



Il linguaggio C

September 15, 2018

Introduzione al linguaggio C

- Il C è un linguaggio di programmazione di sistema utilizzabile sia su mini-calcolatori (e.g., Arduino) che su grandi elaboratori.
- Come ingegneri meccanici/energetici e aerospaziali avrete probabilmente la necessità di programmarne uno. Conoscere il linguaggio C è quindi fondamentale.

Simboli utilizzati in queste slides

-  precede il programma da digitare nel vostro ambiente di programmazione
-  precede l'output a terminale del programma stesso una volta che questo viene eseguito.

Il mio primo programma C



```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello world!");
5      return 0;
6  }
```

Stampa a video del programma:



```
1  Hello world!
```

Questa direttiva indica al compilatore che nel nostro programma verranno utilizzate delle funzioni esterne:

```
1  #include <stdio.h>
```

Per usare una libreria nel nostro programma occorre specificarlo all'inizio.

Ogni programma in C deve contenere una funzione principale, chiamata `main`:

```
1  int main() {  
2      i1;  
3      i2;  
4      /* Anche sulla stessa linea */  
5      i3; i4;  
6  }
```

Questa funzione contiene le istruzioni che verranno eseguite non appena il nostro programma viene caricato in memoria.

Le istruzioni sono caratterizzate da una **sequenza di esecuzione** (da sinistra a destra) e sono distinte tramite un **separatore** (punto e virgola). Dopo il separatore è ammesso andare a capo.

`int` specifica che il tipo del valore di ritorno della funzione `main` è un numero intero.

```
1  int main() {  
2      ...  
3      return 0;  
4  }
```

Deve essere `0` se non ci sono stati errori, `1` altrimenti.

Usata per stampare stringhe con formato al terminale

```
1  int main() {  
2      int c = 68;  
3      printf("Il codice %d corrisponde al carattere '%c'\n", c, c);  
4      return 0;  
5  }
```

Risultato:

```
1  Il codice 68 corrisponde al carattere 'D'
```

Ingredienti fondamentali di un programma C

Una **variabile** è una cella di memoria con un **nome** ed un **tipo**.

Il **tipo** di una variabile determina quali valori possono essere assegnati ad essa. Matematicamente potremmo dire che è il dominio di valori assegnabili.

Quando il programma viene compilato, viene controllato che l'uso di ciascuna variabile sia coerente con suo tipo.

Dichiarazione e definizione di variabili

Il nome di una variabile deve essere sempre preceduto dal nome del tipo:

```
1  ...  
2  int c = 68;  
3  ...
```

La definizione può essere, in generale, dopo la dichiarazione. **Mai prima!**

```
1  int c; /* Dichiarazione */  
2  ...  
3  c = 68; /* Definizione */
```

Ogni istruzione può essere:

- un assegnamento del valore di un'espressione ad una variabile e/o invocazione di funzione:

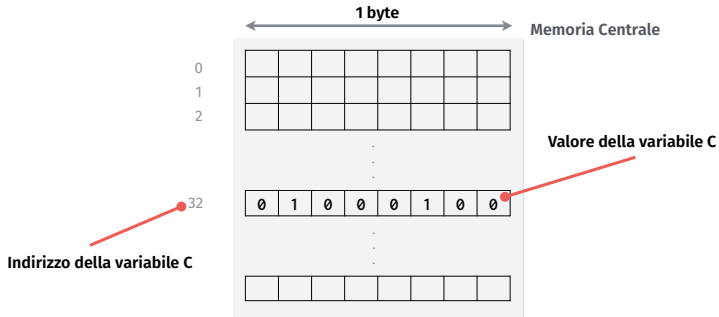
```
1  x = 3.0 + sin(x);
```

- un'istruzione di controllo che include ulteriori istruzioni da eseguire sotto certe condizioni, ad esempio:

```
1  if(x>0) {  
2      x = x - 1;  
3  }
```

