

Unità Didattica 1

# Linguaggio C

Fondamenti.  
Struttura di un programma.

# La storia del Linguaggio C

- UNIX (1969) - DEC PDP-7 Assembly Language
- BCPL - un OS facilmente accessibile che fornisce potenti strumenti di sviluppo prodotti a partire da BCPL. Si tratta di un assembler noioso, lungo ed incline agli errori
- Un nuovo linguaggio "B" come secondo tentativo (1970)
- Un linguaggio "C" totalmente nuovo come successore di "B" (1971)
- Dal 1973 UNIX OS, quasi totalmente scritto in C.

# La storia del Linguaggio C

- E' stato utilizzato per sviluppare il sistema operativo UNIX, viene comunemente usato per scrivere sistemi operativi
- Indipendente dall'hardware (portabile)
- Standardizzazione:
  - inizialmente sono state definite molte piccole varianti del C, incompatibili tra loro;
  - ANSI (American National Standards Institute) si è occupato della sua standardizzazione, nel 1989, standard aggiornato nel 1999.

# Caratteristiche del C

- **Linguaggio** a medio/alto livello. Basso livello di controllo degli errori nella fase di compilazione.
- **Variabili tipizzate**, con notevoli possibilità di conversione mediante il **type casting**, che permette di forzare una variabile a cambiare tipo;
- Abbina ad un **medio-alto livello di astrazione**, un buon controllo delle **operazioni a basso livello**.

# Fasi di traduzione ed esecuzione di un programma C

- **Traduzione:**

- **Preprocessing:** il preprocessore si occupa di includere altri file (es: librerie standard) e sostituisce simboli speciali che seguono particolari direttive (es: costanti simboliche);
- **Compilazione:** traduzione del programma in codice oggetto (linguaggio macchina (\*.obj));
- **Linking:** risoluzione dei riferimenti a funzioni e variabili definite altrove (es. in librerie standard o definite dall'utente), producendo una immagine eseguibile (progr.exe);

- **Esecuzione:**

- **Loading:** caricamento in memoria il codice eseguibile;
- **Esecuzione delle istruzioni;**

# Primo programma in C

```
#include <stdio.h>    /*# = Preprocessore C
                        precede le direttive che il compilatore
                        valuta prima di iniziare la traduzione
                        effettiva del programma (in questo caso
                        l'inclusione della libreria delle funzioni
                        standard di I/O (stdio.h))
                        */

void main(void)       /* Funzione principale, l'esecuzione di un
                        programma C consiste nella esecuzione del main */
{
    printf("Hello world\n"); /* Visualizza sullo standard output
                                (monitor) la frase fra " "
                                \n rappresenta un a capo
                                */
}
```

# Esecuzione

- Ogni programma comincia la sua esecuzione con la funzione **main**.
- I commenti sono inseriti tra i caratteri `/*` e `*/` (oppure `//` per commentare un'intera riga) e vengono ignorati in fase di compilazione.
- In **void main (void)**, la sintassi indica che il programma non ha argomenti di ingresso e che non restituisce alcun valore in uscita.

# Istruzioni e Librerie

- Il set di istruzioni del C è molto limitato: le primitive più comunemente utilizzate (es. I/O, matematiche) sono contenute nelle **librerie standard** del C sotto forma di **funzioni**.
- Se si utilizzano delle funzioni contenute in una certa libreria, questa deve essere inclusa mediante la direttiva **#include** del preprocessore.
- Es:
  - la funzione di output **printf()** è contenuta della libreria standard **stdio.h**;
  - Per usare la funzione deve essere presente la direttiva di inclusione:  
**#include <stdio.h>**

# Preprocessore

- # è il simbolo con cui iniziano i comandi del preprocessore.
- Tali comandi non sono terminati da “;”
- Attraverso il preprocessore si esprimono direttive al compilatore;
- Principali direttive:
  - Inclusione librerie;
  - Definizione delle costanti simboliche e delle macro;
  - Compilazione condizionale del codice.

