

Breve Riepilogo del Linguaggio C

Note

- Gli script shell cominciano “sempre” con la seguente riga:

```
#!/bin/sh
```

- NON esistono spazi all'interno della stringa “#!/bin/sh”
- NON ci sono altri caratteri (oltre il newline) sulla prima riga

Linguaggio C

- I sorgenti devono essere creati utilizzando un editor di testo (vi/emacs...)
 - Non utilizzare word processor!
- Per gcc, il suffisso del nome del file identifica la/le operazione/i che il compilatore esegue:
 - <file>.c: Codice sorgente C che deve essere preprocessato
 - <file>.C (maiuscola): Codice sorgente C++.
 - <file>.h: Header file che non deve essere ne' compilato ne' utilizzato per l'operazione di link

Contenuto dei file

- I file “.c” contengono:
 - direttive per il pre-processore
 - codice sorgente
- I file “.h” possono contenere:
 - direttive per il pre-processore
 - dichiarazioni di funzioni
 - dichiarazioni di variabili, strutture
- I file “.h” NON contengono l'implementazione delle funzioni

Modularita'

- E' opportuno distribuire le funzionalita' delle applicazioni complesse in piu' file.
 - Semplifica lo sviluppo, il debugging ed il riuso del codice.
- L'accorpamento delle funzionalita' avviene durante la compilazione:
 - NON utilizzare direttive del tipo:
 - #include "miofile.c"

Compilazione

- Nei sistemi Linux, la compilazione di sorgenti C avviene utilizzando il comando gcc (GNU C Compiler)
- Per compilare il programma test1.c potrebbe essere sufficiente eseguire:
 - gcc test1.c
 - Il nome dell'eseguibile, in questo caso, è a.out
- L'opzione -Wall mostra tutti i warning

Compilazione

- Per creare l'eseguibile, gcc attraversa varie fasi, tra cui:
 - preprocessing
 - compilazione
 - linking
- Se l'applicazione e' distribuita in piu' sorgenti, puo' essere sufficiente eseguire il seguente comando:
 - `gcc -o nomefile file1.c file2.c...`
 - In questo caso, il nome del programma eseguibile e' nomefile

Compilazione separata

- Normalmente, il compilatore esegue sia la compilazione che il linking
- **gcc -c file1.c** esegue solo la compilazione, producendo il file oggetto file1.o
- se abbiamo i file oggetto file1.o e file2.o, possiamo “linkarli” con **gcc file1.o file2.o**, producendo a.out
- il programma **make** automatizza questo processo

Linguaggio C

- In ogni applicazione esiste sempre un'unica funzione main
 - unica per l'applicazione
 - NON “unica in ogni file”.
- Le “firme” standard della funzione main sono:
 - `int main(int argc, char *argv[])`
 - Consente di ottenere i parametri passati all'applicazione dalla linea di comando
 - `int main(void)`

Puntatori

```
#include <stdio.h>
int main(void) {
    int a=5;
    int *p;
    printf("%d",a);
    p=&a;
    *p=6;
    printf("%d",a);
}
```

p=&a; (tutte le operazioni su “*p” sono operazioni su a)

p=a; ERRATO! p accetta indirizzi di interi, NON interi!

Operatori

- Operatori relazionali:

> >= < <= == !=

- Operatori logici

&& (and) || (or) ! (not)

- Operatori di incremento e decremento:

k++ ++k k-- --k

- Operatori orientati ai bit:

& (and) | (or) ^ (xor) << (shift a sx)

>> (shift a destra) ~ (complemento a uno)

Funzioni e prototipi

- E' opportuno **dichiarare** sempre le funzioni utilizzate all'interno di ogni file.
 - semplifica la correzione degli errori legati ai tipi
 - possibilmente tramite un file header “.h”
- La firma (o prototipo) di una funzione include:
 - tipo del valore restituito dalla funzione
 - nome della funzione
 - elenco dei parametri con rispettivi tipi

Funzioni e prototipi

in pippo.h:

```
int fun(int a, float b);  
...
```

in pippo.c:

```
#include "pippo.h"  
int fun(int a, float b){  
    return(a+(int)b);  
}
```

Esempio 1

Qual e' il comportamento del seguente codice C ?

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

int main(void)
{
    printf("Hello World!");
    sleep(5);
}
```

Esempio 1

Qual e' il comportamento del seguente codice C ?

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

int main(void)
{
    printf("Hello World!");
    sleep(5);
}
```

Una volta in esecuzione, aspetta 5 secondi e POI stampa "Hello World!"