Puntatori

```
1 int c = 68;  
2 int *indirizzo_di_c = &c;
```

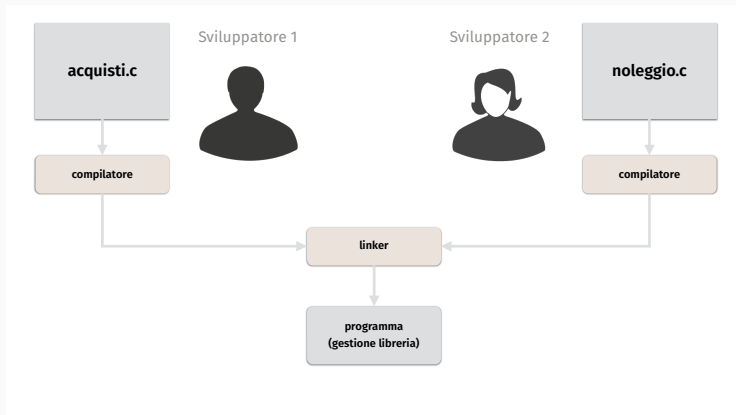


```
1  int max(int a, int b) {  
2      if(a > b) {  
3          return a;  
4      } else {  
5          return b;  
6      }  
7  }
```

Attenzione; bisogna usare l'istruzione **return** per ritornare un valore!

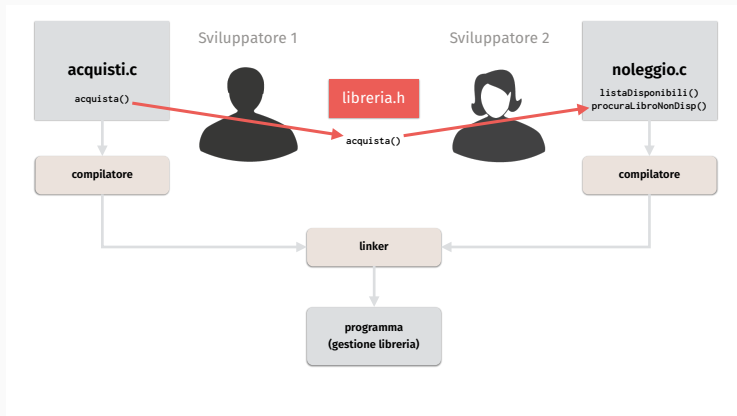
Files sorgente (.c) e catena di programmazione

Posso avere più funzioni in uno stesso file.



Headers e files sorgente

Per usarle devo avere un file header (.h) che contiene le dichiarazioni di funzioni necessarie.



Tipi built-in

Tipi built-in

- `int`: numeri interi
- `float`: numeri con virgola (singola precisione)
- `double`: numeri con virgola (doppia precisione)
- `char`: caratteri (sono interi che possono variare tra 0-255)

Esistono diverse varianti:

- `short int`: per numeri interi di piccole dimensioni
- `long int`: adatti a numeri interi di grandi dimensioni
- `unsigned`: per indicare che si utilizzerà la variabile solo per numeri positivi

Dichiarazione/inizializzazione

```
1  int a;    /* Dichiarazione */
2  a = 0;    /* Inizializzazione */
3
4  /* oppure */
5
6  int a = 0; /* Dichiarazione ed inizializzazione */
```

Shortcuts operatori su interi

```
1  int a,b,c;
2  ...
3
4  a = b / c;  /* divisione intera */
5  a = b % c;  /* resto divisione intera */
6
7  a += 3;     /* a = a+3 */
8  a -= 3;     /* a = a-3 */
9  a *= 3;     /* a = a*3 */
10 a /= 3;     /* a = a/3 */
11
12 a++;        /* a = a+1 */
13 a--;        /* a = a-1 */
```

Precisione singola

```
1 float f1 = 1.045;  
2 float f2 = .855;      /* --> 0.855 */  
3 float f3 = 4.5567e3;  /* --> 4556.7 */  
4 float f4 = 4.53e-2;   /* --> 0.0453 */
```

Precisione doppia

```
1 double d1 = .00005;  
2 double d2 = - 4.3e50; /* -4.3 * 1050, numero grande */  
3 double d3 = 4.2e-78;  /* 4.2 * 10-78, numero piccolo */
```


