

Introduzione al C

Lez. 1

Elementi

Rossano Venturini

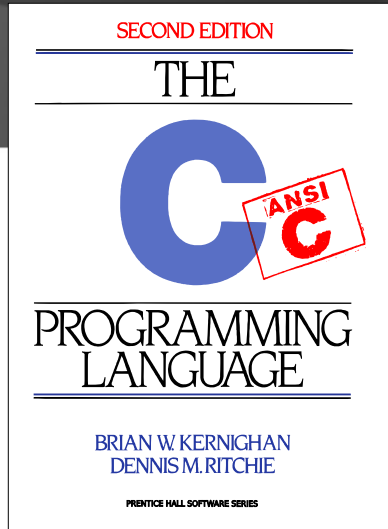
rossano@di.unipi.it

Pagine del corso

<http://didawiki.cli.di.unipi.it/doku.php/informatica/all-b/start>

<http://algoritmica.spox.spoj.pl/ALGLAB2013>

Lezioni di ripasso



- Mercoledì 11-13 Aula B
- Giovedì 16-18 Aula H/M
- Mercoledì 11-13 Aula B
- Giovedì 1*-1* Aula H

Introduzione al C

Gli strumenti che utilizzeremo in queste esercitazioni sono semplicemente:

- **Editing dei sorgenti**: editor di testo generico (e.g. Gedit per Linux)
- **Compilatore**: gcc per Linux, o tools di pubblico dominio per Windows (vedi pagina Web del corso)

Introduzione al C

Gli strumenti che utilizzeremo in queste esercitazioni sono semplicemente:

- **Editing dei sorgenti**: editor di testo generico (e.g. Gedit per Linux)
- **Compilatore**: gcc per Linux, o tools di pubblico dominio per Windows (vedi pagina Web del corso)

Struttura di un programma C

Direttive per il pre-processore:

```
#include ...
```

```
#define ...
```

```
#include <stdio.h> (generalmente necessaria)
```

Definizioni di variabili globali

```
...
```

```
int C;
```

Definizioni/Dichiarazioni di funzioni

```
...
```

```
int main() { /* esecuzione inizia qui */ }
```

Struttura di un programma C

Direttive per il pre-processore:

```
#include ...
```

```
#define ...
```

```
#include <stdio.h> (generalmente necessaria)
```

Definizioni di variabili globali

```
...
```

```
int C;
```

Definizioni/Dichiarazioni di funzioni

```
...
```

```
int main() { /* esecuzione inizia qui */ }
```

Struttura di un programma C

Direttive per il pre-processore:

```
#include ...
```

```
#define ...
```

```
#include <stdio.h> (generalmente necessaria)
```

Definizioni di variabili globali

```
...
```

```
int C;
```

Definizioni/Dichiarazioni di funzioni

```
...
```

```
int main() { /* esecuzione inizia qui */ }
```

Struttura di un programma C

Direttive per il pre-processore:

```
#include ...
```

```
#define ...
```

```
#include <stdio.h> (generalmente necessaria)
```

Definizioni di variabili globali

```
...
```

```
int C;
```

Definizioni/Dichiarazioni di funzioni

```
...
```

```
int main(){ /* esecuzione inizia qui */ }
```


Esempio: main()

Semplice main() senza istruzioni:

```
#include <stdio.h>

int main(){          /* definizione di main() */

    /* esecuzione inizia qui */

    /*     corpo della funzione
     *     ... codice ...
     */

    return 0;
}
```

Esempio: main()

Semplice main() senza istruzioni:

```
#include <stdio.h>

int main() {      /* definizione di main() */

    /* esecuzione inizia qui */

    /* corpo della funzione
     * ... codice ...
     */

    return 0;
}
```

Esempio: main()

Semplice main() senza istruzioni:

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

```
int main() {      /* definizione di main() */
```

```
    /* esecuzione inizia qui */
```

```
    /*      corpo della funzione  
     *      ... codice ...  
     */
```

```
→ return 0;
```

```
}
```

Esempio: “hello world”

Aggiungiamo una chiamata alla funzione di libreria *printf* (definita in *stdio.h*) per stampare una stringa:

```
#include <stdio.h>

int main(){
    printf("Hello, world!\n");
    return 0;
}
```

Esempio: “hello world”

