Variabili, tipi primitivi e costrutti condizionali

Fondamenti di Programmazione e Laboratorio

A.A. 2018/2019

Dichiarazioni di variabili

La sintassi da utilizzare per dichiarare una variabile la seguente:

```
tipo nome_variabile [=valore iniziale];
```

- tipo della variabile;
- nome della variabile: i nomi possono essere composti da lettere e cifre, ma devono sempre cominciare con una lettera. Ricordatevi le regole stilistiche viste la scorsa volta
- si può definire un valore iniziale della variabile. È fortemente consigliato inizializzare le variabili

Tipi di dato primitivi: int

`int`: un intero rappresentabile sulla macchina (segnaposto `%d`)

- dimensione e gamma di valori rappresentabile dipende dalla macchina su cui viene compilato il programma.

Tipi di dato primitivi: int

`int`: un intero rappresentabile sulla macchina (segnaposto `%d`)

- dimensione e gamma di valori rappresentabile dipende dalla macchina su cui viene compilato il programma.
- la funzione predefinita `sizeof()` fornisce la lunghezza in byte di un qualsiasi tipo o variabile C. Ad esempio se lanciamo sulle macchine del laboratorio:

```
#include <stdio.h>
int main()
{
    printf("%d\n",sizeof(int));
}
```

Tipi di dato primitivi: int

`int`: un intero rappresentabile sulla macchina (segnaposto `%d`)

- dimensione e gamma di valori rappresentabile dipende dalla macchina su cui viene compilato il programma.
- la funzione predefinita `sizeof()` fornisce la lunghezza in byte di un qualsiasi tipo o variabile C. Ad esempio se lanciamo sulle macchine del laboratorio:

```c
#include <stdio.h>
int main()
{
    printf("%d\n",sizeof(int));
}
```

Otteniamo in output:

```
4
```

Tipi di dato primitivi: int

`int`: un intero rappresentabile sulla macchina (segnaposto `%d`)

- dimensione e gamma di valori rappresentabile dipende dalla macchina su cui viene compilato il programma.
- la funzione predefinita `sizeof()` fornisce la lunghezza in byte di un qualsiasi tipo o variabile C. Ad esempio se lanciamo sulle macchine del laboratorio:

```c
#include <stdio.h>
int main()
{
    printf("%d\n",sizeof(int));
}
```

Otteniamo in output:

```
4
```

- gli attributi `short` e `long` vengono usati per denotare interi di lunghezze diverse.

Signed vs Unsigned

`signed` e `unsigned` sono utilizzati per numeri con e senza segno. Dalla documentazione ISO C:

- The int data type is signed and has a minimum range of at least -32767 through 32767 inclusive. The actual values are given in `limits.h` as `INT_MIN` and `INT_MAX` respectively.
- An unsigned int has a minimal range of 0 through 65535 inclusive with the actual maximum value being `UINT_MAX` from that same header file.

Tipi di dato primitivi: char

`char`: un singolo byte, in grado di contenere un carattere (codifica `ASCII`). Segnaposto `%c`.

```
char a='a'; //i caratteri si indicano tra apici
```

Tipi di dato primitivi: char

`char`: un singolo byte, in grado di contenere un carattere (codifica `ASCII`). Segnaposto `%c`.

```
char a='a'; //i caratteri si indicano tra apici
```

- i caratteri sono visti in C come interi. Infatti:

```
int a='a';
printf("%c\n",a);
printf("%d\n",a);
```

Stampa in output:

```
a
97
```

`97` corrisponde alla codifica `ASCII` di `a`

Tipi di dato primitivi: char (2)

- Alcune costanti carattere e i relativi valori interi:

Costante carattere	'a'	'b'	...	'z'
Valore intero	97	98	...	122
Costante carattere	'A'	'B'	...	'Z'
Valore intero	65	66	...	90
Costante carattere	'0'	'1'	...	'9'
Valore intero	48	49	...	57

Non c'è alcuna relazione tra una costante carattere cifra e la cifra stessa: '2' non è 2.