Scrivere un semplice programma che converte una temperatura da gradi Fahrenheit a gradi Celsius. La formula è:

$$C = \frac{(F - 32) \times 5}{9} \quad (1)$$

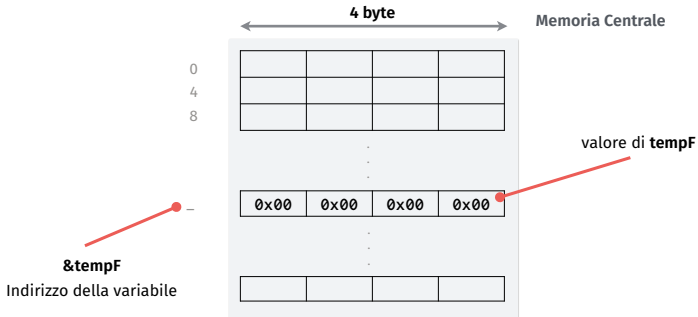


```
1  #include <stdio.h>
2
3  int main()
4  {
5      float tempF, tempC;
6      printf("Inserire temperatura in Fahrenheit: ");
7      scanf("%f",&tempF);
8      tempC = (tempF-32) * (5.0/9.0); /* Non usare 5/9 !!!!*/
9      printf ("Temperatura in Celsius %f\n",tempC);
10     return 0;
11 }
```

scanf

L'operazione `scanf` ha bisogno della posizione in memoria dell'elemento da andare a scrivere.

```
1 scanf("%f", &tempF);
```



Il C mette a disposizione il tipo `char` che può contenere un carattere.

Un `char` viene rappresentato solitamente con 1 Byte e contiene la codifica numerica del carattere: un valore nell'intervallo [0,255] (codifica ASCII);

Caratteri speciali:

- `'\n'` a capo
- `'\t'` tab
- `'\r'` carriage return
- `'\b'` backspace

Attenzione

```
1  char a;
2  char b, c = 'q'; /* Le costanti di tipo carattere si indicano con ' */
3  a = "q";          /* NO: "q" è una stringa, anche se di un solo carattere */
4  a = '\n';         /* OK: \n è un carattere a tutti gli effetti */
5  c = 'ps';         /* NO: 'ps' non è un carattere valido */
6  a = 75;           /* Che cosa succede? */
7
8  a = 'c' + 1;      /* a <-- 'd' */
9  a = 'c' - 1;      /* a <-- 'b' */
10
11 a = 20;
12 a *= 4;
13 a -= 10; /* a <-- 70 che corrisponde al carattere 'F' */
```

Tipi enumerativi



```
1  enum {falso,vero} condizione, condizione2;  
2  
3  condizione = falso;  
4  condizione2 = vero;  
5  
6  printf("condizione = %d \n",condizione);  
7  printf("condizione2 = %d \n",condizione2);
```



```
1  condizione = 0  
2  condizione2 = 1
```

Per definire un array, bisogna specificare il tipo, il nome e la dimensione:

```
1 float vendite[12];
```

È possibile inizializzare un array in fase di dichiarazione, specificandone tutti gli elementi fra parentesi graffe e separati da virgole:

```
1  float prezzo[4] = {13.4 , 11.10 , 20.9 , 30.4 };
```


In C è il programmatore a doversi preoccupare di non accedere a elementi dell'array non validi:

```
1  float prezzo[4];  
2  
3  ....  
4  
5  prezzo[4] = 46;    /* ERRORE! indici validi 0..3 */
```

Le matrici sono strutture dati bidimensionali Vengono rappresentati come array di array. Ad esempio:

```
1 float b[3][3];
```

Inizializzazione di matrici

Inizializziamo le matrici riga per riga:

```
1  float b[2][2] = {  
2      { 1.0, 2.0 },  
3      { 2.1, 3.1 }  
4  };
```

Accesso a matrici

```
1  float a[RIGHE][COLONNE];  
2  
3  for (i=0; i<RIGHE; i++)  
4      for (j=0; j<COLONNE; j++)  
5          scanf("%f", & a[i][j]);
```

In C non esiste un tipo di dato specifico per rappresentare i concetti **vero** e **falso**. Si usano gli **interi**. In generale, ogni valore diverso da zero è considerato vero.

