

LabSO 2022

Laboratorio Sistemi Operativi - A.A. 2021-2022

dr. Andrea Naimoli	Informatica LT andrea.naimoli@unitn.it
dr. Michele Grisafi	Ingegneria informatica, delle comunicazioni ed elettronica (LT) michele.grisafi@unitn.it

C - introduzione

—

Perchè C?

- Struttura minimale
- Poche parole chiave (con i suoi pro e contro!)
- Unix compliant, alla base di Unix, nato per scrivere Unix
- Organizzato a passi, con sorgente, file intermedi ed eseguibile finale
- Disponibilità di librerie conosciute e standard
- Efficiente perché di basso livello
- Pieno controllo del programma e delle sue risorse
- Ottimo per interagire con il sistema operativo

Direttive e istruzioni fondamentali

- `#include ... / #define ... /`
- `char / int / ... / enum (v. esempio seguente)`
- `for (initialization ; test; increment) { ... ; }`
- `break / continue`
- `switch (expression){ case val: ... [break;] [default: ...] }`
- `while (expression) { ... } / do { ... } while (expression)`
- `if (expression) { ... } [else { ... }]`
- `struct / union`

(consultare una documentazione standard ed esercitarsi)

Tipi e Casting

C è un linguaggio debolmente tipizzato che utilizza 8 tipi fondamentali. È possibile fare il casting tra tipi differenti:

```
float a = 3.5;  
int b = (int)a;
```

La grandezza delle variabili è **dipendente dall'architettura di riferimento** i valori massimi per ogni tipo cambiano a seconda se la variabile è *signed* o *unsigned*.

- void (0 byte)
- char (1 byte)
- short (2 bytes)
- int (4 bytes)
- float (4 bytes)
- long (8 bytes)
- double (8 bytes)
- long double (8 bytes)

NB: non esiste il tipo boolean, ma viene spesso emulato con un char.

sizeof (*operatore*)

`sizeof (type) / sizeof expression`

Si tratta di un operatore che elabora il tipo passato come argomento (tra parentesi) o quello dell'espressione e restituisce il numero di bytes occupati in memoria.

```
#include <stdio.h>
void main() {
    int x = 10;
    printf("variable x      : %lu\n", sizeof x);
    printf("expression 1/2 : %lu\n", sizeof 1/2);
    printf("int type       : %lu\n", sizeof(int));
    printf("char type      : %lu\n", sizeof(char));
    printf("float type      : %lu\n", sizeof(float));
    printf("double type     : %lu\n", sizeof(double));
}
```

Puntatori di variabili

C si evolve attorno all'uso di puntatori, ovvero degli alias per zone di memorie condivise tra diverse variabili/funzioni. L'uso di puntatori è abilitato da due operatori: '*' ed '&'.

'*' ha significati diversi a seconda se usato in una dichiarazione o in un'assegnazione:

`int *punt;` → crea un puntatore ad intero

`int valore = *(punt);` → ottiene valore puntato

'&' ottiene l'indirizzo di memoria in cui è collocata una certa variabile.

`long whereIsValore = &valore;`

Esempi:

```
float    pie    =    3.4;
float    *pPie  =    &pie;
pie      *=      2;
float pie4 = *pPie * 2;

char    *array  =    "str";
*array      =      's';
array[1]    =      't';
*(array + 2) = 'r';
```

Puntatori di variabili

```
int    i    = 42;  
int *   punt = &i;  
int b = *(punt);
```

Tipo	Nome	Valore	Indirizzo
int	i	42	0xaaaabbbb
int *	punt	0xaaaabbbb	0xccccdddd
int	b	42	0x11112222

```
i = 20;
```



Nome	Valore
i	20
punt	0xaaaabbbb
b	?

Puntatori di funzioni

```
#include <stdio.h>
float xdiv(float a, float b) {
    return a/b;
}
float xmul(float a, float b) {
    return a*b;
}
int main() {
    float (*punt)(float, float);
    punt = xdiv;
    float res = punt(10,10);
    punt = &xmul; //& opzionale
    res = (*punt)(10,10);
    printf("%f\n", res);
    return 0;
}
```

C consente anche di creare dei puntatori a delle funzioni: puntatori che possono contenere l'indirizzo di funzioni differenti.