Tipi di dato primitivi: numeri reali

`float`, **`double`**: sono i tipi utilizzati per rappresentare numeri reali (precisione singola o doppia)

```
float x=123.34;
double y=100.1e5; //anche notazione scientifica
```

Tipi di dato primitivi: numeri reali

`float`, **`double`**: sono i tipi utilizzati per rappresentare numeri reali (precisione singola o doppia)

```
float x=123.34;
double y=100.1e5; //anche notazione scientifica
```

- segnaposto `%f` e `%lf`;
- dimensioni differenti: `double` viene tipicamente rappresentato con 8 byte, `float` con 4;

Dimensione tipi

È possibile utilizzare l'attributo `long`. Standard C:

- `int` almeno 16 bits
- `long` almeno 32 bits
- `long long` almeno 64 bits

Su tipiche architetture a 32-bit:

- `int` è 32 bits
- `long` è 32 bits
- `long long` è 64 bits

Su tipiche architetture a 64-bit:

- `int` è 32 bits
- `long` è o 32 o 64 bits
- `long long` è 64 bits

Costanti

L'attributo `const` può essere applicato alla dichiarazione di qualasiasi variabile, con l'effetto di prescrivere che il suo valore non cambierà.

```
const double pi=3.141592;
const int cinque=5;
```

I tentativi di modifica a costanti sono tipicamente segnalati dal compilatore con un errore.

Differenza tra `#define` e `const`????

Costanti

L'attributo `const` può essere applicato alla dichiarazione di qualasiasi variabile, con l'effetto di prescrivere che il suo valore non cambierà.

```
const double pi=3.141592;
const int cinque=5;
```

I tentativi di modifica a costanti sono tipicamente segnalati dal compilatore con un errore.

Differenza tra `#define` e `const`????

- `#define` è una direttiva del preprocessore e sostituita all'interno del codice prima della compilazione;
- una variabile definita con l'attributo `const` viene maneggiata dal compilatore: ha un tipo e un indirizzo.

Operatori aritmetici

Nel C abbiamo essenzialmente i seguenti operatori aritmetici:

$$+ \quad - \quad * \quad / \quad \%$$

che rappresentano rispettivamente le usuali operazioni di addizione, sottrazione, moltiplicazione, divisione e modulo. Vengono utilizzati per definire, trasformare il valore delle vostre variabili.

Ricordiamo che il valore di *a* modulo *b* è dato dal resto della divisione di *a* per *b*: ad esempio $5\%3 = 2$.
L'operatore modulo non può essere applicato a variabili `float` e `double`.

Operatori aritmetici(2)

Gli operatori possiedono regole di precedenze e associatività che determinano come avviene la valutazione delle espressioni.

Operatori aritmetici(2)

Gli operatori possiedono regole di precedenze e associatività che determinano come avviene la valutazione delle espressioni.

Così come avviene nell'aritmetica tradizionale, $+$ e $-$ hanno lo stesso grado di precedenza, inferiore a quello di $*$, $/$ e %.

Operatori aritmetici(3)

Altri operatori:

- operatori contratti: permettono di effettuare un'operazione su una variabile e assegnarne il risultato alla stessa.
  - l'espressione `var op= expr`
  - equivale a `var = var op expr`

Operatori aritmetici(3)

Altri operatori:

- operatori contratti: permettono di effettuare un'operazione su una variabile e assegnarne il risultato alla stessa.
  - l'espressione `var op= expr`
  - equivale a `var = var op expr`
  - ad esempio: `j*=i+2` $\Leftrightarrow$ `j=j*(i+2)`

Operatori aritmetici(3)

Altri operatori:

- operatori contratti: permettono di effettuare un'operazione su una variabile e assegnarne il risultato alla stessa.
  - l'espressione `var op= expr`
  - equivale a `var = var op expr`
  - ad esempio: `j*=i+2` ⇔ `j=j*(i+2)`
- incremento/decremento unitario: sono effettuati rispettivamente dagli operatori `++` e `--`. Sono utilizzabili sia come prefisso (ossia prima della variabile: `++n`) o suffisso (dopo la variabile: `n++`). L'effetto è sempre quello di incrementare `n`:
  - `++n` esegue l'incremento prima di usare il valore `n`;
  - `n++` lo fa dopo l'impiego del valore.

