**NB: guardare per bene il codice di questa lezione con le reference**

Prima credo i .o poi li linko in un exe

O meglio:

main.cpp include f.h (che contiene solo le dichiarazioni delle funzioni e variabili, in caso di classi si dichiara proprio la classe con variabili e funzioni, ma sempre soltanto le dichiarazioni)

🡪nel file .h posso anche metterci i valori alle costanti, le costanti invece vanno messe nei .cpp

Poi c’è il file f.cpp che contiene quello che fanno le funzioni

Poi il makefile fa il resto

Nel file f.cpp si usa il nome della classe come se fosse un namespace, ad esempio:

//Costruttore

F::F(){}

void f::FunzioneFileF(){}

Istruzioni pre-processore:

Se ho due .h interconnessi, ad esempio: personaggio.h e arma.h, in quello che usa l’altro (personaggio.h) dovrò includere l’altro (arma.h)

Problema: se ho tanti .h che includono arma.h, avrò tante reference ad arma.h  
Soluzione:

#ifndef \_ARMA\_H (if not defined)

//allora…

#define \_ARMA\_H

Class arma(){bla}

#endif

Inoltre dal distruttore del personaggio, conviene fare delete arma (se ha un puntatore ad un’arma che utilizza soltanto lui)

**NB:** è importante non avere un “cerchio” di import di .h, sennò si ha un ciclo infinito, **MAI FARLO**

Funzione const: promette di non modificare valori (per ottimizzare)

Inline: mi fa mettere la funzione su una riga

Es: inline const int getHp ()const{return hp;}

Così restituisce una variabile statica e promette di non modificarla (infatti è un get, deve funzionare così)

Per non fare copie posso anche fare

Const &int getHP()const{return hp};

così returno il puntatore, ma essendo const non può modificare il valore all’interno (risparmio memoria)

Nei cin>> per le stringhe si mettono puntatori ad esse es:

char \*stringa = new char[10];

cin>>stringa;

**OPERATORI**

Posso fare  
oggetto1 + oggetto2  
Se dico nella classe come ci si deve comportare con questo operatore

🡪posso fare overload degli operatori

Es:

Classe operator+(Classe oggetto2 //dato che starà a destra){ //A sinistra dell’operatore c’è il this

//vedere operatoroverloading.cpp per un esempio

}

**NB:** in c++ il this è un puntatore all’oggetto (bisogna deferenziare per restituirlo)

OPERATORI UNARI (OVERLOAD):

Classe operator++(){} //oggetto++

Classe operator++(int){} // ++oggetto

**CORRISPETTIVO DEL TOSTRING**

Cosa succede quando faccio cout<<oggetto; ??? Lo decido così:

Ostream& operator<<(ostream& output, const Classe &a){

Output<<a.quellochevosgliostampare<<”boh altra roba”;

return output;

}