Sintassi simile ma diversa!

```
float (*punt)(float, float);
```

```
ret_type (* pntName)(argType, argType, ...)
```

main.c

- A parte casi particolari (es. sviluppo moduli per kernel) l'applicazione deve avere una funzione “**main**” che è utilizzata come punto di ingresso.
- Il valore di ritorno è **int**, un intero che rappresenta il codice di uscita dell'applicazione (variabile **\$?** in bash) ed è 0 di default se omissso. Può essere usato anche void, ma non è standard.
- Quando la funzione è invocata riceve normalmente in input il numero di argomenti (**int argc**), con incluso il nome dell'eseguibile, e la lista degli argomenti come “vettore di stringhe” (**char * argv[]**) (*)

(*) in C una stringa è in effetti un vettore di caratteri, quindi un vettore di stringhe è un vettore di vettori di caratteri, inoltre i vettori in C sono sostanzialmente puntatori (al primo elemento del vettore) → lista di argomenti spesso indicata con “**char ** argv**”

Utile riferimento generale: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html>

main.c

```
#include <stdlib.h>
#include <stdio.h>
#define DIVIDENDO 3

int division(int var1, int var2, int *result){
    *result = var1/var2;
    return 0;
}

int main(int argc, char * argv[]){
    float var1 = atof(argv[1]);
    int result = 0;
    division((int)var1,DIVIDENDO,(int *)&result);
    printf("%d \n",result);
    return 0;
}
```

Esecuzione - esempio

Compilazione:

```
gcc main.c -o main
```

Esecuzione:

```
./main arg1 arg2
```

```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("%d\n", argc);
    printf("%s\n", argv[0]);
    return 0;
}
```

In output si ha “3” (numero argomenti incluso il file eseguito) e “./main” (primo degli argomenti).

In generale quindi `argc` è sempre maggiore di zero.

Direttive

Il compilatore, nella fase di preprocessing, elabora tutte le direttive presenti nel sorgente. Ogni direttiva viene introdotta con ‘#’ e può essere di vari tipi:

<code>#include <lib></code>	copia il contenuto del file <i>lib</i> (cercando nelle cartelle delle librerie) nel file corrente
<code>#include "lib"</code>	come sopra ma cerca prima nella cartella corrente
<code>#define VAR VAL</code>	crea una costante VAR con il contenuto VAL, e sostituisce ogni occorrenza di VAR con VAL
<code>#define MUL(A,B) A*B</code>	dichiara una funzione con parametri A e B. Queste funzioni hanno una sintassi limitata!
<code>#ifdef, #ifndef, #if, #else, #endif</code>	rende l’inclusione di parte di codice dipendente da una condizione.

Macro possono essere passate a GCC con `-D NAME=VALUE`

Librerie standard

Librerie possono essere usate attraverso la direttiva `#include`.

Tra le più importanti vi sono:

- `stdio.h`: `FILE`, `EOF`, `stderr`, `stdin`, `stdout`, `fclose()`, etc...
- `stdlib.h`: `atof()`, `atoi()`, `malloc()`, `free()`, `exit()`, `system()`, `rand()`, etc...
- `string.h`: `memset()`, `memcpy()`, `strncat()`, `strcmp()`, `strlen()`, etc...
- `math.h`: `sin()`, `cos()`, `sqrt()`, `floor()`, etc...
- `unistd.h`: `STDOUT_FILENO`, `read()`, `write()`, `fork()`, `pipe()`, etc...
- `fcntl.h`: `creat()`, `open()`, etc...

...e ce ne sono molte altre.

Direttive - esempi

```
#include <stdio.h>
#define ITER 5
#define POW(A) A*A

int main(int argc, char **argv) {
#ifdef DEBUG
    printf("%d\n", argc);
    printf("%s\n", argv[0]);
#endif
    int res = 1;
    for (int i = 0; i < ITER; i++){
        res *= POW(argc);
    }
    return res;
}
```

```
gcc main.c -o main.out -D DEBUG=0
```

```
gcc main.c -o main.out -D DEBUG=1
```

```
./main.out 1 2 3 4
```

Stesso risultato!

Per un risultato “diverso”:

```
gcc main.c -o main.out
```

Structs e Unions

Structs permettono di aggregare diverse variabili, mentre le unions permettono di creare dei tipi generici che possono ospitare uno di vari tipi specificati.

```
#include <string.h>

struct Books{
    char author[50];
    char title[50];
    int bookID;
} book1, book2;

struct Books book3 =
{"Rowling", "Harry Potter", 2};
strcpy(book1.title, "Moby Dick");
book2.bookID = 3;
```

```
union Result{
    int intero;
    float decimale;
} result1, result2;

union Result result3;
result3.intero = 22;
result3.decimale = 11.5;
```


Typedef

Typedef consente la definizione di nuovi tipi di variabili o funzioni.

```
typedef unsigned int intero;  
  
typedef struct Books{  
    ...  
} bookType;  
  
intero var = 22; // = unsigned int var = 22;  
bookType book1; // = struct Books book1;
```

C - esempio “enum”

```
#include <stdio.h>

enum State {Undef = 9, Working = 1, Failed = 0};
void main() {
    enum State state=Undef;
    printf("%d\n", state); // output è “9”
}
```

C - vettori e stringhe

—

C - vettori I

I vettori sono sequenze di elementi omogenei (tipicamente liste di dati dello stesso tipo, ad esempio liste di interi o di caratteri).

I vettori si realizzano con un puntatore al primo elemento della lista.