Aggiungiamo una chiamata alla funzione di libreria *printf* (definita in *stdio.h*) per stampare una stringa:

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

```
int main(){
```

```
➔ printf("Hello, world!\n");
```

```
return 0;
```

```
}
```

Esempio: "hello world"

Aggiungiamo una chiamata alla funzione di libreria *printf* (definita in *stdio.h*) per stampare una stringa:

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

```
int main(){
```

➔ **printf("Hello, world!\n");**

```
return 0;
```

```
}
```

Le costanti stringa in C sono sempre specificate tra una coppia di apici

`\n` è la *sequenza di escape* che codifica il carattere di fine-linea

Compilazione ed esecuzione (Linux)

Utilizziamo gedit, per editare il sorgente del nostro programma:

```
> gedit hw.c &
```

Compilazione ed esecuzione (Linux)

Utilizziamo gedit, per editare il sorgente del nostro programma:

```
> gedit hw.c &
```

Compiliamo il precedente programma (hw.c) con gcc :

```
> gcc -o hw hw.c
```

The diagram illustrates the components of the compilation command. The text '> gcc -o hw hw.c' is shown. Two red boxes highlight the arguments '-o hw' and 'hw.c'. An arrow points from the box containing '-o hw' down to the word 'eseguibile'. Another arrow points from the box containing 'hw.c' down to the word 'sorgente'.

Compilazione ed esecuzione (Linux)

Utilizziamo gedit, per editare il sorgente del nostro programma:

```
> gedit hw.c &
```

Compiliamo il precedente programma (hw.c) con gcc :

```
> gcc -o hw hw.c
```

The diagram illustrates the compilation command `gcc -o hw hw.c`. The option `-o hw` is enclosed in a red box, with an arrow pointing to the word `eseguibile` (executable) below it. The source file `hw.c` is also enclosed in a red box, with an arrow pointing to the word `sorgente` (source) below it.

Risultato esecuzione:

```
> ./hw  
Hello, world!
```

Dichiarazioni e assegnamenti

Una dichiarazione introduce il nome di una variabile e il tipo di dati che conterrà:

```
int x;   int y=0;  
int a, b;
```

Il contenuto variabili si può inizializzare o modificare mediante *assegnamento* o *pre/post-incremento*:

(forma equivalente)

```
x = 0;  
x = x + 1;      x++; 0 ++x;  
x = x + y;     x += y;  
x = x*3;       x *= 3;
```

Tipi numerici di base in C

Tutti i tipi primitivi del C sono *numerici* :

- Gli operatori aritmetici standard (+, -, *, /, %) sono definiti su di essi
- Interi: **char, short, int, long**
 - **1 (char), 2 (short), 4/8 (int/long) bytes** a seconda dell'implementazione,
 - Il modificatore **unsigned** restringe a valori positivi
 - I *char* sono tipicamente usati per codificare caratteri ASCII: 'a', 'A', ... Lo specificatore **%c** stampa il simbolo codificato
- Floating-Point: **float, double**
 - **4, 8 bytes**, specificatore : **%f**

Branch in C: if-elseif-else

Costrutto condizionale di base in linguaggio C:

```
if (guardia) { /* blocco1 */ }  
else { /* blocco2 */ }
```

Esempio:

```
if (voto < 18) {  
    printf("respinto"); }  
else {  
    printf("promosso"); }
```

Branch in C: if-elseif-else

Costrutto condizionale di base in linguaggio C:

```
if (guardia)    { /* blocco */ }  
else if (guardia_2)  { /* blocco */ }  
.....  
else if (guardia_n)  { /* blocco */ }  
else { /* blocco */ }
```

Esempio:

```
if (voto < 18) {  
    printf("respinto"); }  
else {  
    printf("promosso"); }
```

Operatori logici e di confronto

Le guardie sono espressioni tipicamente ottenute combinando operatori di *confronto* (<, <=, >, >=, ==, !=) e *logici* (&&, ||, !):

Operatori logici e di confronto:

- | | | | |
|--------|--------------|-----------|-------------------|
| • && : | and | • <, <= : | minore (uguale) |
| • : | or inclusivo | • >, >= : | maggiore (uguale) |
| • ^ : | or esclusivo | • == : | uguaglianza |
| • ! : | not | • != : | diverso da |

