STRUTTURE:

In molti casi, un oggetto del mondo esterno è rappresentato da più dati all’interno del calcolatore.

Un persona, ad esempio, potrebbe essere formata da nome, data di nascita e stipendio.

Riferire tutte queste variabili a una sola persona è tuttavia molto lungo e noioso e se si devono apportare delle modifiche anche laborioso.

Sarebbe meglio poter creare un nuovo TIPO di dato: la persona

Una struttura è una collezione finita di variabili (non necessariamente dello stesso tipo) ognuna identificata dal proprio nome

COMANDO STRUCT:

struct [<etichetta>]

{

{<definizione di variabile>}

} <nomeStruttura>

struct persona

{ char nome[20];

int eta;

float stipendio;

} pers; -> definisce una variabile pers strutturata nel modo illustrato

per richiamare una sola variabile della struttura, ad esempio il nome, dovrò scrivere:

pers.nome

COMANDO TYPEDEF

typedef <TipoEsistente> <NuovoTipo>

typedef struct

{ char nome[20];

int eta;

float stipendio;

} persona;

Con questo comando quindi prendo una struttura e la faccio diventare un tipo al pari di int, float, ecc

Quindi poi potremo dichiarare

persona Alberto, Lisa, Silvia;

Come dichiareremmo

int a, b, c;

UN TIPO DI DATO È:

1. Un dominio di valori
2. Un insieme di funzioni (operazioni svolgibili come somma, prodotto ecc)
3. Un insieme di predicati (ad esempio maggiore, minore, ecc)

In C l’utenti può definire nuovi tipi strutturati (come quelli sopra) ma anche ridefinire tipi esistenti e definire nuovi tipi enumerativi

Tipi ridefiniti: typedef TipoEsistente NuovoTipo;

typedef int MioIntero;

A questo punto invece di scrivere int a, b,c; scriverò:

MioIntero a, b, c;

Tipi enumerativi: typedef enum {a1, a2, ... , aN} EnumType;

typedef enum {L,M,M,G,V,S,D} Giorni;

Un tipo enumerativo viene specificato tramite l’elenco dei valori che può assumere

L, M, G ecc. sono detti “identificativi di valore” e possono comparire una sola VOLTA nella definizione di UN SOLO TIPO, altrimenti si crea ambiguità

Un tipo enumerativo è totalmente ordinato e vale l’ordine di definizione, quindi in questo caso L<M è vero.

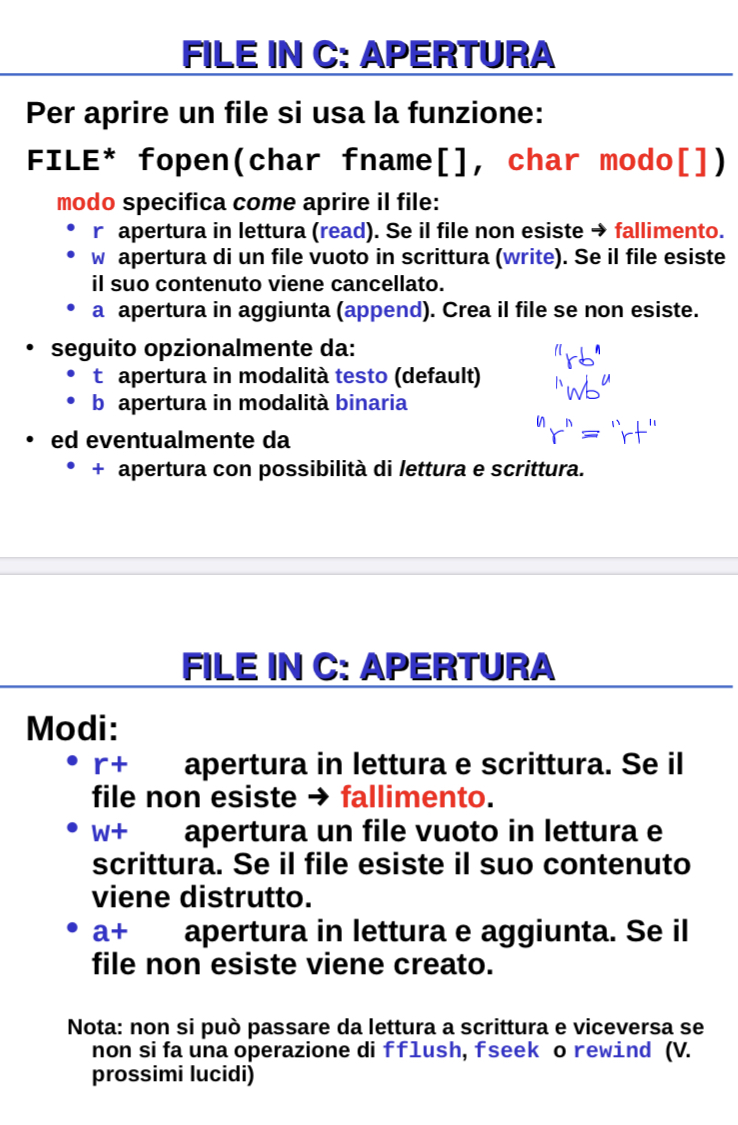
Per il C un tipo enumerativo è indistinguibile da un intero potrei fare:

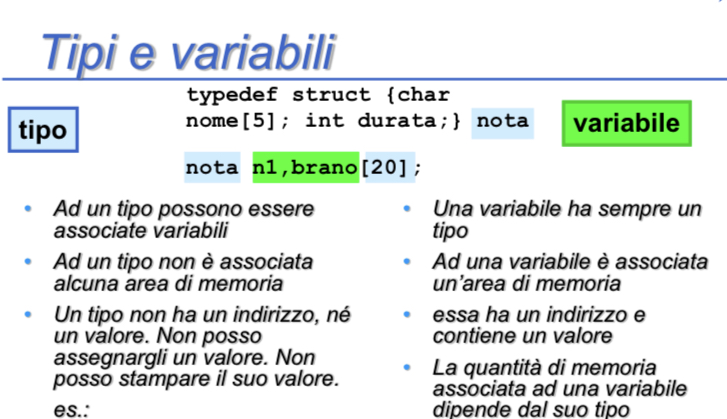
Giorni g;

g=5; /\*questo significa che g=sabato\*/

MA questa è una pratica da EVITARE

Volendo potrei specificare i valori naturali cui associare gli identificatori ad esempio:

typedef enum {L=1,M,M,G,V,S,D} Giorni; ordinati da 1 a 7

typedef enum {L=3,M=9,M,G=8,V,S,D} Giorni;

UNION:

<tipoUnion>::= union {[[ <tipo> <identificatore> ; ]] + }

Una Union è una variabile che può contenere, in momenti diversi, dati di tipo diverso, che occupano un’area di memoria condivisa sufficientemente grande per contenere il più grande dei dati possibili.

Esempio:

typedef union {

int a; float b;

} numero;

numero n;

n.a=3;

n.b=2,6;

FILE DI TESTO:

Gestione dei file: per poter mantenere disponibili i dati attraverso le diverse esecuzioni dei programmi, questi devono essere archiviati su MEMORIA DI MASSA (cd, nastri, ecc)

I file possono poi esse manipolari attraverso i programmi

Un file è un astrazione fornita dal sistema operativo per memorizzare informazioni sulla memoria di massa

Concettualmente, un file è una sequenza di record (registrazioni) uniformi (=dello stesso tipo)

È un astrazione di memorizzazione potenzialmente illimitata, ma non infinita, ad accesso SEQUENZIALE