Valore	Significato
1	VERO
1 + 1	VERO
0	FALSO
1 - 1	FALSO

Operatori relazionali e logici

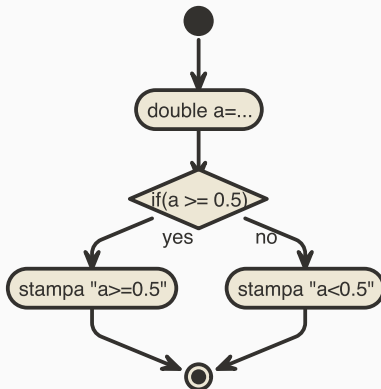
I valori logici sono prodotti da operatori relazionali o puramente logici

Espressione	Significato	Tipo di x	Tipo di y	Tipo risultato
$x > y$	x maggiore di y	numerico	numerico	logico
$x < y$	x minore di y	numerico	numerico	logico
$x \geq y$	x maggiore o uguale a y	numerico	numerico	logico
$x \leq y$	x minore o uguale a y	numerico	numerico	logico
$x == y$	x uguale a y	numerico	numerico	logico
$x != y$	x diverso da y	numerico	numerico	logico
$x \&\& y$	$x \wedge y$	logico	logico	logico
$x y$	$x \vee y$	logico	logico	logico
$! x$	$\neg x$	logico	—	logico

Strutture di controllo

If/then/else

L'istruzione di controllo `if` permette di eseguire delle istruzioni in maniera condizionale.





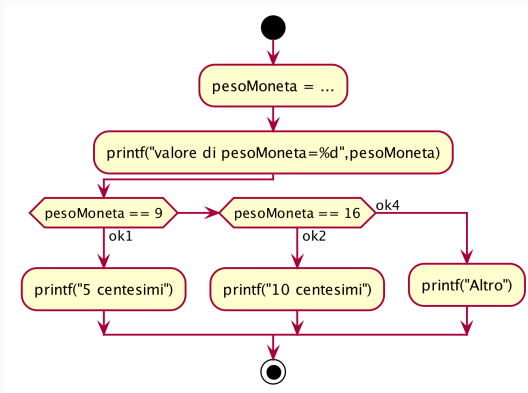
```
1 printf("valore di a=%lf\n",a);
2 if(a >= 0.5) {
3     printf("a maggiore di 0.5\n");
4 } else {
5     printf("a minore di 0.5\n");
6 }
7 printf("Finito\n");
```



```
1 .200000 non e" maggiore di 0.5
2 Finito
```

Switch/case

Se le alternative sono più di due, possiamo usare il costrutto di switch case



Switch case



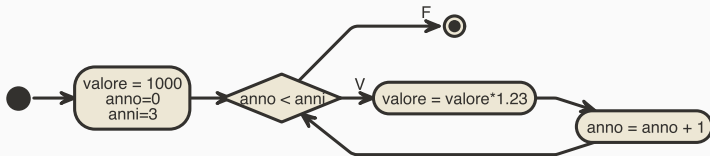
```
1  printf("valore di pesoMoneta=%d\n", pesoMoneta);
2  switch(pesoMoneta) {
3      case 9:  printf("5 centesimi\n"); break;
4      case 16: printf("10 centesimi\n"); break;
5      default: printf("Altro\n");
6  }
```



```
1  peso moneta: 19 - 20 centesimi
```

While

Supponiamo di voler calcolare gli interessi di 1000 euro su tre anni, con rata del 23%. Abbiamo bisogno di un'istruzione di controllo per ripetere il calcolo in un ciclo. Questa istruzione si chiama **while**.



