



**Laboratorio di Algoritmi e  
Strutture Dati**

**Aniello Murano**  
<http://people.na.infn.it/~murano>

Murano Aniello  
LASD 05/06 Seconda Lezione

1



**Il linguaggio C**

Murano Aniello  
LASD 05/06 Seconda Lezione

2

## Indice (Prima parte)

- Cenni storici
- Caratteristiche generali
  - Identificatori
  - Struttura di un programma C
  - Tipi base
  - Variabili e costanti
- Istruzioni elementari
  - Espressioni ed operatori
  - Assegnazioni
  - Input/output
- Istruzioni di controllo del flusso
  - Istruzioni condizionali
  - Iterazioni



## Cenni Storici

- Il linguaggio C è stato sviluppato intorno al 1972 nei Bell Laboratories AT&T americani, da Dennis Ritchie.
- E' nato come linguaggio di sviluppo del Sistema Operativo UNIX.
- Gli antenati del C possono essere riuniti in linea genealogica:
  - **Algol 60** 1960 (Comitato Int.)
  - **CPL** 1963 (Cambridge)
  - **BCPL** 1967 (Cambridge)
  - **B** 1970 (Thompson)
  - **C** 1972 (Ritchie)



## Dall'Algol al C

- L'Algol presentava le seguenti caratteristiche:
  - regolarità della sintassi,
  - struttura in moduli,
  - ma era particolarmente **complesso**.
- Il CPL (Combined Programming Language) ed il BCPL migliorarono molte caratteristiche dell'Algol senza però riuscire ad eliminare la complessità.
- Il linguaggio B (Thompson) era molto legato alla struttura dell'hardware.
- Ritchie seppe riassumere le migliori caratteristiche di questi linguaggi.



## Evoluzione

- 1978: Definizione precisa del linguaggio C (B.W. Kernigham & D. Ritchie)
- 1983: Standardizzazione da parte dell'ANSI
- 1986: Objective C (Cox), C++ (Stroustrup)
- 1999: Ultima release dello standard ANSI



## Perché programmare in C

- Portabilità del Codice e del Compilatore
- Codice generato molto efficiente
- Facilità di accesso al livello "macchina"
- Interfacciamento completo al S.O. UNIX
- Varietà di operatori di linguaggio
- Strutture dati potenti
- Non complesso (poche keywords)
- Modularità e Riutilizzo



Murano Aniello  
LASD 05/06 Seconda Lezione

7

## Programmare in C

- Per scrivere un programma in linguaggio C, i passi fondamentali sono:
- Scrivere un codice sorgente in linguaggio C con un **editor**
- Produzione di codice eseguibile utilizzando un **compilatore**
- Dev C++ è un compilatore Free (under the GNU General Public License) <http://www.bloodshed.net/dev/devcpp.html>



Murano Aniello  
LASD 05/06 Seconda Lezione

8

## Struttura del Programma Sorgente C

- Un **programma sorgente C** è formato da uno o più blocchi chiamati **funzioni**
- La definizione di una funzione rappresenta la specificazione delle operazioni che dovranno essere svolte all'atto della chiamata.
- La definizione è costituita da due parti:
  - *intestazione dichiarativa*
  - *Corpo*
- L'**intestazione** definisce le regole di interfaccia.
- Il **corpo** specifica le operazioni da eseguire ed è formata da un insieme di istruzioni delimitato utilizzando le parentesi graffe { **corpo** }



## Le istruzioni

- Le istruzioni C terminano con un punto e virgola (;)
- Esempio: `printf("Stampa questa riga \n");`
- Generalmente, un'istruzione può essere interrotta e ripresa nella riga successiva, dal momento che la sua conclusione è dichiarata chiaramente dal punto e virgola finale.
- Raggruppamenti di istruzioni si fanno utilizzando le parentesi graffe { }
- Esempio: `<istruzione>; {<istruzione>; <istruzione>; <istruzione>;}`



## Struttura del Sorgente (2)

- Altri oggetti fondamentali di un codice sorgente in linguaggio C sono:
  - Direttive del preprocessore
  - Commenti
- Le direttive del preprocessore guidano alla compilazione del codice. L'uso più comune riguarda l'inclusione di codice esterno al file sorgente (librerie), composto da file che terminano con l'estensione ".h". Tali istruzioni iniziano con il simbolo "#".
- I commenti vengono indicati tra i simboli /\* e \*/.