Conversioni di tipo

Le espressioni aritmetiche hanno un valore ed un tipo dettato da quello delle variabili in gioco.

Quando un operatore ha operandi di tipo diverso, essi sono convertiti nello stesso tipo applicando alcune regole automatiche. In genere quello che si fa è di ampliare il campo dei valori rappresentabili.

Esempio

Se `x` ha tipo `int` e `y` ha tipo `float`, il risultato dell'espressione `x+y` viene automaticamente convertito a `float`

Casting

Oltre alle conversioni implicite, sono possibili anche conversioni esplicite dette cast secondo la sintassi:

`(tipo)` espressione;

Esempio:

```
int somma, n;
float media;
...
media = somma/n; /* divisione tra interi */
media = (float)somma/n; /* divisione tra reali */
```

L'operatore di cast ha precedenza più alta degli operatori binari e associa da destra a sinistra. Dunque `(float)somma/n` equivale a `((float)somma)/n`

Booleani ed operatori

In C non esiste un tipo Booleano. Si usa il tipo `int`:

- `0` rappresenta FALSO;
- `diverso da 0` (tipicamente 1) rappresenta VERO.

Booleani ed operatori

In C non esiste un tipo Booleano. Si usa il tipo `int`:

- `0` rappresenta FALSO;
- `diverso da 0` (tipicamente 1) rappresenta VERO.

Operatori logici:

- `!`: NOT (operatore unario). Esempio: `!a`;
- `&&`: AND (operatore binario). Esempio: `a && b`;
- `||`: OR (operatore binario). Esempio: `a || b`;

Restituiscono un valore intero pari a 0 o 1 a seconda del valore (falso/vero) dell'espressione.
Altri lavorano sui singoli bit: ad esempio operatori di shift («, »), AND (`&`), OR (`|`), XOR (`^`) ...

Operatori relazionali e di uguaglianza

Gli operatori relazionali sono:

`<` `>` `<=` `>=`

sono tutti binari: hanno come operandi due espressioni e restituiscono un risultato di tipo `int` che può essere `0` o `1`.

Ad esempio l'espressione relazione `a<b`:

- se `a` è minore di `b` assume valore `1` (vero);
- altrimenti assume valore `0` (falso).

Operatori relazionali e di uguaglianza

Gli operatori relazionali sono:

$$< \quad > \quad <= \quad >=$$

sono tutti binari: hanno come operandi due espressioni e restituiscono un risultato di tipo `int` che può essere `0` o `1`.

Ad esempio l'espressione relazione `a<b`:

- se `a` è minore di `b` assume valore `1` (vero);
- altrimenti assume valore `0` (falso).

Abbiamo poi gli operatori di ugualianza $==$ e $!=$, sempre binari, che assumono valore `1` (vero) se i due operandi sono rispettivamente uguali o diversi. `0` in caso contrario.

La libreria `stdio.h`

Nel C le funzionalità di input/output sono demandate a librerie esterne. Ad esempio la libreria `stdio.h` implementa un semplice modello di input/output di dati testuali.

Le funzioni `printf` e `scanf` sono in essa definite: la prima viene utilizzata per l'output, la seconda per l'input. La lettera `f` alla fine dei nomi delle due funzioni sta per "formatted".