While



```
1  /* calcola gli interessi di 1000 su tre anni (rata = 23%) */
2  int anno = 0, anni = 3;
3  float valore=1000.0;
4  printf("anno   valore\n");
5  printf("----   -----\n");
6  while(anno < anni) {
7      printf("%4d   %2.2f\n", anno, valore);
8      valore = valore * 1.23;
9      anno = anno + 1;
10 }
```



```
1  anno   valore
2  ----   -----
3      0   1000.00
4      1   1230.00
5      2   1512.90
```

For

```
1  printf("Stringa originaria: %s \n", nome);
2  printf("Stringa maiuscola: ");
3  for(int i=0; i<strlen(nome); i++) {
4      if(nome[i] > 'a' && nome[i] < 'z') {
5          printf("%c", nome[i] - 'a' + 'A');
6      } else {
7          printf("%c", nome[i]);
8      }
9  }
```

```
1  Stringa originaria: Vittorio
2  Stringa maiuscola: VITTORIO
```

Break



```
1  for(int i = 0; i < 10; i++) {  
2      if(i > 5) {  
3          break;  
4      }  
5      printf("%d ", i);  
6  }
```



```
1  0 1 2 3 4 5
```

Continue



```
1  for(int i = 0; i < 10; i++) {  
2      if((i % 2) == 0) {  
3          continue;  
4      }  
5      printf("%d ", i);  
6  }
```



```
1  1 3 5 7 9
```


Estendere il C con nuovi tipi

Definizione di tipo

Motivo: leggibilità e maggiore controllo sul programma

```
1  typedef int colore;  
2  colore coloreMacchina;  
3  coloreMacchina = 5;
```

Motivo: leggibilità e maggiore controllo sul programma

```
1  typedef enum {lun,mar,merc,gio,ven,sab,dom} giorno;  
2  giorno oggi;  
3  oggi = gio;
```

Stringhe

Gli array di tipo `char` sono detti anche stringhe. Dal momento che sono molto usati, il C mette a disposizione funzioni specifiche per questo tipo di dato:

```
1  char nome[4];  
2  
3  nome[0] = 'A';  
4  nome[1] = 'n';  
5  nome[2] = 'n';  
6  nome[3] = 'a';  
7  
8  typedef char stringa[30];  
9  stringa messaggio;
```

Stringhe costanti

In C, le costanti di tipo stringa si rappresentano come una sequenza di caratteri racchiusi tra “

- E.g. “anna” è una costante di tipo stringa

L’inizializzazione può avvenire in fase di dichiarazione:

```
1  typedef char stringa[30];  
2  stringa messaggio="prova";
```

Lettura e scrittura stringhe

```
1 char nome[30];  
2 printf("Inserisci il tuo nome: ");  
3 scanf("%s",nome);  
4 printf("Ciao %s!\n", nome);
```

La scanf assume che la stringa non contenga spazi!

Per leggere stringhe con spazi si utilizza `gets` (messa a disposizione dalle librerie `stdio.h`)

```
1  printf("Inserisci il tuo nome: ");  
2  gets(nome);
```

Carattere di terminazione

In C esiste un carattere speciale che indica la fine di una stringa: il carattere `'\0'`.

Quando la funzione `printf` individua questo carattere speciale smette di stampare a video gli elementi della stringa. `scanf` e `gets` lo aggiungono direttamente.

```
1 char msg[30];  
2 msg[0] = 'A';  
3 msg[1] = 'B';  
4 msg[2] = '\0';  
5 printf("%s",msg);
```

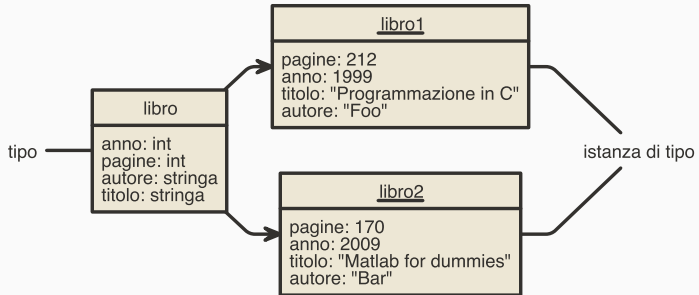

- La struct permette di rappresentare in maniera compatta ed incapsulata tipi di dati con una struttura complessa.
- Rispetto agli array, gli elementi non sono numerati ma hanno un nome e possono essere di tipo diverso.