Un esempio con condizionale più sofisticato:

```
if ( anno%4 == 0 &&  
    !( anno%100==0 && anno%400 != 0 ) ) {  
    printf("Bisestile");  
}
```

Operatori logici e di confronto

Le guardie sono espressioni tipicamente ottenute combinando operatori di *confronto* (<, <=, >, >=, ==, !=) e *logici* (&&, ||, !):

Operatori logici e di confronto:

- && : and
- || : or inclusivo
- ^ : or esclusivo
- ! : not
- <, <= : minore (uguale)
- >, >= : maggiore (uguale)
- == : uguaglianza
- != : diverso da

Un esempio con condizioni

Attenzione a non confondere:
x=y (assegnamento) con **x==y** (confronto)

```
if ( anno%4 == 0 &&  
    !( anno%100==0 && anno%400 != 0 ) ) {  
    printf("Bisestile");  
}
```

Rappresentazione di booleani

In C *non esistono tipi primitivi per i booleani*, bensì questi vengono codificati con interi:

- 0 rappresenta FALSO, qualsiasi valore $\neq 0$ rappresenta VERO

Gli operatori logici e di confronto sono operatori aritmetici a tutti gli effetti che restituiscono :

- 0 se sono falsi
- 1 se sono veri

Qualsiasi espressione aritmetica è utilizzabile come guardia:

```
Ex.: if (x)
    printf("x è diverso da zero");
```


Loops in C: while

Forma più semplice di loop in linguaggio C:

```
while ( /* condizione */ ) {  
    /* corpo del while */  
}
```

Esempio :

```
int counter=0;  
while (counter < 10) {  
    printf("%d\n", counter);  
    counter++;  
}
```

Loops in C: for

Loop con inizializzazione ed incremento:

```
for ( /* inizializzazione */ ; /* test */ ; /*  
incremento */ ) {  
  
    /* corpo del for */  
  
}
```

Frammento di codice equivalente al precedente :

```
int counter;  
for(counter=0; counter < 10; counter++) {  
    printf("%d\n", counter);  
}
```

Loops in C: for

Loop con inizializzazione ed incremento:

```
for ( /* inizializzazione */ ; /* test */ ; /*  
incremento */ ) {  
  
    /* corpo del for */  
  
}
```

Può essere riscritto come :

```
inizializzazione;  
while (test)  
{  
    corpo;  
    incremento;  
}
```

Array

Tipo di dato (o struttura dati) di estrema importanza nella programmazione imperativa

Ad alto livello, un array è una *collezione di oggetti dello stesso tipo*, ciascuno identificato da un *indice* intero 0,1,2,..., n-1 (n: dimensione dell' array)

- Una trasposizione del concetto di vettore o matrice usato in matematica

Nella macchina, un array è una sequenza di locazioni di memoria adiacenti, contenenti le rappresentazioni degli elementi dell'array in sequenza

indice	elemento
0	150
1	80
2	400
3	13
4	1520

- Gli array permettono di leggere/modificare il valore di un elemento dato il suo indice

Array in C

La sintassi per dichiarare array di **dimensione costante** in C è la seguente:

```
tipo nome-array [ dimensione ]
```

```
Esempi: int a[10];  
        char s[30];  
        int b[5]={0,1,2,3,4}; /* con inizializzazione */
```

Per riferire un elemento di un array si specifica il suo indice tra parentesi quadre preceduto dal nome dell'array:

```
a[0] = 20;    x= a[5] + y;
```

```
a[i]++;      /* l'indice può essere il risultato di un espressione, come la  
              variabile i in questo caso */
```

Array in C

Per scandire un array si usa tipicamente un ciclo for

```
/* inizializzare un array con interi pseudo-casuali usando rand() */
#include <stdlib.h>
#include <time.h>

...
srand(time(NULL)); /* inizializza generatore pseudo-casuale */
int a[10];
int i;

for (i=0; i<10; i++) { /* l'indice deve essere in [0,9]*/
    /* fai qualcosa con a[i] */

    a[i] = rand()% 100;
}
...
```

Array in C

Per scandire un array si usa tipicamente un ciclo for

```
/* inizializzare un array con interi pseudo-casuali usando rand() */
#include <stdlib.h>
#include <time.h>

...
srand(time(NULL)); /* inizializza generatore pseudo-casuale */
int a[10];
int i;

for (i=0; i<10; i++) { /* l'indice deve essere in [0,9]*/
    /* fai qualcosa con a[i] */

    a[i] = rand()% 100;
}
...
```

