Char \*s = stringa;

if(s == nullptr) -> controlla se il puntatore non punta a nulla

for(; \*s != 0; s++){ 🡨 l’ultimo elemento della stringa è “\0”

}

Int const è identico a const int

Se i puntatori sono costanti o puntano a valori costanti si dichiarano in modo diverso

Per puntare ad una costante:

int const \*p

o const int \*p

int \*const a

invece è diverso, è un puntatore costante che punta ad un intero

int const \* const a

puntatore costante che punta ad un intero costante

(leggere da destra a sinistra per capire meglio, meglio se in inglese)

Reference

NB: se passo un puntatore ad una funzione, quest’ultima ne riceve una copia. Quindi se da lei modifico a cosa punta il puntatore, il puntatore originale rimane invariato

Int a = 5;

function(a);

function(int &i){

i+=10;

//Qui invece non si creano copie, più leggero

}

Si indicano anche con

int& reference = a;

Se uso

Int v[] = {2, 3};

f(v);

f(int \*arr){

arr[1] = 10;

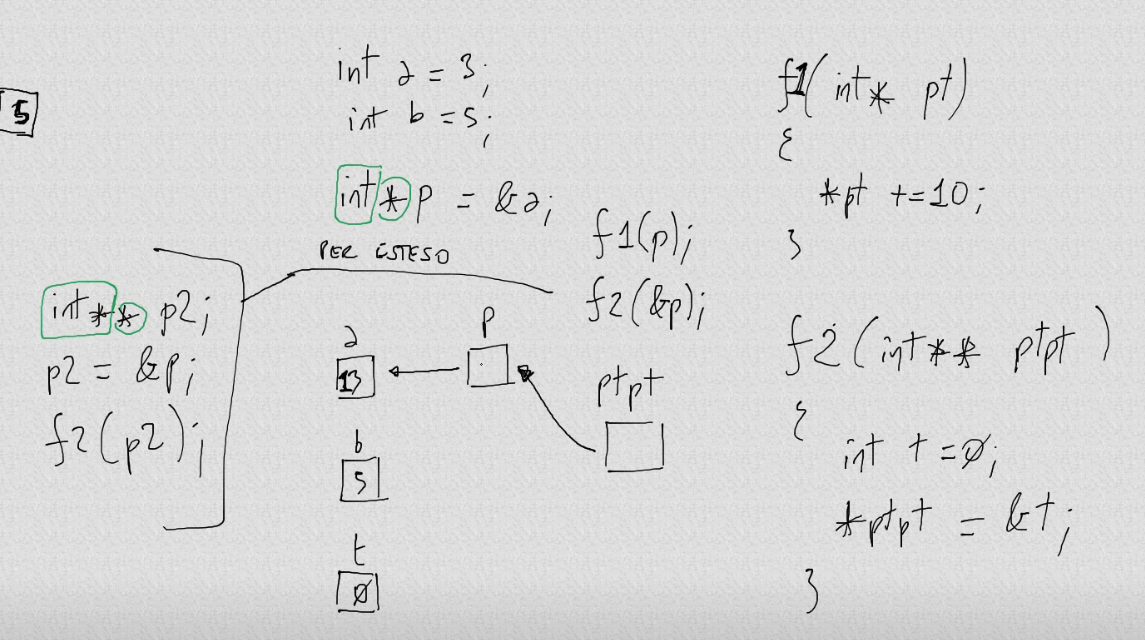
equivalente a

\*(arr+1) = 10;

qui si modifica direttamente l’array

}

//continuo sul file cpp



Sostanzialmente più devo modificare “in fondo” più asterischi devo mettere

🡪Semplificato usando reference

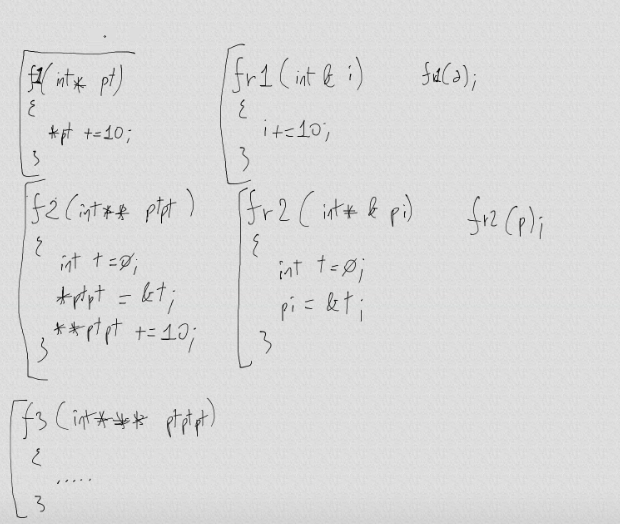
Int a =2;

fr(a);

fr(int &i){

i+=5;

} -> a diventa 7



Void f(int (&array)[10]){} //prende solo array di dimensione 10

Makefile

Linking: server per includere altri file senza ricompilarli ogni volta che viene compilato il principale

Si usano i file header -> contengono funzioni e/o variabili, ma solo i loro nomi e parametri (e tipo restituito)

Le vere funzioni poi le prende dai file (.cpp) indicati dal makefile (makefile della cartella è il più semplice)

* Da terminale serve solo scrivere “make” (fa partire il comando nel makefile)
* L’ “all” è chiamato target, è il nome dell’azione? (di standard il make richiama questo)
* Il “tab” prima del comando è importante perché fa capire che un comando per terminale

Un target può chiamare altri target

Exec: main.o func1.o 🡨avvia prima questi target (si possono anche avviare in cartelle es obj/func1.o

(è preferibile mettere il nome della cartella in una variabile se viene richiamato più volte)

Clean: 🡨ripulisce i file, lo si può richiamare via “make clean” come con ogni altro target

Make all = make

Struct

In C++ classi e struct sono interscambiabili, l’unica differenza è che la classe di standard mette tutto private, le struct mettono il public, per entrambe valgono:

* Incapsulamento
* Ereditarietà
* Polimorfismo

NomeStruct \*a = new NomeStruct(argomenti);

🡪Nell’heap crea l’ “oggetto” e nella stack il puntatore

E’ importante fare

Delete a;

Perché sennò quando si chiude il programma rimane l’oggetto nell’heap

Bool pg1Turn = (scelta == 2) ? false: true;

se scelta == 2 mette pg1turn false

altrimenti true

Class

Distruttore: viene chiamato con la distruzione dell’oggetto (ripulisce variabili) 🡪si indica con la tilde

Anche qui è importante il delete degli oggetti (oltre ai delete dei vari puntatori contenuti nella classe)

NB: se nella classe ci sono puntatori a altre classi/variabili di altre classi, forse non conviene distruggerle nel distruttore, lì semplicemente si mettono i puntatori uguali a nullptr

Array literal:

(\*char)”String” -> array 🡪non si fa il delete su questa memoria, non sta nella heap