PROGRAMMING KEY

PROGRAMMAZIONE MODULARE → vengo costituiti moduli che raccolgono funz. omogenee e nascondono metodi implementativi (es. prog. in C)
↳ posso quindi sfruttarlo senza conoscerne i retroscena

regola d'implementazione: **SEMPLICITÀ MODULI ↔ COMPLESSITÀ RELAZ. TRA ESSI**

1 → information hiding;
2 → accoppiamento, dati 2 moduli A e B hanno attributi in comune o funzioni; MINIMIZ.
3 → coesione, correlazione parti di uno stesso modulo; MAXIM.

MODULO IN C → · file header ".h", dichiarazioni;
· file sorgente ".c", definizioni;

es. CONTATORE

counter.h

```
#ifndef counter_h
#define counter_h
  void cnt_reset(void);
  ⋮
```

→ per non includere 2 volte (#ifndef counter_h)

} mandatory + optional functions

counter.c

```
#include "counter.c"
static int cnt=0;
void cnt_reset(void){
      cnt=0;
}
  ⋮
```

→ internal linkage, non deve essere modificato fuori dal modulo, visibilm. (static int)

→ per queste funz. ho linkage esterno e sono visibili

quindi se ho:

a.c

```
int var
int other
int fun(float){...}
```

posso →

b.c

```
extern int var *
void bar(){
  int n=var+1;
  int m=fun(1,5);
  int k=other-1;
```

→ DA ERRORE PERCHÈ VA DEF. TRAM *

```
*static int var
 static int other
   "       fun(float){...}
```

→

```
extern int var
  "        fun
```

→ DA ERRORE PERCHÈ HANNO LINKAGE INTERNO *

PROTOTIPO DI UNA FUNZIONE → nome, tipo di ritorno, numero e tipo per ogni argomento

PROCESSO → preprocessore (cpp): produce nuovo file sorgente e analizza le macro e le direttive
↓ gcc -E
compilatore (cc1): traduce codice sorgente in assembly (lexer → parser → semantica....)
↓ gcc -S
assembler (as): traduce da assembly a cod. macchina, produce file oggetto
↓ gcc -C a ogg. gcc da ogg a linker
linker (ld, ar, ranlib): combina file oggetto in eseguibile binario

PUNTATORI → var. in grado di contenere l'indirizzo di un'altra var., tramite 2 op (&) e (*)
possono puntare anche a funzioni;

STRUCT → consentono di aggregare dati non omogenei e non ordinati, dim per ogni elem som[mata] (posso accedere a tutte le var contemp.)

UNION → memorizzano come le struct ma dati che ricoprono lo stesso ruolo nel programma e ha dim del dato + grande (posso accedere ad 1 var alla volta)

TYPEDEF → per creare nuovi tipi

→ quando faccio **cast di puntatori** (→ cambio il tipo di variabile e non il valore) sto forzando la macchina a prendere dati dalla RAM

MODIFICATORI → consentono di modificare o non le variabili, il loro accesso e visibilità

es. CONST, STATIC, REGISTER, AUTO, VOLATILE

ENDIANNES → in macchine da 32 bit, consiste nell'ordine di memorizzazione dei Byte

→ LITTLE-ENDIAN: parte dal meno significativo;

→ BIG-ENDIAN: parte dal + signif;