# Preprocessore: #include

- La direttiva **#include**: consente di includere la copia di un file specificato.
- Esistono due forme:
  - **#include <nome\_file>**  
viene utilizzata per includere file di intestazione (*header file*) di una libreria standard, che sono memorizzati in directory standard (dipendenti dall'implementazione del compilatore);
  - **#include "nome\_file"**  
cerca il file nella directory corrente, altrimenti in directory standard (perciò è utilizzata per includere *header file* definiti dal programmatore).

# Preprocessore: #define

- La direttiva **#define** è utilizzata per definire costanti simboliche e macro.
- Il formato è:
  - #define [*identificatore*] [*testo\_di\_sostituzione*]
- All'interno del file in cui è presente, tutte le successive occorrenze dell'*identificatore* saranno automaticamente sostituite dal *testo\_di\_sostituzione*, prima della compilazione.

# Preprocessore: #define

(per definizione di costanti simboliche)

- #define PI 3.14159

sostituirà tutte le occorrenze della costante simbolica PI con quella numerica 3.14159

# Preprocessore: #define

(per definizione di macro)

- Le macro possono essere definite con o senza *argomenti*.
- Una macro senza argomenti viene elaborata come una costante simbolica.
- In una macro con argomenti, essi saranno rimpiazzati all'interno del testo di sostituzione e solo in seguito sarà *espansa* la macro: ovvero, il testo di sostituzione rimpiazzerà la lista degli identificatori e degli argomenti all'interno del programma.
- **Esempio:**
  - `#define AREA_CERCHIO(x) ( PI * (x) * (x) )`  
ogni volta che nel programma appare `AREA_CERCHIO(r)` il valore di `r` sarà usato al posto di `x`, la costante `PI` sarà rimpiazzata dal suo valore (definito precedentemente) e la macro verrà espansa all'interno del programma.

# Preprocessore: #define

(per definizione di macro)

- **Esempio 1:**
  - **area = AREA\_CERCHIO(4)**sarà espanso in
  - **area = ( 3.14159 \* (4) \* (4) );**
- **Esempio 2:**
  - **area = AREA\_CERCHIO(c+2)**sarà espanso in
  - **area = ( 3.14159 \* (c+2) \* (c+2) );**
- **Vantaggi nell'uso delle macro:**
  - risparmio nella definizione di una funzione;
  - miglioramento nella leggibilità del programma.

# Preprocessore:

## Compilazione condizionale

- Consente al programmatore di controllare la compilazione del codice del programma.
- Le direttive condizionali del preprocessore vengono valutate come espressioni costanti intere.

- **Esempio:**

```
#define DEBUG
```

```
#ifdef DEBUG
```

```
    printf("Variabile x = %d\n", x );
```

```
#endif
```

l'istruzione printf viene compilata (ed eseguita) solo nel caso in cui la variabile DEBUG sia definita.

# Variabili

- In C ogni variabile è caratterizzata dai seguenti aspetti:
  - Tipo;
  - Classe di memoria.
- Assegnare un tipo ad una variabile significa assegnarle il dominio dal quale assume i valori;
- La classe di memoria determina la durata della vita (*ciclo di vita*) e l'ambito di visibilità (*scope*) delle variabili.

# Tipi di dati

- **Tipi di base**
  - **char**: carattere
  - **int**: intero
  - **float**: virgola mobile, singola precisione
  - **double**: virgola mobile, doppia precisione
  - **void**: indica che il dominio della variabile è l'insieme vuoto
- **Qualificatori:**
  - **unsigned**: (es: unsigned int, unsigned char)
  - **short**: (es: short int)
  - **long**: (es: long int, long double)

# Dichiarazione di Variabili

- `[tipo_var] var0, var1, ...;`
- E' possibile inizializzare una variabile al momento della dichiarazione:

– Esempi:

```
int i, j, k=1;  
float x=2.6, y;  
char a;
```

# Dichiarazione di Costanti

- La definizione di identificatori per le costanti, oltre che con il comando di preprocessore `#define`, può avvenire usando il modificatore `const`:

```
const tipo nome_costante = valore ;
```

- **Esempi:**

```
const double e = 2.71828182845905;
```