## Esempio di Programma C

- Il seguente è un semplice esempio di programma scritto in linguaggio C

```
#include <stdio.h>
main() /* esempio di programma C */
{
    printf("Primo programma C\n");
}
```

- Questo programma ha una sola funzione "main()".
- In un programma C esiste un solo **main** e l'esecuzione del programma corrisponde alla chiamata di tale funzione.
- Si notino inoltre la parentesi graffa aperta "{" per l'inizio del corpo della funzione e la parentesi graffa chiusa "}" per la fine del corpo della funzione



## Standard C library

- Esaminiamo l'istruzione

```
printf ("Primo programma C\n");
```

- **printf** è il nome di una funzione il cui codice è già scritto ed inserito nella **standard C library "stdio.h"**.
- Questa libreria viene inclusa durante la fase di compilazione del programma attraverso la direttiva del preprocessore **#include <stdio.h>**.
- La libreria standard è necessaria alla gestione dei flussi di standard input, standard output e standard error.
- Altre funzioni appartenenti a questa libreria sono sono:
- **printf()**, **fprintf()**, **sprintf()**, **scanf()**, **fscanf()**, **sscanf()**, **getc()**, **gets()**, **getchar()**, **putc()**, **puts()**, **putchar()**, **fgetc()**, **fgets()**, **fputs()**, **fwrite()**, **fread()**, etc...



## Gli identificatori

- Gli identificatori di qualsiasi oggetto in un programma possono consistere di un numero qualsiasi di caratteri alfanumerici minuscoli o maiuscoli incluso il carattere **"\_"** (underscore).
- Il primo carattere deve necessariamente essere una lettera oppure il carattere underscore("\_").
- Il compilatore fa differenza tra lettere minuscole e maiuscole
- **Esempi validi:** `sp_addr sp2_addr F_lock_user _found`
- **Esempi non validi:** `20_secolo -pippo`



## Significato degli identificatori

- Un identificatore non è altro che il nome associato ad un oggetto
- Ogni identificatore possiede due attributi che lo caratterizzano:
  - classe di memoria
  - tipo
- La classe di memoria determina la durata della memoria associata all'oggetto
- Il tipo determina il significato dei valori assunti dall'oggetto



## Classi di Memoria

- Esistono due classi di memoria: *automatica* e *statica*
- La classe di memoria automatica è relativa a quegli oggetti locali ad un blocco (funzione o programma) che viene liberata non appena si raggiunge la fine di quel blocco.
- La classe di memoria statica è relativa a quegli oggetti locali ad un blocco od esterni a qualsiasi blocco che non viene liberata tra uscite ed entrate successive tra diversi blocchi.
- Se non esistono altre specificazioni ogni oggetto dichiarato in un blocco ha classe di memoria automatica, se la loro definizione è accompagnata dalla parola chiave "static", allora la loro classe di memoria è statica.
- Gli oggetti dichiarati all'esterno di qualsiasi blocco (variabili globali) hanno sempre classe di memoria statica.





## Tipi di dati

- Tutte le variabili devono essere dichiarate con il loro tipo prima di essere utilizzate
- Ci sono 4 tipi base in C:
  - **char** /\* carattere \*/
  - **int** /\* intero \*/
  - **float** /\* numero in virgola mobile \*/
  - **double** /\* float a doppia precisione \*/
- I valori tipici per la rappresentazione interna di questi tipi di dati sono:
  - char 8 bit
  - int 16 bit
  - float 32 bit
  - double 64 bit



## Variabili di programma

- Una variabile è una entità che può assumere un valore qualunque all'interno di un insieme di valori prestabilito.
- Seguono alcune dichiarazioni di variabili
  - **char** c, scelta; /\* due variabili di tipo "char" \*/
  - **int** size, index, contatore; /\* variabili intere \*/
  - **float** dare, avere; /\* variabili in virgola mobile \*/
  - **double** ex, cx; /\* float in doppia precisione \*/



## Parole chiave

- Alcuni identificatori sono parole riservate (**keywords**) del linguaggio e pertanto non possono essere usate come nomi di variabili.