Sia `printf()` che `scanf()` ricevono una stringa di controllo, che può contenere le specifiche di conversione indicate con il simbolo % (segnaposto), e una serie di parametri, che possono essere ad esempio le variabili da stampare o leggere.

printf

Per stampare delle variabili di un determinato tipo dobbiamo utilizzare i segnaposto relativi:

- interi: `%d`, `%u` (unsigned). Si antepone `h` per short e `l` per long;
- reali: `%f` (float), `%e` (notazione scientifica), `%g` la piu' breve tra notazione standard e scientifica. Per i double `%l` o `%lf` (a seconda dello standard, il secondo in C99), per i long double `Lf`;
- caratteri: `%c`;
- stringhe: `%s` (le vedremo piu' avanti).

printf(2)

Altri flag, vanno messi subito dopo il %:

- `-`: allinea a sinistra
- un numero intero `n`: specifica l'ampiezza minima del campo. Esempio:

```
printf("%c%3c%5c\n",'A','B','C');
```

  Stampa: `A B    C`

- un parametro `m.d`: per i numeri reali specifica l'ampiezza (minima) del campo `m` e il numero di cifre decimali `d`. Esempio:

```
printf("%2.3f\n",123.4557454);
```

  Stampa: `123.456`

printf(3)

Infine abbiamo i caratteri di escape, dei caratteri speciali che iniziano per \:

- \n: va a capo;
- \t: inserisce un tab;
- \%: inserisce il simbolo %;
- ....

NOTA: come per i comandi della shell, anche per le funzioni di libreria standard possiamo usare il man:

```
man 3 <nome_funzione>
```

scanf

La funzione `scanf` è analoga alla `printf` ma viene utilizzata per la lettura. Anche in questo caso abbiamo una stringa di controllo, mentre gli altri parametri sono gli indirizzi delle variabili in cui saranno memorizzati i valori letti.

Ad esempio:

```
scanf("%d",&x);
```

legge un intero e lo memorizza il valore all'indirizzo di `x`, indicato con la notazione `&x`.

Gli specificatori di formato sono quelli visti con la `printf`, tranne che per i reali:

- `double`: si antepone `l`;
- `long double`: si antepone `L`.

Comando condizionale

Le istruzioni di un programma vengono normalmente eseguite in sequenza, ma la maggior parte dei programmi richiede una modifica al normale flusso sequenziale del controllo.

Le istruzioni `if` e `if-else` ci permettono ad esempio di scegliere tra azioni alternative. Sintassi:

```
if (espressione){
      blocco1
}
else{
      blocco2
}
```

`espressione` è un' espressione booleana. Se vera vengono eseguite le istruzioni in `blocco1` altrimenti vengono eseguite le istruzioni in `blocco2`. Il ramo `else` è opzionale.

Comando condizionale(2)

Esempio: si acquisisca l'età da tastiera e si stabilisca se l'utente sia maggiorenne o meno.

```
#include <stdio.h>
int main(void)
{
      int eta;
      printf("Inserisci la tua et\'a in anni: ");
      scanf("%d",&eta);
      if(eta>=18)
          printf("Sei maggiorenne\n");
      else
          printf("Sei minorenne\n");
      return 0;
}
```

NOTE:

- ricordate sempre l'indentazione;
- se i rami dell'`if` sono composti da una sola istruzione, potete non inserire le parentesi graffe.

Switch

L'istruzione `switch` può essere usata per realizzare una selezione a più vie.

Switch

L'istruzione `switch` può essere usata per realizzare una selezione a più vie.