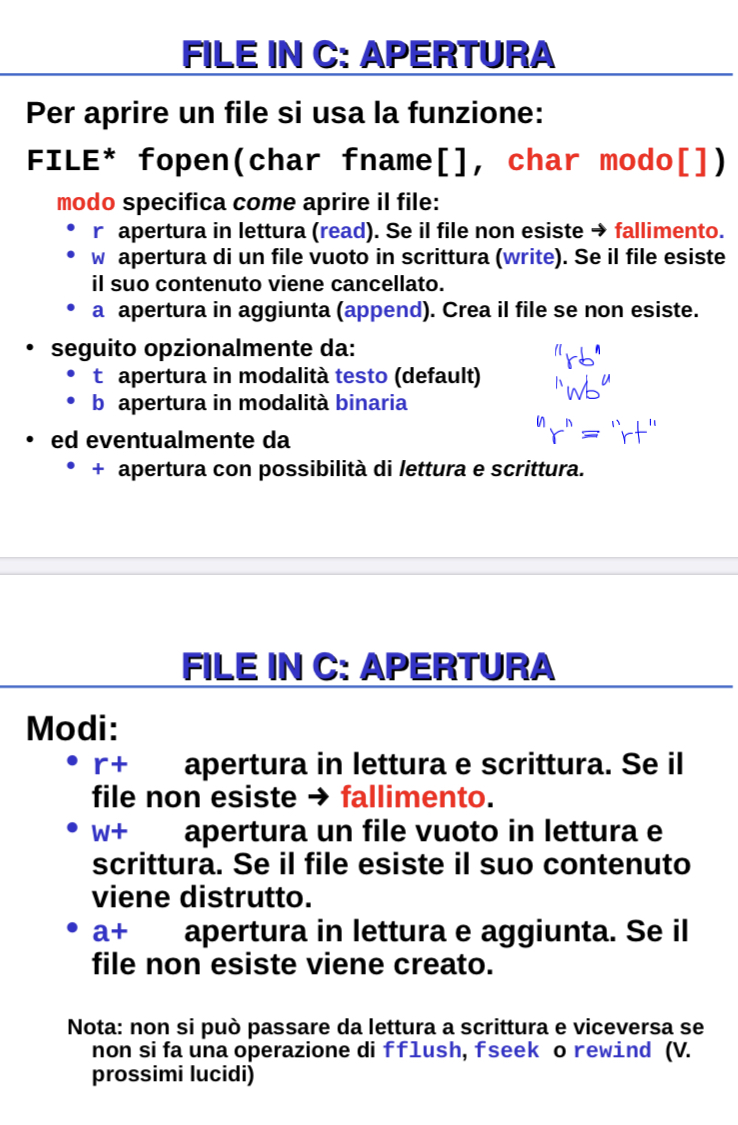
È denotato univocamente dal suo nome assoluto che comprende nome relativo e percorso.

Il tipo FILE è definito nello header stdio.h.

Per aprire un file si Sial la funzione:

FILE \* fopen (char fname[], char modo[])

Questa funzione apre il file di nome fname nel modo specificato e restituisce un puntatore a file (che punta ad una nuova struttura file appositamente creata).

Una volta dato il comando di speratura del file bisogna assicurarsi che il programma abbia spero correttamente il file. Devo quindi creare un if in cui se il file è == NULL il programma si chiuda automaticamente, ad esempio con il comando di chiusura exit(n) dove n è il numero identificativo non nullo di chiusura che scelgo per quell’errore. Il comando exit è incluso nella libreria stdlib.

Per chiudere il file si usa la funzione fclose(FILE\*)

In chiusura tutti i buffer vengono copiati e svuotati, quindi usare fclose è imperativo in quanto altrimenti rischio che le ultime modifiche al file non vengano salvate.

Un file di testo è un file che contiene sequenze di carattere ed ha caratteristiche proprie come:

1. un concetto di riga e fine riga (‘\n’)
2. Solo alcuni caratteri sono stampabili a video
3. La sequenza di caratteri è chiusa dal carattere speciale EOF

In realtà anche per leggere da tastiera e scrivere a video posso utilizzare comandi per file, esistono infatti tre file già aperti:

1. stdin: aperto in lettura e agganciato alla tastiera
2. stdout stderr: aperti in scrittura agganciati al video

La funzione feof(FILE \* fp) permette di vedere se un programma è stato o meno chiuso

Lettura di stringhe: char fgets (char \*s, int n, FILE \*fp);

Questo comando:

1. Trasferisce nella stringa s i caratteri letti dal file puntato da fp, fino a quando ha letto n-1 caratteri, oppure ha incontrato un newline o la fine del file
2. Il carattere newline, se letto, è mantenuto dalla stringa a
3. Restituisce NULL in caso di errore o fine del file

Scrittura di stringhe: int fputs (char \*s, FILE \*fp);

Questo comando:

1. trasferisce la stringa s, terminata da ‘\n’, nel file puntato da fp. Non copia il carattere ‘\n’ ne aggiunge un newline finale.
2. Restituisce un numero non negativo in caso di terminazione corretta, altrimenti EOF.