- Le **keywords** del linguaggio C standard sono 32:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while



## Interi

- La dichiarazione di variabili intere deve essere posta all'inizio di un blocco di codice.
- **Esempio:** `int a,b,c;`
- Dopo che una variabile è stata dichiarata è possibile assegnarle un valore intero tramite l'operatore di assegnamento
- **Esempio:** `a = 100;`
- Avendo a disposizione un calcolatore che memorizza interi a 16 bit, i due valori estremi che possono essere assegnati ad un intero sono: -32768 e +32767



## Caratteri

- Le variabili di tipo carattere vengono dichiarate nel modo seguente:

`char anno, mese;`

- Per assegnare un valore carattere `A` ad una variabile `c` di tipo `char` usiamo la seguente sintassi

`c='A'`

- In effetti, nel linguaggio `C` i caratteri sono valori numerici ai quali per convenzione sono associate lettere dell'alfabeto, segni di interpunzione ed altri simboli alfanumerici.
- La codifica più comunemente usata è la codifica **ASCII** (American Standard Code for Information Interchange)
- Ad esempio la lettera `'A'` viene codificata come 65 mentre la minuscola `'a'` corrisponde a '97'



## Dati in virgola mobile

- I dati floating point sono una approssimazione dei reali, espressi come frazioni decimali.
- Per esempio: 3.14159, 2.71828
- Solitamente vengono utilizzati 32 bit per la rappresentazione interna di questi dati, di cui almeno i primi 6 sono i più significativi.
- Il range di variazione dei floating point è:  $1...e-39$  fino a  $1...e+38$



## Costanti

● In C esistono diversi tipi di costanti, in cui possiamo distinguere 5 tipi fondamentali:

- costanti **interi**
- costanti esplicitamente **long**
- costanti **carattere**
- costanti in **virgola mobile**
- **stringhe di caratteri** costanti



## Costanti carattere speciali

● Esistono alcune costanti particolari:

- New line (lf) '\n'
- Carriage return (cr) '\r'
- Backspace (bs) '\b'
- Horizontal tab (tab) '\t'
- Backslash (\) '\\'



## Operatori

- Gli operatori del linguaggio possono essere classificati in tre categorie: aritmetici, relazionali e logici.

- Aritmetici:

- "+" somma,
- "-" sottrazione,
- "\*" moltiplicazione,
- "/" divisione,
- "%" modulo



## Operatori(2)

- Relazionali

- "<" minore di,
- ">" maggiore di,
- "<=" minore o uguale,
- ">=" maggiore o uguale,
- "==" uguale a,
- "!=" diverso da

- Logici

- "&&" AND logico "&" AND bit a bit
- "||" OR logico "|" OR bit a bit
- "!" NOT logico "~" NOT bit a bit



## Operatori aritmetici e di assegnamento

- Si consideri l'espressione:
- `anno = anno + 1;`
- che significa: "prendere il valore corrente della variabile anno, sommarci 1 e memorizzare il risultato nuovamente in anno".
- Nell'espressione sono presenti due operatori
- "+" e "=" che implicano l'esecuzione dell'operazione in due passi.
- L'ordine di esecuzione è: somma e poi assegnamento.



## Operatori di incremento e decremento

- Gli operatori di incremento e decremento sommano o sottraggono 1 all'operando cui sono applicati.
- L'espressione: `a++` incrementa `a` di 1
- L'espressione: `a--` decrementa `a` di 1
- La posizione degli operatori di incremento e decremento può essere *prefissa* o *suffissa*.
- **Esempio:**
- `a=5`
- `b =++a; a=6 b=6`
- `a=5`
- `b =a++; a=6 b=5`



## Operatori di assegnamento composti

- Il C dispone di operatori di assegnamento che sono combinazioni opportune dell'operatore di assegnamento con operatori aritmetici

- Esempio:

```
a = a + 10;
```

```
a += 10;
```

- L'elenco degli operatori di assegnamento è quindi:

```
> =      +=
```

```
> -=     *=
```

```
> /=     %=
```



## Printf e Scanf con formattazione

- Si consideri il seguente codice

```
#include <stdio.h>
main()
{
    int a,b,c;
    printf("\nIl primo numero e");
    scanf("%d",&a);
    printf("Il secondo numero e");
    scanf("%d",&b);
    c=a+b;
    printf("Il totale e' %d \n",c);
}
```



## Printf e Scanf con formattazione (2)

Alcuni tipi di % che possono essere usati in ANSI C sono:

### Usual variable type Display

%c char single character

%d (%i) int signed integer

%e float or double exponential format

%f float or double signed decimal

%s array of char sequence of characters



Murano Aniello  
LASD 05/06 Seconda Lezione

31

## Strutture di controllo

### ● SEQUENZIALI

- *Statement semplice*
- *Statement composto*

### ● CONDIZIONALI

- `if (< expr.>) { } else { }`
- `switch(<expr.>) { case < cost> : .. }`

### ● ITERATIVE

- `while(<expr.>) { }`
- `for (..) { }`
- `do { } while(< expr.>)`



Murano Aniello  
LASD 05/06 Seconda Lezione

32



## Statement Semplici

- Uno statement semplice può essere :