Sintassi:

```
switch (espressione) {
      case valore_1: istruzioni_1
           break;
      ...
      case valore_n: istruzioni_n
           break;
      default : istruzioni_default
}
```

Switch(2)

Semantica:

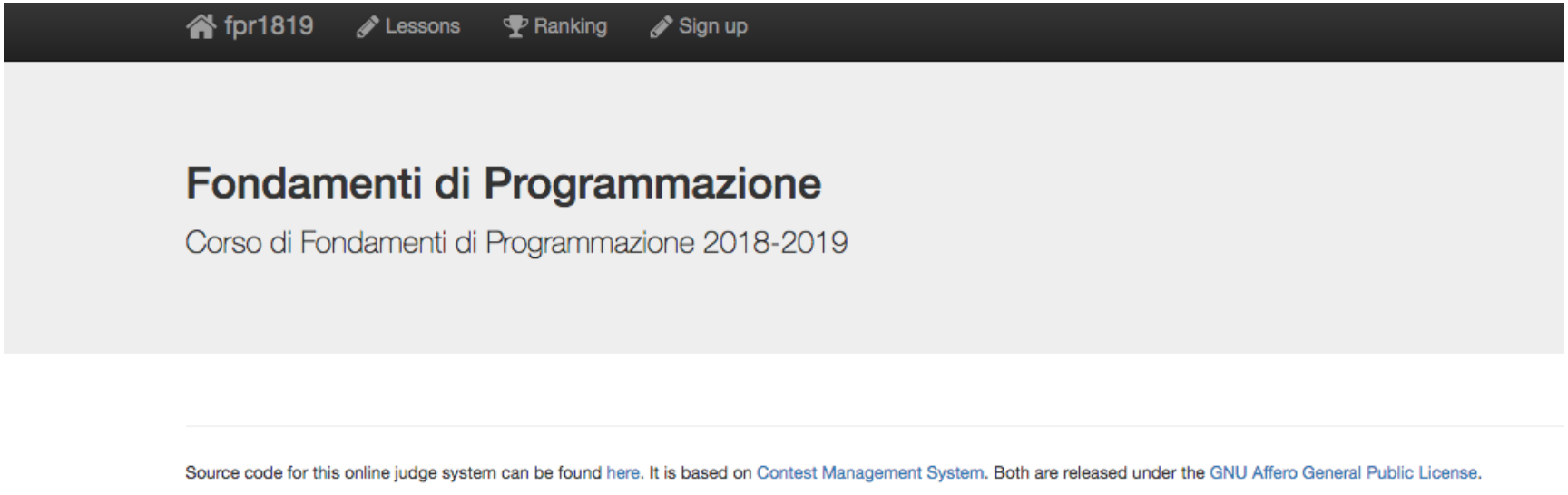
1. viene valutata `espressione` (deve restituire un intero) e confrontata con i valori contenuti nelle varie etichette (che devono essere costanti);
2. quando viene raggiunta un'etichetta uguale all'espressione esegue tutti i comandi che seguono (per interrompere tale esecuzione si usa il comando `break`);
3. se nessuna etichetta è uguale al valore dell'espressione si esegue il ramo `default` se definito (è opzionale).

Switch(3)

Esempio:

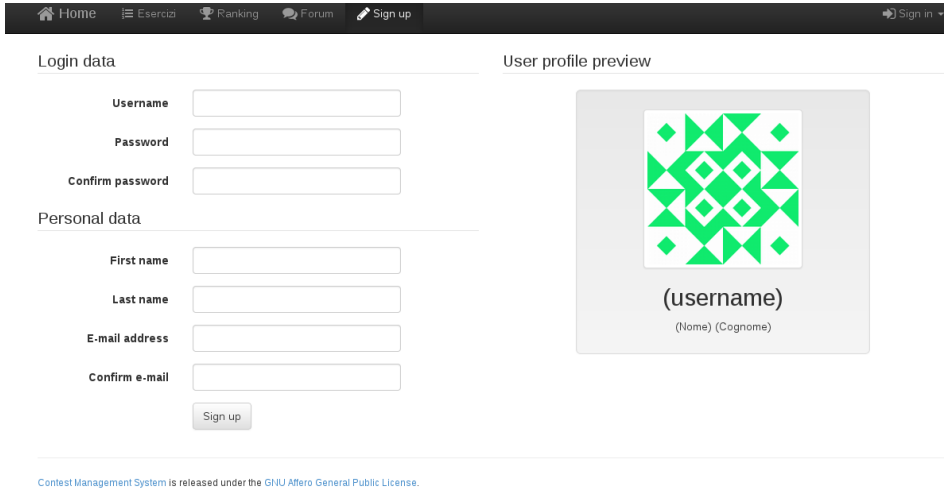
```
int giorno;
...
switch (giorno) {
      case 1: printf("Lunedi\n");
              break;
      case 3: printf("Martedi\n");
              break;
      case 3: printf("Mercoledi\n");
              break;
      case 4: printf("Giovedi\n");
              break;
      case 5: printf("Venerdi\n");
              break;
      default: printf("Week end\n");
}
```