Struct esempio



```
1  struct
2  {
3      int anno;
4      int pagine;
5      char autore[30];
6      char titolo[100];
7  } libro1, libro2;
8
9  libro1.anno = 1998;
```

Visualizzazione



Soluzione problematica

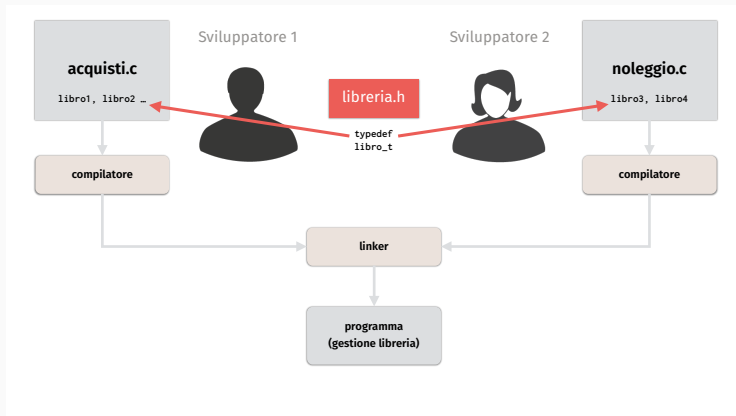
```
1  struct
2  {
3      int anno;
4      int pagine;
5      char autore[30];
6      char titolo[100];
7  } libro1, libro2, libro3, libro4, libro5;
8
9  libro1.anno = 1998;
```

La tecnica non è **scalabile**. Se devo aggiungere un libro al programma devo modificare sempre questo file. La **typedef** ci viene in aiuto anche in questo caso.

Soluzione con typedef

```
1  /* libreria.h */
2  typedef struct {
3      int anno;
4      int pagine;
5      char autore[30];
6      char titolo[100];
7  } libro_t;
8
9  /* acquisti.c */
10 libro_t libro1, libro2;
11
12 /* noleggio.c */
13 libro_t libro3, libro4;
```

Soluzione con typedef

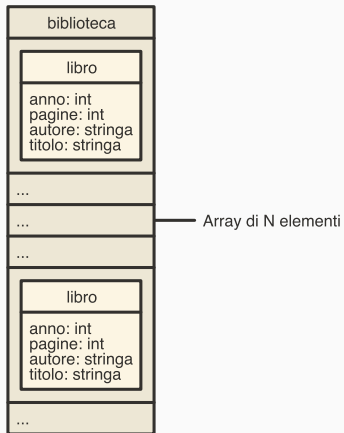


Inserimento di dati

```
1  libro_t mio_libro;  
2  printf("Inserire anno: ");  
3  scanf("%d", & mio_libro.anno);  
4  printf("Inserire num pagine: ");  
5  scanf("%d", & mio_libro.pagine);  
6  printf("Autore: ");  
7  gets( mio_libro.autore );
```

Array di struct

```
1  typedef struct
2  {
3      int anno;
4      int pagine;
5      char autore[30];
6      char titolo[100];
7  } libro_t;
8
9  libro_t biblioteca[N];
```

```
1  libro_t l1, l2;  
2  ...  
3  l1=l2;  
4  ...
```

Non è possibile effettuare il confronto con l'operatore ==.

Funzioni e procedure

Funzioni in C

Posso definire funzioni C proprio come in Matlab:

```
1  /*
2     _____ tipo di ritorno
3     /  _____ nome della funzione
4     /  /
                                           */
5  int max(    int a, int b    ) {
6      if(a > b) { // \      \__ inizio corpo funzione
7          return a; // \_____ parametri ingresso
8      } else {
9          return b;
10     }
11 }
12 // \_____ fine corpo funzione
```

E invocarle con una sintassi simile:

```
1  int m;  
2  ...  
3  
4  m = max(2, 1)
```

Procedure in C

Una procedura è una funzione che non ha valori di ritorno (`void`):

```
1  /*
2     _____ tipo di ritorno
3     /          _____ nome della funzione
4     /      /
                                           */
5  void stampa (int a, int b.....) {
6              // \
7              //  \_____ parametri ingresso
8  ...
```

Definite le procedure quando dovete stampare qualcosa (e quindi non avete bisogno che venga fatto nessun calcolo).