L'accesso "out-of-bound" (es. a[1000]) genera comportamenti indesiderati (NON fatelo :-)

Sommario su *printf* e *scanf*

printf (<stdio.h>) funzione di libreria per scrivere testo formattato sullo standard output

Uso:

```
printf("formato-output", lista-argomenti)
```

Rimpiazza in *formato-output* ogni place-holder con il corrispondente argomento e stampa il risultato:

```
printf("Il valore di x è %d", x)
```

Place-Holder per interi,
rimpiazzato da x in
output

Sommario su *printf* e *scanf*

Scanf è la funzione di libreria tipicamente usata per leggere dallo standard input

Sintassi simile alla *printf* ma comportamento simmetrico:

```
scanf("formato-input", lista-argomenti)
```

Ex.:

```
printf("Inserisci il valore di x");  
scanf("%d", &x);
```

Variabili precedute da un &
(passaggio per riferimento ...)

Esempio: printf/scanf

```
#include <stdio.h>
#define pi 3.1415f
```

```
int main(){
```

```
    float r; /* raggio del cerchio */
    float a; /* area */
```

```
    scanf("%f", &r);
    a = pi * r * r;
    printf("%f\n", a);
```

```
}
```

Esempio: printf/scanf

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

```
➔ #define pi 3.1415f
```

```
int main(){
```

```
    float r; /* raggio del cerchio */
```

```
    float a; /* area */
```

```
    scanf("%f", &r);
```

```
    a = pi * r * r;
```

```
    printf("%f\n", a);
```

```
}
```

Esempio: printf/scanf

```
#include <stdio.h>
#define pi 3.1415f
```

```
int main(){
```

```
➔ float r; /* raggio del cerchio */
   float a; /* area */
```

```
   scanf("%f", &r);
   a = pi * r * r;
   printf("%f\n", a);
}
```

Esempio: printf/scanf

```
#include <stdio.h>
#define pi 3.1415f
```

```
int main(){
```

```
    float r; /* raggio del cerchio */
    float a; /* area */
```

```
    scanf("%f", &r);
    a = pi * r * r;
    printf("%f\n", a);
```

```
}
```



Esempio: printf/scanf

```
#include <stdio.h>
#define pi 3.1415f
```

```
int main(){
```

```
float r; /* raggio */
float a; /* area
```

```
scanf("%f", &r);
a = pi * r * r;
printf("%f\n", a);
```

```
}
```

Place-holder per float `%f`. Indica che un valore float è atteso in input

Richiede l'indirizzo della variabile che conterrà l'input (operatore `&`).
Esempio: `&r`

Esercizio 1

Laboratorio di Algoritmica
Università di Pisa

[admin](#) [Logout](#)

Menu

Algoritmica e Laboratorio

- [Edit](#)
- [Lessons](#)
- [\[Lezione 1\] Ripasso C](#)
- [News](#)
- [Status](#)
- [Ranking](#)

- [Courses](#)
- [Problems](#)
- [Config](#)
- [Tutors](#)

Ciao Mondo

Submitted solutions: 0

Ciao Mondo!

scrivere un programma che stampi "Ciao Mondo!" sul terminale

```
> ./a.out
```

```
Ciao Mondo!
```

Available languages C (gcc 4.3.2)

Esercizio 2

NumeroPrimo

Submitted solutions: 0

Scrivere un programma che legga da tastiera un intero X e stabilisca se il numero è primo.

Input

Un'unica riga contenente il valore X.

Output

Un'unica riga contenente il valore 1 se X è primo, il valore 0 altrimenti.

Example

Input:

10

Output:

0

Input:

13

Output:

1

InvertireArray

Submitted solutions: 0

Il programma deve leggere da input un array A di N interi e deve invertire A, ossia scambiare il contenuto della prima e dell'ultima cella, della seconda e della penultima, etc...

Si assuma che $N < 10000$.

Input

La prima riga contiene la dimensione N dell'array.

Le N righe successive contengono gli elementi dell'array.

Output

Gli elementi dell'array invertito. Un intero per riga.

Example

Input :

5

3

1

4

0

4

Output :

4

0

4

1

3