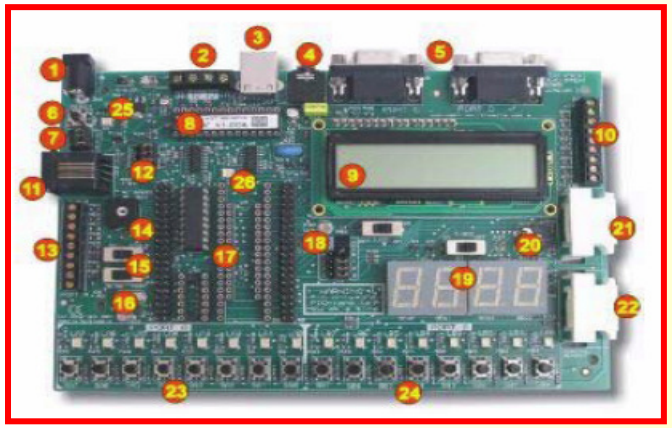
# Università degli studi di Cassino

**Corso di Laurea in**
**Ingegneria della Produzione Industriale**




# Corso di Informatica Applicata

## Lezione 5

Ing. Saverio De Vito
e-mail: saverio.devito@portici.enea.it
Tel.: +39 081 7723364

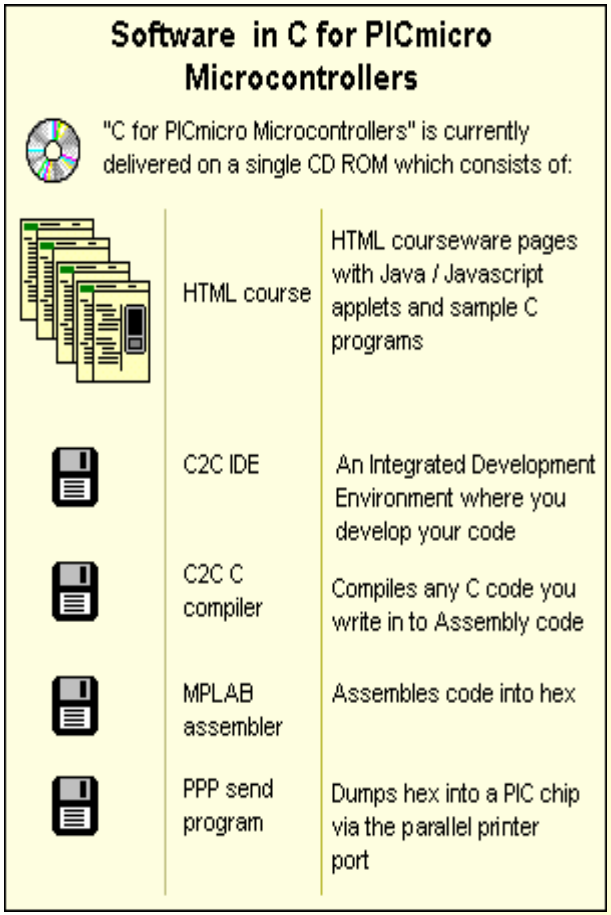
# Breve Riepilogo

- Introduzione alla architetture a microprocessore
- Abbiamo inquadrato le architetture PIC
- Abbiamo introdotto i linguaggi assembly in generale
- … e quello del PIC16F84A
- Abbiamo progettato, codificato, compilato, simulato l' esecuzione di semplici programmini in Assembly PIC16F84A
- ….. Ed ora cosa ci aspetta?

# Linguaggio C

- Appunti dal Corso
- Dispense dal Corso
- Materiale contenuto nel CD Free
- Test consigliato (opzionale):
  - Kernighan & Ritchie “Linguaggio C”

# Strumenti C




C2C IDE :

- Codifica
- Compilazione in Assembly

MPLAB:

- Compilazione Assembly
- Esecuzione

PPP:

- Programmazione PIC

# Linguaggio C: un po' di storia

- Nasce negli AT&T Bell labs…..
- …… come evoluzione del B (Thompson 1970)……
- …… dal lavoro di D. Ritchie (1972), che rese più efficiente il B e lo potenziò con nuove features……
- Primo grosso progetto sviluppato in C : il sistema operativo UNIX per PDP-11.
- I tempi di sviluppo estremamente (ed inaspettatamente) rapidi fecero subito decollare la popolarità di C per la riscrittura di kernel UNIX
- Alla fine degli anni 70 la diffusione aumentò ancora, anche se ci trovavamo ancora in ambienti relativamente ristretti. Il C allora era ancora appesantito dalla retrocompatibilità con i suoi predecessori. Nel 1978 nasce il white book, il K&R, il libro che fornirà la leva per la standardizzazione del C e la sua ulteriore espansione.
- Negli anni 80 il C esce dagli ambienti relativamente ristretti dove veninva ampiamente utilizzato per esploder sul mercato general purpose; a causa della sua notevole efficienza cominciò a rubare quote a linguaggi assai diffusi allora anche in ambineti quali il calcolo ad alte prestazioni.
- Nell' 89 nasce il primo standard ANSI e quasi contemporaneamente avviene la standardizzazione dell' interfaccia con il sistema operativo (POSIX)
- L' impatto del C sull' informatica è ormai globale, diviene il linguaggio più usato ed occupa sempre di più nicchie specialistiche fino a saturarle (programmazione embedded e real time)
- Nel 99 l' ultimo standard ANSI (ANSI99)

# Linguaggio C

Un programma scritto in C è composto sostanzialmente da una sequenza di statements (frasi) o compound statements (frasi composte) eventualmente raggruppate in funzioni.

