato puntato viene detto tipo referenziato. I puntatori sono usati per la creazione di strutture dinamiche o per il passaggio per riferimento. *****: operatore di deferenziazione per accedere al contenuto dell'indirizzo puntato. (int *p=&v[1]); Con l'aritmetica dei puntatori è possibile sommare o sottrarre un intero all'indirizzo puntato in modo che il puntatore andrà a puntare ad un elemento successivo o precedente, quindi p=p+2; equivale a far puntare a p la casella v[2]. Se p punta ad un vettore "v", posso accedere alle caselle di v con *(p+2)(cioè indirizzo base del vettore+2) che equivale a p[2];

una stringa letterale *p="casa" è un letterale costante perchè p punta all'indirizzo del primo byte di a quindi non è possibile modificare la stringa in nessun caso. una stringa variabile invece è un array contenente alla fine il "**tappo**". È lecito dichiarare un vettore e poi inizializzarlo con una stringa (v[5]="ciao"). per gestire più stringhe si può usare o una **matrice** anche se poi in caso di lunghezza non omogenea delle parole, ci sarà uno spreco di spazio,oppure un array di puntatori a stringhe (char *v[]), in cui ogni puntatore punta ad una stringa di lunghezza diversa.

Allocazione dinamica: è possibile allocare dinamicamente la memoria necessaria per memorizzare una stringa, usando la funzione **malloc** o **calloc**, che restituiscono un puntatore alla zona di memoria cercata. tali funzioni sono utili anche quando si vuole allocare lo spazio necessario a contenere un vettore di dimensione dinamica. dichiariamo un puntatore int *v. Poi v=malloc(sizeof(int)*dim); Lo spazio allocato da malloc va svuotato alla fine del programma con **free**(puntatore); Usata free, la zona puntata viene svuotata e si crea un **puntatore pendente** che è vietato usare.

Una struttura serve a memorizzare un insieme di informazioni collegate fra loro Definendo un nuovo “tipo”. e possibile accedere al membro di un record mediante il **punto**. Per evitare di passare il record ad una funzione e quindi copiare tutti i campi, è utile passare un puntatore al record.

un file è una sequenza di byte. I bytes in un file di **testo** rappresentano caratteri e quindi possono essere stampati, letti e modificati con un **“text editor”**. In un file **binario**, i byte possono assumere qualsiasi valore, gruppi di byte potrebbero rappresentare altri tipi di dati ecc.. I file sono aperti con **fopen** in vari modi e restituisce un puntatore **nullo** se non riesce ad accedere al file; Si chiudono con **fclose**.

file testo:
prendere un carattere **getc(p)**, inserirlo **putc(ch,p);**
fprintf e **fscanf** salvano e prendono caratteri da steam.
La chiamata **feof(fp)** restituisce un valore non-zero se il segnalatore **EOF** è stato attivato dalla precedente operazione sullo stream fp.
per leggere un'intera riga compresi gli spazi si usa fgets, mentre per inserirla viene usata **fputs(char buffer[ ], FILE *outfile)** e mette il fine riga. (fputs corrisponde a fprintf con la specifica "%s")

file binari:
per scrivere su file **fwrite(v[i],sizeof(int),1,fp);**
per leggere da file **fread(v[i],sizeof(int),1,fp);**