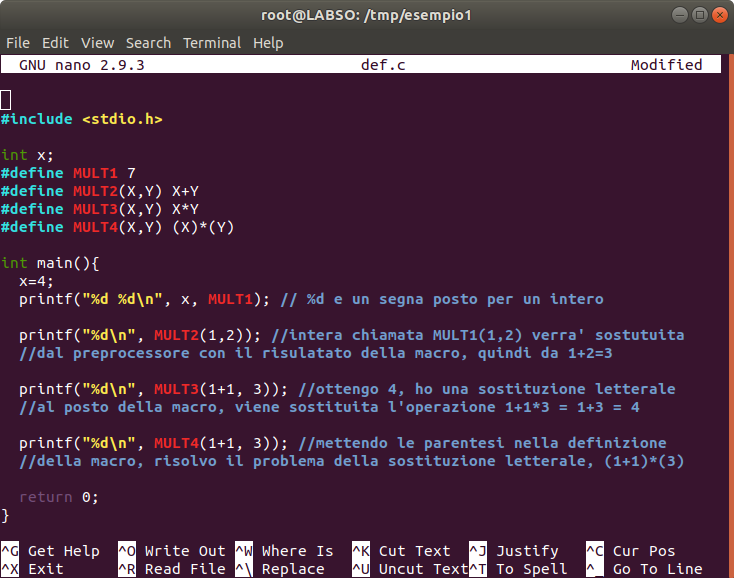
**Laboratorio Sistemi Operativi: Linguaggio C**

* È un linguaggio di **programmazione semplice** (ha poche keywords).
* La **sintassi del linguaggio** è **portabile tra diversi dispositivi e diversi sistemi operativi**, però più le funzioni che si realizzano scendono di livello (verso il livello macchina), più queste istruzioni si legano all’architettura rendendo così la semantica del linguaggio non propriamente portabile.
* **Partendo da un file sorgente** (lista di operazioni che si vogliono eseguire) e **arrivando ad un file binario** eseguibile, si attraversano vari **step**:
  + Dal **codice sorgente** (scritto in C) si passa ad un **codice intermedio** (il codice sorgente viene tradotto in **codice assembly**, codice più vicino a quello macchina) che dipende dall’architettura sottostante.
  + Da qui si andrà a realizzare il **codice oggetto**, una trasposizione del codice assembly in linguaggio macchina vero e proprio (sequenza di bit), ma non si ha ancora un eseguibile in quanto il file oggetto non è collegato all’ambiente dove verrà eseguito.
  + Dal **codice oggetto**, si passa al **codice binario eseguibile** (codice linkato al contesto corrente e pronto all’esecuzione).
* Per un’applicazione standard è **necessario definire** una funzione chiamata **main()** (a meno che non si stiano realizzando librerie).
* **#include <stdio.h>:**
  + È un **riferimento ad un file esterno** .h (sta per .header) che descrive delle funzioni che sono implementate in altri file (contiene la firma delle funzioni).
  + Siamo dicendo che vogliamo **includere** la definizione delle funzioni contenute nel file <stdio.h> (standard I/O) nel nostro progetto.
* Il compilatore contiene al suo interno **tre componenti**, che in cascata vengono eseguiti, uno dopo l’altro:
  + **Pre-processore**:
    - **Effettua una pre-analisi del codice sorgente**, in particolare **analizza le direttive** presenti nel codice (**analizza** il codice in **maniera letterale**, esegue le direttive che incontra).
    - Le **direttive** sono dei **pseudo comandi** (non sono istruzioni C) che vengono usate per **informare il pre-processore di eseguire certe operazioni**:
      * Tutti i commenti (**righe che iniziano con #**) seguiti (**senza spazi tra # e la parola**) da una **keyword**:
        + dal **compilatore C** vengono **visti** come semplici **righe commentate** (dunque ignorate).
        + dal **precompilatore**, queste righe **vengono** **interpretate come direttive** e vengono elaborate.
      * **#include <stdio.h>** è una delle **direttive** che dal compilatore C viene ignorata, ma non dal precompilatore.
    - **Direttiva** **#include <…>** o #**include “…”**:
      * Questa direttiva prende la riga dove si trova la keyword **#include** e la **sostituisce con il contenuto del file specificato**.
      * **<...>**
        + Indichiamo delle **librerie di sistema** o librerie installate nel nostro ambiente.
        + Il file indicato tra parentesi angolari viene **cercato** in una serie di cart**elle particolari** che dipendono dall’installazione che abbiamo effettuato oppure vanno cercate in una serie di cartelle che possiamo indicare da riga di comando aggiungendo **specifici flag** al comando gcc.
      * **“...”**:
        + Andiamo ad **indicare un percorso nel file system**.
        + Viene utilizzato per **includere file che fanno parte al nostro stesso progetto** **utilizzando un percorso relativo** alla cartella in cui ci troviamo.
    - **Direttiva** **#define**:
      * Definisce una **macro**, **una sorta di variabile**, solo che la sostituzione avviene in maniera letterale a tempo di compilazione.
      * **#define NAME VALUE**:
        + Il **valore è statico** e può contenere delle piccole computazioni/elaborazioni **(#define A (20/10) == #define A 2**).
        + Il **preprocessore** **ogni volta** che trova **NAME** nel **codice**, **sostituirà** questo **con** **il valore associato VALUE** in maniera letterale (prende e sostituisce).