Esempio 1

- Alcune funzioni di I/O standard utilizzano il *buffering*
 - utilizzano un'area di memoria per memorizzare le informazioni prima di inviarle al device
 - riduce il numero di system call “effettive” e migliora le performance
- Esistono tre tipi di buffering:
 - Completo: L'I/O effettivo avviene solo al riempimento del buffer
 - Non utilizzato per device interattivi come schermo e tastiera
 - Buffering a linea: L'I/O viene eseguito non appena viene inserito nel buffer un carattere newline '\n' (o al termine del processo)
 - Utilizzato da printf e scanf
 - Senza buffering: L'I/O avviene immediatamente
 - Utilizzato, ad esempio, per lo standard error

Esempio 1

Il seguente codice si comporterà, quindi, come atteso.

```
#include <stdio.h>
#include <unistd.h>
int main(void)
{
    printf("Hello World!\n");
    sleep(5);
}
```

Esempio 2

Qual e' l'output del seguente codice C ?

```
#include <stdio.h>
int main(void)
{
    int x,y;
    x=0;
    y=0;
    while(x=y) x=x+1;
    printf("x=%d, y=%d\n",x,y);
}
```

Esempio 2

Qual e' l'output del seguente codice C ?

```
#include <stdio.h>
int main(void)
{
    int x,y;
    x=0;
    y=0;
    while(x=y) x=x+1;
    printf("x=%d, y=%d\n",x,y);
}
```

Risposta: x=0, y=0

La condizione del while e' implementata utilizzando l'operatore di assegnazione "=" invece dell'operatore di confronto "=="

Esempio 3

Qual e' l'output del seguente codice C ?

```
#include <stdio.h>
int main(void)
{
    int x,y;
    x=0;
    for(y=0; y<99; y++);
    {
        x = x+1;
    }
    printf("x=%d, y=%d\n", x, y);
}
```

Esempio 3

Qual e' l'output del seguente codice C ?

```
#include <stdio.h>
int main(void)
{
    int x,y;
    x=0;
    for(y=0; y<99; y++);
    {
        x = x+1;
    }
    printf("x=%d, y=%d\n", x, y);
}
```

Risposta: x=1, y=99

Il ";" dopo il "for" rende il corpo del ciclo "vuoto".

Esempio 4: scanf

Il seguente programma compilato termina con errore. Perché?

```
#include <stdio.h>
int main(void)
{
    int x;
    printf("Inserisci un intero:");
    scanf("%d", x);
    printf("L'intero inserito è %d\n", x);
}
```

L'errore generato sui sistemi Unix è Segmentation Fault o, tentativo di accesso ad un "Segmento" di memoria non allocato.

Questo tipo di errore è dovuto, in genere, all'uso improprio dei puntatori.

Esempio 4: scanf

Il seguente programma compilato termina con errore. Perché?

```
#include <stdio.h>
int main(void)
{
    int x;
    printf("Inserisci un intero:");
    scanf("%d", &x);
    printf("L'intero inserito è %d\n", x);
}
```

Esempio 5: scanf

Qual e' il comportamento del seguente programma?

```
#include <stdio.h>
int main(void)
{
    int x;
    char stringa[31];
    printf("Inserisci un intero:");
    scanf("%d", &x);
    printf("Inserisci una riga di testo:");
    fgets(stringa, 30, stdin);
}
```

Esempio 5: scanf

Qual e' il comportamento del seguente programma?

```
#include <stdio.h>
int main(void)
{
    int x;
    char stringa[31];
    printf("Inserisci un intero:");
    scanf("%d\n", &x);
    printf("Inserisci una riga di testo:");
    fgets(stringa, 30, stdin);
}
```

Legge l'intero e termina.

La funzione "scanf" legge dal buffer solo l'intero e vi lascia il "\n".

Esempio 6: array

Qual e' l'errore in questo codice ?

```
#include <stdio.h>

int main(void)
{
    int a[10], i;
    for(i=1; i<=10; i++)
        a[i] = i;
    printf("a[%d]=%d\n", i, a[i]);
}
```

Esempio 6: array

Qual e' l'errore in questo codice ?

```
#include <stdio.h>

int main(void)
{
    int a[10], i;
    for(i=0; i<=9; i++)
        a[i] = i;
    printf("a[%d]=%d\n", i, a[i]);
}
```

Gli elementi in un array a con 10 elementi sono:
a[0], a[1], ... a[9]

Esempio 7

Qual e' l'errore in questo codice ?

