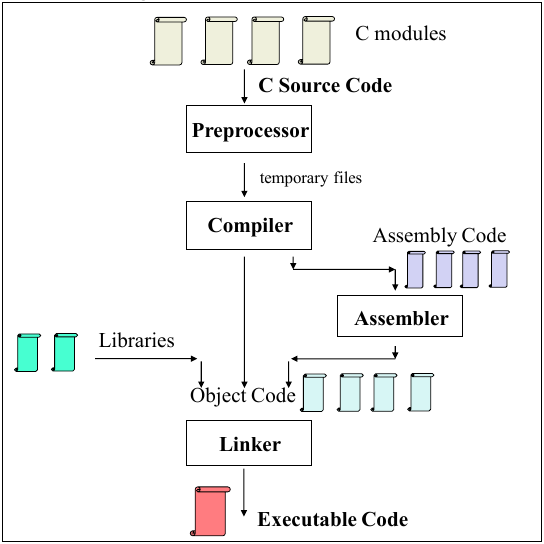
# ANSI-C

## Caratteristiche del linguaggio C

* Utilizzo frequente di chiamate a **funzioni**.
* **Debole controllo** sui tipi di dato.
* Linguaggio **strutturato**.
* Programmazione a basso livello facilmente disponibile.
* Implementazione dei **puntatori**.
* **Portabilità** sulla maggior parte delle architetture.
* Disponibilità di **librerie standard**.

## Il Modello di Compilazione per il C



### Il Preprocessore

Prende in input un file C e:

* Processa le **direttive al preprocessore**, ovvero:
  + **Inclusione di file**

#**include**

* + **Definizione di simbolo**

#**define** SIMBOLO DefinizioneDiSimbolo

#**undefine** SIMBOLO

* + **Definizione di Macro**

#**define** MACRO(X) DefinizioneDiMacro

* + **Compilazione condizionale su esistenza simbolo**

#**ifdef** SIMBOLO

Istruzioni C /\* eseguite se prima esiste #define SIMBOLO \*/

#**else**

Istruzioni C /\* eseguite se prima NON esiste #define SIMBOLO \*/

#**endif**

#**ifndef SIMBOLO**

#**endif**

* + **Compilazione condizionale su condizione costane numerica intera**

#**if** **Espressione1\_COSTANTE\_intera**

Istruzioni C /\* eseguite se Espressione1 è vera \*/

#**elif Espressione2\_COSTANTE\_intera**

Istruzioni C /\* eseguite se Espressione2 è vera \*/

#**else**

Istruzioni C /\* eseguite se sono false entrambe le espressioni \*/

#**endif**

* **Elimina i commenti** contenuti nel sorgente;
* **Genera un nuovo file** senza commenti e con tutte le dovute sostituzioni.

### Il Compilatore

Prende in input il codice generato dal Preprocessore e crea il codice Assembly.

### L’Assemblatore (Assembler)

Prende in input il codice Assembly e crea il codice oggetto, ovvero del codice comprensibile solo dalla macchina.

### Il Linker

Pende in input i vari files di codice oggetto e genera il file eseguibile, collegando tutti i riferimenti a variabili e funzioni da un file all’altro.

## Struttura di un programma C

**Direttive per il Preprocessore** #include …

#define …

**Definizione di tipi** typedef struct structName {

…

};

**Prototipi di funzioni** void f1(…);

**Dichiarazione delle variabili Globali** int var1;

**Dichiarazione del main** int main(void)

{

…

}

**Implementazione delle funzioni dichiarate** void f1(…)

{

…

}

# Librerie

Sono file che **contengono implementazioni**, in codice macchina di alcune funzioni. Possono implementare funzioni di base messe a disposizione dal sistema operativo oppure funzioni definite da utenti.

Le librerie si dividono in due categorie, che si distinguono in funzione di come vengono collegate all’eseguibile che lo deve utilizzare:

* **Statiche**: a link-time (quando si genera l’eseguibile) sono fisicamente accorpate ai moduli e inserite nell’eseguibile generato. Il linker deve sapere dove cercare sul disco le librerie statiche per copiarne il contenuto e inserirlo nell’eseguibile. Quindi l’eseguibile conterrà le librerie statiche.
* **Condivise** (o dinamiche): vengono collegate ad un programma in due passaggi. Durante la fase di compilazione, il linker verifica che tutti i simboli richiesti siano effettivamente collegati o al programma o a una delle sue librerie condivise. In ogni caso i moduli oggetto della libreria dinamica non vengono inseriti direttamente nel file dell’eseguibile. A run-time il *Dynamic loader* controlla quali librerie dinamiche sono state collegate al nostro programma, le carica in memoria e le attacca alla copia del programma in memoria.

# Makefile

Viene utilizzato per automatizzare le operazioni di compilazione e linking di progetti scritti in linguaggio C.

È un file di testo che contiene un insieme di regole (*make rules*) che nel loro complesso:

* Descrivono la struttura dell’albero delle dipendenze del progetto;
* Descrivono le regole di generazione dei vari target di ogni dipendenza del progetto.

Ciascuna make rule:

* Indica uno o più target a cui la regola si riferisce (*target list*);
* Elenca tutte le dipendenze da cui quel target dipende (*dependency list*);
* Elenca tutte le operazioni (*command list*) che devono essere svolte quando una delle dipendenze è più recente del target oppure quando il file target non esiste.

## Esecuzione

**Make -f Makefile *Target***

## Struttura

**Target1: Dependency\_list1**

TAB **Command\_list1**

RIGA VUOTA

**Target2: Dependency\_list2**

TAB **Command\_list2**

RIGA VUOTA