VETTORI MULTIDIMENSIONALI:

Si possono definire anche array di array, basta utilizzare typedef.

Mettiamo caso di voler creare una matrice:

Typedef int colonna [4]

Typedef colonna matrice [3]

Dichiariamo poi:

matrice M

e abbiamo la nostra matrice. Per trovare un della della matrice in particolare:

M [c][r]

OPPURE è possibile definire variabili di tipo array con più di una dimensione. Gli array con due dimensioni vengono detti matrici.

<tipo> <identificatore> [dim1] [dim2] [...] [dimN]

float M[20][30]

Per accedere all’elemento si utilizza la notazione M [r][c]

Le matrici vengono memorizzate per righe, quindi per calcolare l’indirizzo della cella a[i][j] dalla posizione iniziale devo: fare la i per la lunghezza delle righe (o numero delle colonne) più j

Se voglio chiamare una matrice all’interno di una funzione nella dichiarazione devo già dichiarare il tipo di matrice (colonne e righe) mentre nell’innovazione metto solo il nome della matrice. In pratica se io dichiaro la funzione per una matrice 2x3 e poi voglio utilizzarne una 3x4 quella funzione non va più bene.

Tuttavia basta dichiarare il numero di colonne, mentre quello di righe è opinabile, quindi io potrei scrivere una funzione

int f(int a[][3])

All’interno della quale posso usare tutte le matrici con 3 colonne, indipendentemente dalle loro righe.

Questo perché il calcolo per trovare la posizione di un elemento non comprende il numero di righe.

Progetti multifile:

Per ottenere un file oggetto da un file come main.c devo compila con l’opzione c:

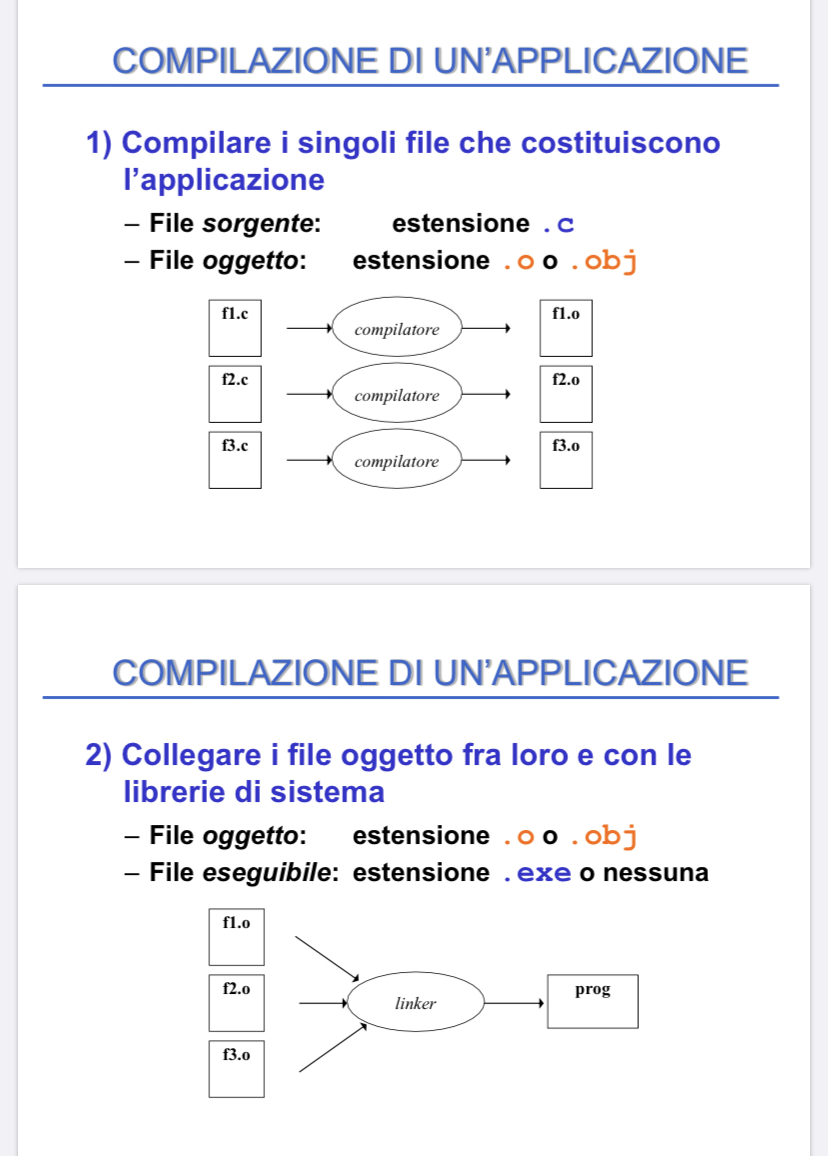
gcc -c main.c ed otterrò così il file NON ESEGUIBILE main.o