- assegnamento
- espressione
- chiamata a funzione

- Esempio:

```
main()
{
  int x;
  x = - 456; /* assegnamento */
  x = x + 1; /* espressione */
  printf (" X = %d \n ", x); /* funzione */
}
```



Murano Aniello  
LASD 05/06 Seconda Lezione

33

## Istruzione "IF - ELSE"

- Lo statement "if - else" è usato per prendere delle decisioni all'interno di un programma

- Sintassi:

```
if ( espressione )
  istruzione;
```

- oppure

```
if ( espressione )
{
  istruzione_1;
  istruzione_2;
}
else
  istruzione_3;
```



Murano Aniello  
LASD 05/06 Seconda Lezione

34

## Esempio

- L'espressione `< expr >` viene valutata se risulta essere TRUE o NON ZERO viene eseguito il set di istruzioni corrispondente.

- Esempio:

```
if ( x > 0 )
{
  if ( k == m )
  y = m;
}
else
{
  y = 0;
  k = 100;
}
```



## Operatore Ternario

- L'operatore ternario è una forma compressa di
- **if-else.**
- Sintassi: `< expr > ? < expr1 > : < expr2 >;`
- Se `< expr >` è vero viene valutato solo `< expr1 >`
- altrimenti viene valutato `< expr2 >`



## Istruzione else-if

- Quando occorre effettuare una scelta plurima è possibile, ma non consigliabile, utilizzare l'istruzione else-if.

- Sintassi:

```
if (espressione)
<istruzione/i>
else if (espressione)
<istruzione/i>
else if (espressione)
<istruzione/i>
else
<istruzione/i>
```



## Istruzione switch

- L'istruzione switch consente di trasferire l'elaborazione a uno o più statement composti in funzione della valutazione di una espressione.

- Sintassi:

```
switch (<expr>){
case <cost_1>:
statement;
case <cost_2>:
statement;
case <cost_3>:
statement;
default
statement; }
```



## Istruzione switch

- Questa istruzione viene usata quando occorre effettuare delle scelte plurime.
- Occorre fare attenzione in quanto l'esecuzione delle istruzioni comincia e prosegue in maniera SEQUENZIALE a partire dalla condizione trovata di *espressione= cost\_x* e fino al termine dell'istruzione switch.
- Ciò garantisce una certa flessibilità, ed in particolare la possibilità di associare più possibilità ad un'unica etichetta.
- Per evitare l'esecuzione delle istruzioni successive a quelle corrispondenti alla condizione soddisfatta si usa il comando `break`



## Esempio

```
#include <stdio.h>
main()
{
    int i;
    printf("Enter a number between 1 and 3");
    scanf("%d",&i);
    switch (i)
    {
        case 1:
            printf("one"); break;
        case 2:
            printf("two"); break;
        case 3:
            printf("three"); break;
        default:
            printf("unrecognized number");
    } /* end of switch */
}
```



## Istruzione "while"

- L'istruzione **while** permette di eseguire una serie di istruzioni fintanto che una condizione iniziale rimane **TRUE**.

- Sintassi:

```
while ( <expr> ){  
    statement;  
}
```

- Esempio

```
a = 1;  
while ( a <= 100 )  
{  
    total += a*a;  
    a += 1;  
}
```



## Istruzione "for"

- L'istruzione **for** è molto simile al **while** precedente.

- Sintassi:

```
for ( <inizializza var>; <expr>; <aggiorna var> ) {  
    statement;  
}
```

- Equivalente alla struttura seguente che utilizza **while**:

```
< inizializza variabili >;  
while ( <expr> ) {  
    < statement >;  
    < aggiorna variabili >; }
```



## Esempio

```
main()
{
  int sum , i;
  sum = 0;
  for ( i = 1 ; i <= 10; i++) {
    sum += i;
  }
}
```



## Istruzione "do-while"

- L'istruzione `do while` è simile alla `while`; l'espressione viene valutata alla fine e non all'inizio del ciclo.
- Questo comporta che un ciclo `do-while` venga eseguito almeno una volta.
- Sintassi:

```
do {
  statement;
} while ( <expr>);
```

N.B: questo tipo di istruzione è comodo quando si desidera eseguire una serie di istruzioni almeno una volta.