I compound statements sono gruppi (sequenze) di statements racchiusi in parentesi graffe “{“.

Una funzione è di fatto una lista di compound statement eseguiti in sequenza, ogni funzione “ritorna” un valore (anche vuoto) in uscita. Questa sequenza possiede un intestazione detta header, che dà informazioni sul nome della funzione ed il nome, il numero e la tipologia dei valori di ingresso, se esitono; infine dà informazioni riguardo la tipologia del valore di ritorno.

Ogni programma C deve contenere una funzione denominata ***main***

# Linguaggio C

Le istruzioni (statements) sono tipicamente:

- chiamate a funzioni.
- espressioni.
- Istruzioni o sequenze con esecuzione condizionale while, for, if

# Linguaggio C

- Le istruzioni (statements) agiscono su **variabili**
- Le variabili posseggono:
  - Un nome
  - Un tipo (la descrizione dell' insieme dei valori ammissibili)
  - Valore
  - Un puntatore alla prima locazione di memoria che contiene il valore
  - La classe di memorizzazione

# Linguaggio C

Come in assembly è possibile definire dei commenti mediante l' utilizzi di opportuni tag:

/* Questo è un commento */

# Linguaggio C

Un semplicissimo programma C:

```
/* il nostro primo programmino C */
void main()
{
 /* Questo programma non fa nulla !! */
}
```

Void è una parola chiave che indica il nulla……. In questo caso la funzione main non ha alcun valore di ritorno.

# Linguaggio C

In assembler dovevamo aver cura di ricordare l' indirzzo in cui era immagazzinato il valore di una variabile, in C le variabili, come già accennato, posseggono un nome.

Ci si riferisce alle variabili tramite questo nome, associato ad un tipo al momento della *dichiarazione* della variabile.

```
int pippo;
```

La variabile pippo è un intero, in ANSI C può contenere valori interi appartenenti all' intervallo [-32768, 32767].

Il C non azzera il contenuto della variabile, semplicemente alloca una o + locazioni di memoria per contenere il valore della variabile, le variabili appena dichiarate andrebbero, prima di essere usate, inizializzate.

# C for PIC16F84A

Il PIC16F84A come sappiamo nn dispone ne dell' HW né del SW relativo al supporto per un aritmetica floating point.

Generalmente, invece, il C (standard ANSI et al.) supportano un tipo float destinato ad immagazzinare valori di tipo Floating Point.

E' previsto un tipo double in grado di immagazzinare valori a virgola mobile in doppia precisione.

Come risultato delle limitazioni intrinseche della nostra piattaforma di riferimento il C per PIC16F84 non supporta il tipo float.

Esistono ovviamente dei workaround, basati su codifiche ad hoc, per trattare valori non interi (es.: scalare i valori).

# C per PIC16F84A

Gli interi andrebbero adoperati con “parsimonia”, per quanto riguarda la programmazione del PIC target. Occupando almeno due byte hanno un aritmetica più complessa da tradurre per il compilatore.

E’ disponibile (anche per il C standard) il tipo char, quest’ ultimo è allocabile in un solo byte e quindi in una sola locazione di memoria del File Register del PIC16F84A.

Ove possibile, la scelta del tipo intero da utilizzare per una variabile ricadrà sul tipo più semplice, ovvero il char. Con il char, come ovvio, possiamo lavorare con valori compresi tra 0 e 255.

# C per PIC16F84A

Fondamentale per l' utilizzo delle variabili è il concetto di "scope" o visibilità.

- In quali blocchi di programma la variabile è visibile?

Semplificando possiamo al momento della dichiarazione modificare lo scope di una variabile anche semplicemente scegliendo il "blocco" in cui dichiararla, ossia renderla nota al compilatore.

Una variabile "locale" è visibile, semplificando, all' interno del blocco in cui è dichiarata, le variabili locali vanno dichiarate all' inizio del blocco di appartenenza.

Es.:

```
 void main(void)
{
  int c=0;   /* dichiarazione ed inizializzazione, visibile all' interno del blocco main */
  int pippo, pluto;      /* visibili all' interno del blocco main */
  pippo = 1;
  paperino =3;           /* errore  dove è dichiarata paperino ?*/
  int pippobaudo;        /* errore, non puoi dichiarare qui una variabile !  */
}
```

# Assegnazione et al.

L' assegnazione del valore ad una variabile avviene tramite l' operatore = :

```
pippo= 2;
pippo= pippo + 1;
pluto= pippo;
pluto+=1; /* ? */
pluto++;
pluto=pluto +1;
/* quanto vale pluto ? */
```

# C for PIC16F84

Identificatori di base

080 : 80 in base ottale

0x80: 80 in base esadecimale

I char possono contenere caratteri, un carattere di fatto è codificato con un numero compreso tra 0 e 255 dal codice ASCII a 8 bit.