Per ottenere un file eseguibile, sia il file con il main che il file con la funzione:

1. compilo entrambi come file oggetto, ottenendo così main.o è f.o
2. compilo come gcc -o eseg main.o f.o
3. ./eseg

Con questo modo tuttavia, in ogni programma devo mettere la dichiarazione della funzione .

Se infatti nel file f ho la dichiarazione della funzione, nel file main, prima del main(), devo dichiarare la funzione, altrimenti il calcolatore darà errore.

Per evitare questo posso creare un file header con tutte le mie funzione da chiamare ad esempio files.h dove dichiaro tutte le mie funzione e che poi nel programma dovrà essere chiamato semplicemente con #include<files.h> senza dover includere ogni volta la dichiarazione.

ARGOMENTI LINEA DI COMANDO:

La funzione main nella sua dichiarazione comprende argc e argv come argomenti.

Argc è un parametro di tipo intero che rappresenta il numero di argomenti nelle linee di comando con cui si invoca la esecuzione. Questo comprende anche il nome stesso del programma quindi argc sarà sempre maggiore o uguale a 1.

Argv invece è un array di stringhe, ciascuna delle quali contiene un argomento diverso. Argv[0] contiene il nome del programma.

Argc vale N+1, argv arriva al massimo a argv[N], quindi argv[argc]=NULL

Se voglio far stampare al programma ciò che scrivo nella linea di comando mi basterà fare:

printf(“%s”,argv[i]);

Ricorsione: non c’hi capito na mazza. Più avanti? Boh

MAKE:

I sistemi di build (o sistemi di costruzione) sono programmi che automatizzano la compilazione di un progetto software, secondo le indicazioni di un file progetto.

Vantaggi:

1. ricompilano solo i file modificati
2. Non dimenticano modifiche

Make è un sistema di build per C e C++

Il file di progetto si chiama Makefile si trova nella stessa cartella dei file sorgente.

Il comando make esegue la costruzione del progetto secondo il contenuto di Makefile.

Formato Makefile:

\_\_target\_\_:\_\_dipendenza1\_\_...\_\_dipendenzaN\_\_

\_\_comandoCompilazioneTarget\_\_

1. La prima riga specifica da che file dipende il target
2. La seconda quale è il comando per ottenere il target (solitamente di compilazione)

Esempio:

main: main.o file1.o file2.o file3.o

gcc -o main main.o file1.o file2.o file3.o -lc

main.o: main.c file1.h file2.h

gcc -c main.c

file1.o: ...

file2.o: ...

file3.o: ..

ALLOCAZIONE DINAMICA:

Permette di creare delle variabili dinamiche, allocate nella memoria heap

In C le variabili possono essere divise tra:

1. Variabili globali (allocate in Data Segment hanno un tempo di vita pari al tempo di esecuzione del programma)
2. Variabili automatiche (allocate nello Stack hanno tempo di vita variabili e sono i parametri formali delle funzioni)
3. Variabili dinamiche (Heap)

Le variabili dinamiche permettono di dimensionare un array dopo aver scoperto quanto grande deve essere.

