

Introduzione al C

Lez. 1 Elementi

Introduzione al C

Strumento che adotteremo in queste esercitazioni per implementare e testare gli algoritmi visti a lezione

Configurazione *minimale* suggerita:

- **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)

URL del corso:

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

Struttura di un programma C

Direttive per il pre-processor:

```
#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-processor:

```
#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-processor:

```
#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;
}
```



Commenti racchiusi tra /* e */

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.o hw.c
```

The diagram illustrates the components of the compilation command. The text '> gcc -o hw.o hw.c' is shown with red boxes around '-o hw.o' and 'hw.c'. Two black arrows point downwards from these boxes: one from '-o hw.o' to the word 'eseguibile' (executable) and one from 'hw.c' to the word 'sorgente' (source).

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.o hw.c
```

The diagram illustrates the components of the compilation command. Two arrows originate from the command: one points from the output specification '-o hw.o' to the word 'eseguibile' (executable), and another points from the source file 'hw.c' to the word 'sorgente' (source).

Risultato esecuzione:

```
> ./hw.o  
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 = x + y;
```

```
x = x*3;
```

```
x++; o ++x;
```

```
x += y;
```

```
x *= 3;
```

Esempio: dichiarazioni

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

```
int main(){
```

```
    int a=1;
```

```
    int b=2;
```

```
    printf("%d",a+b);
```

```
    return 0;
```

```
}
```

Tipi di base in C

Tutti i tipi primitivi del C sono *numerici* :

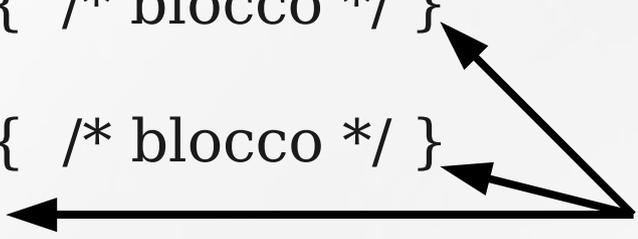
- Gli operatori aritmetici standard (+, -, *, /, %) sono definiti su di essi
- Interi: **char short int long**
 - **1 (char), 2 (short), 2 o 4 (int), 4 o 8 (long) bytes**
 - Il modificatore **unsigned** restringe a valori positivi
 - I *char* sono tipicamente usati per codificare caratteri ASCII:
'a',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

Branch in C: if-elseif-else

Costrutto condizionale di base in linguaggio C:

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

Opzionali



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

Un esempio con condizionale più sofisticato (con operatore %):

```
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 condizionale

Attenzione a non confondere:
= (assegnamento) con == (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 (stampa gli interi da 0 a 9):

```
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 cardinale importanza nella programmazione imperativa

Ad alto livello, un array è una *collezione di oggetti dello stesso tipo*, ciascuno identificato da un *indice* intero $0, 1, 2, \dots, n-1$ (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 */
```

N.B.: La numerazione degli indici in C è *0-based*. Esempio: se l'array *a* ha lunghezza 10 i suoi elementi sono `a[0]`, `a[1]`, `...`, `a[9]`

Array in C

Esempio:

```
/* 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" è possibile (ma non fatelo :-)

Array in C

Esempio:

```
/* 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" è possibile (ma 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)
```

```
printf("coordinate: %f, %f", a, b)
```

%d: interi (decimale)
%f: floating-point
%c: char

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-output", lista-argomenti)
```

Ex.:

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

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-output", 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("%3.2f\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("%3.2f\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("%3.2f\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("%3.2f\n", a);
}
```



Esempio: printf/scanf

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

```
int main(){
```

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

```
scanf("%f", &r);  
a = pi * r * r;  
printf("%3.2f\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

Esercizi

- 1) Scrivere un programma che prenda in input un intero n , e stampa "SI" se n è primo, "NO" altrimenti. (N.B.: un intero n è primo se è solo i suoi unici divisori interi sono 1 e n)
- 2) Scrivere un programma che esegua i seguenti 3 passi in sequenza:
 - Legga in input un intero n e inizializzi un'array A con n valori presi in input dall'utente.
 - *Inverta l'array A* , ossia scambi il contenuto della prima e dell'ultima cella, della seconda e della penultima, etc...
 - Stampi l'array invertito in output

Esempio:

Input: $n = 5$ e $A = \{3, 1, 4, 0, 0\}$

Output: 0 0 4 1 3

N.B.: assumere $n < 10000$