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
    char *stringa;
    strcpy(stringa, argv[1]);
    printf("%s", stringa);
    return 0;
}
```

Esempio 7

Qual e' l'errore in questo codice ?

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
    char stringa[30];
    strcpy(stringa, argv[1]);
    printf("%s", stringa);
    return 0;
}
```

La dichiarazione `char *stringa` alloca lo spazio per un puntatore ad un carattere, NON per una stringa (array) di caratteri.

Anche così, il programma non è molto corretto...

Esempio 8

Perche' la funzione non esegue lo swap ?

```
#include <stdio.h>
int main(void)
{
    int x,y;
    void swap(int a, int b);
    printf("Inserisci l'intero x=");
    scanf("%d", &x);
    printf("Inserisci l'intero y=");
    scanf("%d", &y);
    swap(x,y);
    printf("(y,x)=(%d,%d)\n",x,y);
    return 0;
}
void swap(int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
    return
}
```

Esempio 8

```
#include <stdio.h>
int main(void)
{
    int x,y;
    void swap(int *a, int *b);
    printf("Inserisci l'intero x=");
    scanf("%d", &x);
    printf("Inserisci l'intero y=");
    scanf("%d", &y);
    swap(&x,&y);
    printf("(y,x)=(%d,%d)\n",x,y);
    return 0;
}

void swap(int *a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
    return
}
}
```

Perche' la funzione non esegue lo swap ?

Le variabili passate per valore alla funzione swap, sebbene modificate, mantengono il loro varole originale.

Esempio 9: cast

```
#include <stdio.h>

int main(void)
{
    float x = 3.5f;
    int i = (int) x;
    int *p = (int *) &x;
    printf("%d %d\n", i, *p);
    return 0;
}
```

- Qual e' l'output di questo programma?

Esempio 9: cast

```
#include <stdio.h>

int main(void)
{
    float x = 3.5f;
    int i = (int) x;
    int *p = (int *) &x;
    printf("%d %d\n", i, *p);
    return 0;
}
```

- output:
3 1080033280
- il primo cast effettua il troncamento di 3.5
- il secondo cast permette di leggere la rappresentazione binaria di 3.5 come se fosse un intero

Dichiarazioni

```
int a,b,c;  
float d=0.0,e,f=1.0;  
char g='a';  
struct{  
    int data;  
    float ratio;  
    char iniziale;  
} struttura;  
struttura.data=5;  
struttura.ratio=1.0;  
int array1[10];  
int array2[10][10];
```

Costanti simboliche:

```
#define MINIMO 1
```

```
#define MASSIMO 5
```

Istruzioni e Blocchi

- Il terminatore o delimitatore di istruzioni e' il “;”
 - Oppure la “,” ma con un significato specifico
- Le parentesi { } delimitano un blocco di istruzioni
 - E' necessario delimitare i blocchi contenenti almeno due istruzioni
 - E' opzionale delimitare i blocchi contenenti un'unica istruzione (es. “if”)

Variabili

- Tutte le variabili devono essere dichiarate prima del loro uso
- La variabile e' visibile solo all'interno del blocco in cui e' dichiarata
- Il C fornisce i seguenti tipo di dato base:
 - char, int, float, double, long, short.
 - NON esiste il tipo “stringa”
- E' possibile costruire tipi di dato complessi tramite il costrutto struct
- E' possibile dichiarare puntatori a variabili³⁶

Funzioni

- Il passaggio dei parametri puo' avvenire:
 - Per valore: eventuali modifiche all'interno della funzione NON hanno effetto al suo esterno
 - es., `int fun(int a);`
 - Per riferimento: Eventuali modifiche all'interno della funzione hanno effetto sul valore della variabile nella funzione chiamante
 - es. `int fun(int *a);`

Strutture di controllo

```
If (condizione){  
    lista istruzioni  
}
```

```
else{  
    lista istruzioni  
}
```

```
while (condizione){  
    lista istruzioni  
}
```

```
for (expr1;expr2;expr3){  
    lista istruzioni  
}
```

equivalente a:
expr1;

```
while (expr2){  
    lista istruzioni  
    expr3;  
}
```