Ad esempio con `int arr[4] = {2, 0, 2, 1}` si dichiara un vettore di 4 interi inizializzandolo: sono riservate 4 aree di memoria consecutive di dimensione pari a quella richiesta per ogni singolo intero (tipicamente 2 bytes, quindi $4*2=8$ in tutto)

C - vettori II

`char str[7] = {'c', 'i', 'a', 'o', 56, 57, 0} : 7*1 = 7 bytes`

`str` è dunque un puntatore a `char` (al primo elemento) e si ha che:

`str[n]` corrisponde a `*(str+n)`

e in particolare `str[0]` corrisponde a `*(str+0)=*(str)=*str`

C - stringhe

Le stringhe in C sono vettori di caratteri, ossia puntatori a sequenze di bytes, la cui terminazione è definita dal valore convenzionale 0 (zero).

Un carattere tra **apici singoli** equivale all'intero del codice corrispondente.

In particolare un vettore di stringhe è un vettore di vettore di caratteri e dunque:

```
char c;           #carattere
char *str;        #vettore di caratteri / stringa
char **strarr;    #vettore di vettore di caratteri / vettore di stringhe
```

Si comprende quindi la segnatura della funzione main con ****argv**.

Array e stringhe

C supporta l'uso di stringhe che, tuttavia, corrispondono a degli array di caratteri.

```
int nome[DIM];  
long nome[] = {1,2,3,4};  
char string[] = "ciao";  
char string2[] = {'c','i','a','o'};  
nome[0] = 22;
```

Gli array sono generalmente di dimensione statica e non possono essere ingranditi durante l'esecuzione del programma. Per array dinamici dovranno essere usati costrutti particolari (come malloc).

Le stringhe, quando acquisite in input o dichiarate con la sintassi "stringa", terminano con il carattere `'\0'` e sono dunque di grandezza str_len+1

C - esempio carattere e argc/argv

```
#include <stdio.h>

int main(int argc, char **argv) {
    int code=0;
    if (argc<2) {
        printf("Usage: %s <carattere>\n", argv[0]);
        code=2;
    } else {
        printf("%c == %d\n", argv[1][0], argv[1][0]);
    };
    return code;
};
```


C - argomenti da CLI

- Per il parsing degli argomenti da CLI la libreria `getopt.h` mette a disposizione `getopt` e `getopt_long`.

(v. https://www.gnu.org/software/libc/manual/html_node/Getopt.html)

- Si può effettuare un parsing manuale scorrendo gli argomenti.

C - parsing manuale argomenti: esempio

```
#define MAXOPTL 64
#define MAXOPTS 10
#include <stdio.h>
#include <string.h>
// arrays of options and of values
char opt[MAXOPTS][MAXOPTL];
char val[MAXOPTS][MAXOPTL];

int main(int argc, char **argv) {
    int a=0, o=0;
    // loop into arguments:
    while (++a<argc && o<MAXOPTS) {
        if (strcmp("-h", argv[a])==0)
            strcpy(opt[o++], "help");
```

```
...

        if (strcmp("-k", argv[a])==0) {
            strcpy(opt[o++], "key");
            if (a+1<argc)
                strcpy(val[o-1], argv[++a]);
        }
    }
    // dump options (keys/values):
    for (a=0; a<o; a++) {
        printf("opt[%d]: %s,%s\n",a,opt[a],val[a]);
    }
    return 0;
}
```

C - funzioni stringhe <string.h>

Dato che le stringhe sono riferite con un puntatore al primo carattere non ha senso fare assegnamenti e confronti diretti, ma si devono usare delle funzioni. La libreria standard string.h ne definisce alcune come ad esempio:

`char * strcat(char *dest, const char *src)` aggiunge src in coda a dest
`char * strchr(const char *str, int c)` cerca la prima occorrenza di c in str
`int strcmp(const char *str1, const char *str2)` confronta str1 con str2
`size_t strlen(const char *str)` calcola la lunghezza di str
`char * strcpy(char *dest, const char *src)` copia la stringa src in dst
`char * strncpy(char *dest, const char *src, size_t n)` copia n caratteri dalla stringa src in dst

C - esercizi per casa

1. Scrivere un'applicazione che data una stringa come argomento ne stampa a video la lunghezza, ad esempio:
`./lengthof "Questa frase ha 28 caratteri"`
deve restituire a video il numero 28.
2. Scrivere un'applicazione che definisce una lista di argomenti validi e legge quelli passati alla chiamata verificandoli e memorizzando le opzioni corrette, restituendo un errore in caso di un'opzione non valida.
3. Realizzare funzioni per stringhe `char *stringrev(char str)` (inverte ordine caratteri) e `int stringpos(char str, char chr)` (cerca *chr* in *str* e restituisce la posizione)

(In tutti i casi si può completare l'esercizio gestendo gli eventuali errori di immissione da parte dell'utente come parametri errati o altro)