****

* + - * Posso **definire delle macro direttamente da linea di comando**:
        + **gcc -DNAME=VALUE**
        + Questa invocazione è del tutto analoga ad avere all’inizio del file sorgente la macro che viene specificata da riga di comando.
    - **Direttiva** **#if COND #else #endif**:
      * Permette di fare dei **semplici controlli**.
    - **Direttiva** **#ifdef NAME e #ifndef NAME**:
      * Permette di verificare se una direttiva è stata definita o meno (a prescindere dal suo valore)

****

* + - * In questo caso, posso anche **non definire alcuna macro nel codice** e quando voglio eseguire il codice di debug, semplicemente utilizzare il **flag -DDEBUG**:
        + **gcc -DDEBUG=1** def.c -o def
        + **gcc -DDEBUG** def.c -o def (basta anche la definizione)
  + **Dopo aver analizzato ed eseguito tutte le direttive**, viene **generato il vero codice sorgente C**, che verrà successivamente compilato (tramite il comando **gcc -E “file.c”** è possibile **vedere le varie operazioni** e il codice sorgente generato).
  + **Compilatore**:
    - **Traduce** il **sorgente C pre-processato** dal preprocessore nel **linguaggio intermedio** (assembly) e questo **codice assembly in codice oggetto** (codice macchina).
    - Possiamo **invocarlo** usando i **flag** aggiuntivi:
      * **gcc -S “programma.c”**:
        + Genera un **file .s che contiene il codice sorgente tradotto in codice assembly** (codice che dipende dall’architettura).
      * **gcc -c “programma.s”**:
        + **Partendo dal file intermedio (.s) genera un file oggetto (.o)**, che contiene il codice macchina pronto per la compilazione finale.
  + **Linker**:
    - Trasforma il **codice oggetto** (.o) in **codice eseguibile**.
    - Possiamo **invocarlo** semplicemente **tramite il comando gcc** senza altri flag aggiuntivi:
      * **gcc** **traduce in modo automatico**, per renderlo un eseguibile, il codice:
        + **sorgente** con estensione **.c**
        + **assembly** con estesione **.s**
        + **oggetto** con estesione **.o**
      * **gcc “programma.o”**:
        + crea di default un eseguibile chiamato a.out
      * **gcc “programma.o” -o “programma”**:
        + rinomino l’eseguibile con il nome definito da riga di comando.