Per allocare una variabile dinamica bisogna utilizzare la funzione:

void malloc (size\_t dim);

Che chiede al sistema di allocare un’area di dim byte

La funzione restituisce un indirizzo di memoria, bisognerà quindi assegnarlo ad un puntatore.

Ad esempio:

int \*p;

p=(\*int)malloc(7\*sizeof(int));

In questo modo ottengo un vettore da 7 celle

Per deallocare la memoria devo usare la funzione:

void free (void\*p);

Malloc e free sono dichiarate in stdlib.

LISTE COLLEGATE:

È possibile realizzare strutture dati in grado di allocate memoria solo quando necessario, per ogni singolo dato da memorizzare.

L’implementazione dinamica utilizza l’allocazione della memoria.

Vantaggi:

1. l’occupazione di memoria di una lista collegata è in ogni momento proporzionale alla sua dimensione logica
2. Nelle liste sequenziali, l’inserimento e l’eliminazione di un dato, richiedono di spostare (a destra o sinistra) tutti gli altri.

Svantaggio principale: impossibilità di accesso diretto a un elemento.

Nella rappresentazione collegata, ogni elemento di tipo T è rappresentato tramite un nodo, ossia una struttura con due campi:

1. dato di tipo T, che contiene l’elemento
2. next di tipo puntatore a nodo, che contiene il collegamento all’elemento successivo alla lista

Il tipo che rappresenta il nodo è definito ricorsivamente:

typedef struct nodo

{

int dato;

struct nodo \* next;

} Nodo;

typedef Nodo \* Lista;

Il tag nodo è necessario per permettere la compilazione, in quanto altrimenti il compilatore non saprebbe cosa sia il campo next.

Poiché l’unico accesso ad una lista collegata è il suo primo elemento, una lista è definita dal puntatore ad esso.

Terminologia:

1. Gli elementi della lista sono rappresentati in nodi
2. Nel contesto dell’implementazione collegata la lista non è la sequenza dei suoi elementi, ma un puntatore ad un nodo
3. Una lista con valore NULL si dice vuota
4. La testa della lista è il campo dato, la coda il campo next

ForEach sulle liste collegate:

Non si può più usare il ciclo for come in un semplice array, ma bisognerà utilizzare un ciclo while con un puntatore che:

1. alla prima iterazione è uguale alla lista
2. a ogni iterazione successiva viene sostituito da next
3. quando vale NULL esce dal ciclo

void stampa (Lista l)

{

while (l) \\ perché quando si arriva alla fine della lista l=NULL e il ciclo si chiude da solo

{

printf(“%d”,l->dato);

l=l->next;

}

}

SICUREZZA:

Una risorsa è un entità di valora da proteggere, la vulnerabilità è la debolezza del sistema di protezione e un attacco è lo sfruttamento di una vulnerabilità per compromettere l’integrità, la confidenzialità e la disponibilità di una risorsa.

Esempi di vulnerabilità in C:

1. Il C non controlla che l’accesso agli elementi di un array sia limitato agli indici consentiti (da 0 a DIM-1) quindi se io per un array a[10] facessi un ciclo per a[i]=0 con i che va da 0 a 20 mi cancellerebbe anche le 10 aree di memoria che non fanno parte dell’area che potrebbero contenere informazioni utili.
2. Bisogna inoltre fare attenzione a non immettere in un array informazioni più grandi dello stesso, facendole così sconfinare in altre aree di memoria. Ad esempio per una stringa pwd[10] non possiamo inserire una stringa di 10 o più caratteri (ricordarsi che oltre alle lettere che immettiamo c’è anche il carattere ‘\0’

Per essere sicuri che non vengano sovrascritto dati, bisogna assicurarsi che non vengano letti più di n caratteri.

Questo può essere fatto attraverso le funzioni:

fgets(\_\_buffer\_\_,\_\_nMax\_\_,stdin) che legge da stdin (input) un massimo nMax di caratteri e li immagazzina nell’array buffer

sscanf(\_\_buffer\_\_,”%s”,\_\_s\_\_) che legge la stringa s dall’array buffer