# Operatori Aritmetici C del PIC16F84

Sono supportati fra gli altri i seguenti operatori :

+

-

*

/

% (resto)

E' bene spendere qualche parolina sulla divisione in quanto:

•In c la divisione:

```
int pippo = 0;
int c =8;
int d =3;
pippo = c/d; /* o anche pippo= 8/3 */
```

Darà come valore risultante 2. Il compilatore riconoscerà i tipi di partenza come interi e provvedrà ad eseguire una divisione intera.

Se avessimo scritto x= 8.0/3.0, il compilatore avrebbe scelto una procedura o un' istruzione adatta a dividere due floating point ed il risultato sarebbe stato qualcosa come 2.6667.

Và ricordato che nel nostro caso, l' aritmetica fp non è supportata pertanto l' ultima considerazione non è applicabile.

# C: Operatori Logici Principali

- AND : operatore C &
- OR : operatore C |
- XOR : operatore C ^
- NOT : operatore C !
- ShiftR : operatore C >>
- ShiftL : operatore C <<
- Gli operatori suddetti agiscono bit a bit, ossia sono operatori bitwise

# Casting

La tipizzazione dei valori da parte del compilatore può essere forzata, possiamo in pratica, forzare il compilatore a considerare il contenuto di una variabile come se fosse appartenente ad un particolare tipo, diverso da quello di origine. Questo processo viene detto “coercizione”.

Il processo di coercizione è implicito e va considerato con molta attenzione, per il fatto che il comportamento dipende fondamentalmente dal tipo di compilatore, è spesso foriero di errori, spesso difficilmente individuabili a posteriori.

```
float c;
Int k1=2; int k2=3;
c = k1/k2; /* c=0.0 */
c=2/3; //c=0
c= 2.0/3.0;  /*c=0.66667*/
```

# Casting

Il processo di conversione di tipo può essere reso esplicito (casting) con la sintassi (tipo).

L' esplicitazione rende l' errore meno probabile e comunque ci libera dalla dipendenza dal compilatore. L' errore però è sempre in agguato!

```
int c = 2500;
char k=0;
int k1=2; int k2=3;
float d=0;
k= (char) c; /* k<>c */
d= (float) k1/(float)k2; /*c=0.66667*/
c= 2.3; // c=2
```

# Esecuzione condizionale

- IF statement:

```
if ( cond-expression )  compound-statement1;
   else                 compound-statement2;

if (0) { /* non verrò mai eseguito */ }
if (1) { /* verrò sempre eseguito */ }
if (cont-100)
     {
          /* se cont contiene il valore 100*/
          /* allora verrò eseguito */
     }
```

# Esecuzione condizionale (2)

Operatori logici principali per la valutazione di condizioni:

>= maggiore o uguale

<= minore o uguale

== uguale a (notare la differenza con l' assegnazione)

!= non uguale a

&& congiunzione tra più condizioni semplici

|| Or tra condizioni semplici

! Operatore Not su condizioni

# Operatori condizionali: esempi

Es.:

```
if (i>=0) {/* se i è maggiore o uguale a 0*/}
if (i!=0) { /* se i è diverso da 0 */}
if (k=i-30) { /* se i è diverso da 30 */}
```

Quest' ultimo caso è plausibile in C standard ma non viene accettato dal compilatore C per PIC16F84A

E questo ?

```
if (--cond) {} /*?*/

if (cond--) {} /*?*/
```

# Esecuzione iterativa

- Costrutto while

```
while (condizione)
    {
     /* eseguito fin quanto condizione è true ! */
    }
```

- Costrutto for

```
for (<precond>;<cond>;<postcond>)
    {
            /* eseguito fino a quando la condizione “cond” è true !*/
    }
```

# Funzioni

- Una funzione è una sequenza di istruzioni che viene eseguita a richiesta, dotata di un identificativo univoco, una lista di parametri di tipo specificato ed un valore di ritorno anch' esso tipizzato.

***void function** pippo(**int** i, **int** j);*

# Funzioni: Dichiarazione e Definizione

Una funzione, prima di essere utilizzata deve essere dichiarata, al pari delle variabili, devono essere cioè dichiarati il suo identificativo, l' elenco tipizzato dei parametri ed il tipo del valore di ritorno (anche void).

*char function pippo(int i, int j);*

Dichiarazione di una funzione di parametri i e j, entrambi interi, che alla fine dell' elaborazione restituisce al chiamante un valore di tipo char.

# Funzioni: Definizione

Il contenuto elaborativo della funzione deve essere ugualmente dichiarato in una parte del listato. La codifica di tale contenuto e la sua trascrizione all' interno del codice sorgente viene detta definizione.

```
int somma (char i, char j)
{
  int c=0;
  c = (int) i + (int) j;  /* return (int)i+(int)j */
  return c;
}
```

# Riepilogo

- Introduzione al C
- Un po' di Storia
- Modello di programmazione
- Variabili
- Operatori principali
- Costrutti condizionali
- Costrutti di iterazione