2021_12_06

December 8, 2021

1 2021-12-06

1.0.1 Corso ITS

1.1 Magento & e-commerce software

1.2 ### Fondamenti di Programmazione (Andrea Ribuoli)

1.3 SECONDO ESERCIZIO

Creare una sotto-cartella di nome **math_api**. Crearvi un file sorgente C di nome **quiz.c**. Secondo quanto presentato nel corso creare anche un file *header* avente nome **quiz.h**. Nella cartella radice del progetto creare un file sorgente C di nome **test_driver.c** con una funzione **main()** che accetti due numeri interi dalla linea comando e prepari la chiamata alla funzione **quiz()** passando tali valori e ricevendo il risultato. La funzione main() provveda a stampare una riga di testo del tipo `quiz(22, 121) = r` nella ipotesi che 22 e 121 siano i parametri con cui il programma è stato invocato ed `r` il risultato restituito dalla funzione *quiz()*.

Idealmente realizzare due Makefile uno interno alla cartella *math_api* uno esterno e implementando una shared library di nome **libmath.so**.

```
int quiz(int x, int y) {
    int r;
    if (x < 0) x = -x;
    if (y < 0) y = -y;
    while (y) {
        r = x % y;
        x = y;
        Y = r;
    }
    return x;
}
```

1.4 PRIMA

```
|-- Makefile
|-- test_driver.c
|
|-- math_api
|   |-- Makefile
|   |-- quiz.c
```

```
|   |-- quiz.h
```

1.5 DOPO

```
|-- Makefile
|-- test_driver.c
|-- test_driver.o
|-- test_driver
|
|-- math_api
|   |-- Makefile
|   |-- quiz.c
|   |-- quiz.h
|   |-- quiz.o
|   |-- libmath.so
```

1.6 I "tranelli" del C (Leendert Ammeraal)

1.6.1 a) Il segno di uguaglianza

- il singolo segno di ugualianza = indica **assegnamento**
- il doppio segno di ugualianza == indica **uguaglianza**

(il primo, se utilizzato nel test di una istruzione **if** ad esempio, non viene rilevato dal compilatore)

1.6.2 b) L'operatore &

```
int a[3];
. . .
scanf("%d %d %d", a, a+1, a+2);
```

`a`, `a+1` e `a+2` sono indirizzi a posizioni di memoria

```
int n;
. . .
scanf("%d", &n);
```

Se ometto il simbolo **&** davanti a `n` induco un errore grave.

1.6.3 c) I valori restituiti da funzioni

Se una funzione non viene dichiarata e si ignora la *warning*, il compilatore la considererà ritornare un tipo **int**. Questo è particolarmente grave se utilizzo (senza dichiararle) funzioni matematiche che restituiscono ad esempio un tipo *double*.

1.6.4 d) I tipi degli argomenti e i parametri formali

Usare sempre gli include delle funzioni di libreria. Solo così il compilatore può operare le giuste conversioni di tipi numerici passati come *literal*

1.7 La compilazione “separata”

Nelle lezioni precedenti abbiamo indicato come la modularizzazione del codice consente la compilazione separata e come l’uso del comando **make** riduce l’impatto operativo delle attività di coordinamento necessarie per identificare le sole ri-compilazioni strettamenete necessarie.

Questo è stato un pregio storico del linguaggio **C** specie negli anni in cui si confrontava con il linguaggio **Pascal** sicuramente più idoneo per la didattica ma con limitazioni.

Oltre alla mancanza di compilazione separata, il Pascal non offriva le operazioni a livello di bit (viste nelle lezioni precedenti) e neppure l’accesso casuale ai file.

[1]:
```
!git clone https://github.com/lucatinti13/EserciziC.git
```

```
Cloning into 'EserciziC'…
remote: Enumerating objects: 110, done.
remote: Counting objects: 100% (110/110), done.
remote: Compressing objects: 100% (105/105), done.
remote: Total 110 (delta 46), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (110/110), 434.05 KiB | 856.00 KiB/s, done.
Resolving deltas: 100% (46/46), done.
```

[1]:
```
!cd EserciziC/math_api; make
```

```
gcc -c -o quiz.o quiz.c
gcc -shared -o libmath.so quiz.o
```

1.8 test_driver.c

```c
#include "quiz.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
   int a, b;
   if (argc != 3) {
       printf("Usage: test_driver <int_a> <int_b>\n");
       exit(-1);
   }
   a = atoi(argv[1]);
   if (a == 0) {
       printf("First argument must be an integer\n");
       exit(-2);
   }
   b = atoi(argv[2]);
   if (b == 0) {
       printf("Second argument must be an integer\n");
       exit(-3);
   }
   printf("quiz(%i, %i) = %i\n", a, b, quiz(a,b));
```

```
   return 0;
}
```

1.9 quiz.c

```
#include "quiz.h"

int quiz(int x, int y) {
    int r;
    if (x < 0) x = -x;
    if (y < 0) y = -y;
    while (y) {
        r = x % y;
        x = y;
        y = r;
    }
    return x;
}
```

1.10 quiz.h

```
#ifndef __QUIZ_H__
#define __QUIZ_H__
int quiz(int x, int y);
#endif
```

1.11 Makefile (root)

```
all: sub-make test_driver

sub-make:
    make -C math_api

test_driver : test_driver.o
    gcc -o test_driver -L./math_api -lmath test_driver.o

test_driver.o : test_driver.c
    gcc -c -I./math_api -o test_driver.o test_driver.c
```

1.12 Makefile (math_api)

```
all : libmath.so

libmath.so : quiz.o
    gcc -shared -o libmath.so quiz.o

quiz.o : quiz.c quiz.h
    gcc -c -o quiz.o quiz.c
```

[1]:
```
!make
```

```
make -C math_api
make[1]: Entering directory '/home/jupyter/ITS_Marche/Fondamenti di
Programmazione/2021_12_06/math_api'
gcc -c -o quiz.o quiz.c
gcc -shared -o libmath.so quiz.o
make[1]: Leaving directory '/home/jupyter/ITS_Marche/Fondamenti di
Programmazione/2021_12_06/math_api'
gcc -c -I./math_api -o test_driver.o test_driver.c
gcc -o test_driver -L./math_api -lmath test_driver.o
ld: 0711-224 AVVERTENZA: Simbolo duplicato: .__init_aix_libgcc_cxa_atexit
ld: 0711-345 Usare l'opzione -bloadmap o -bnoquiet per ulteriori informazioni.
```

[2]:
```
!./test_driver 66 242
```

```
quiz(66, 242) = 22
```

[3]:
```
!./test_driver 125 200
```

```
quiz(125, 200) = 25
```