Piattaforma autovalutazione



`http://fpr1819.dijkstra.di.unipi.it/`

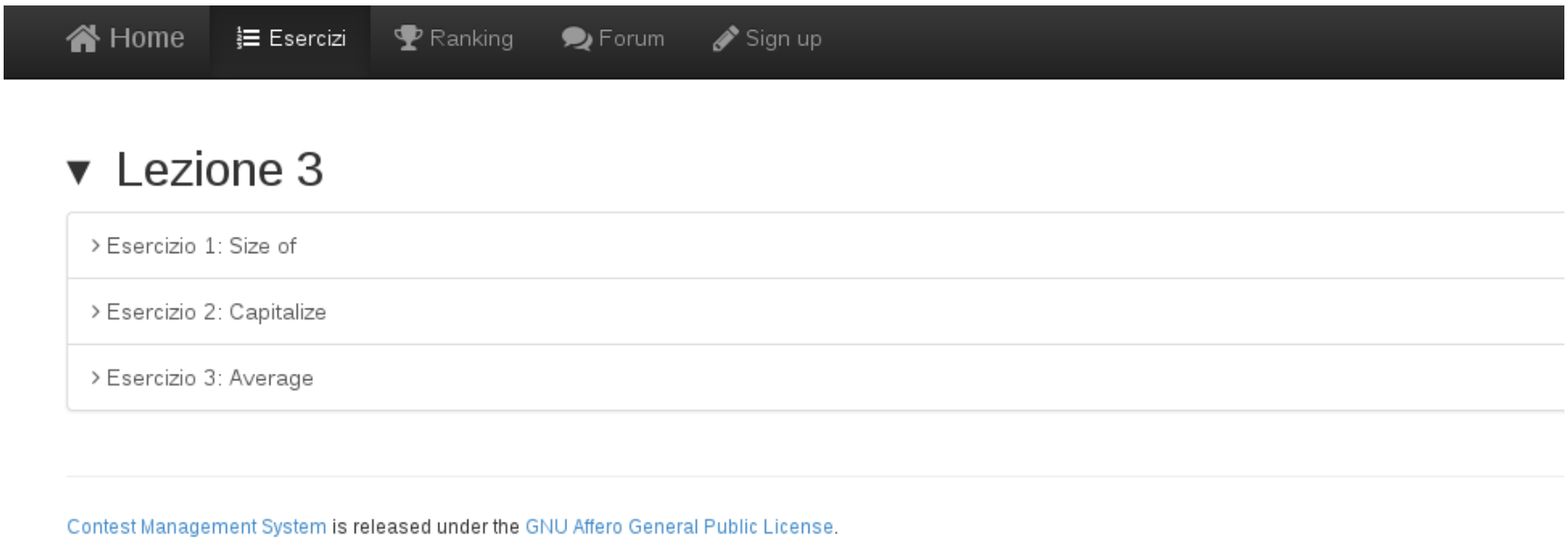
Registrazione



- inserite i vostri dati;
- usate un Username nella forma "cognome.nome.corso"
  ad esempio: `pisanti.nadia.FPL18`

C'è un limite sul numero di caratteri possibili per lo Username, quindi, se necessario, usate solo la prima lettera del vostro nome.

Esercitazioni

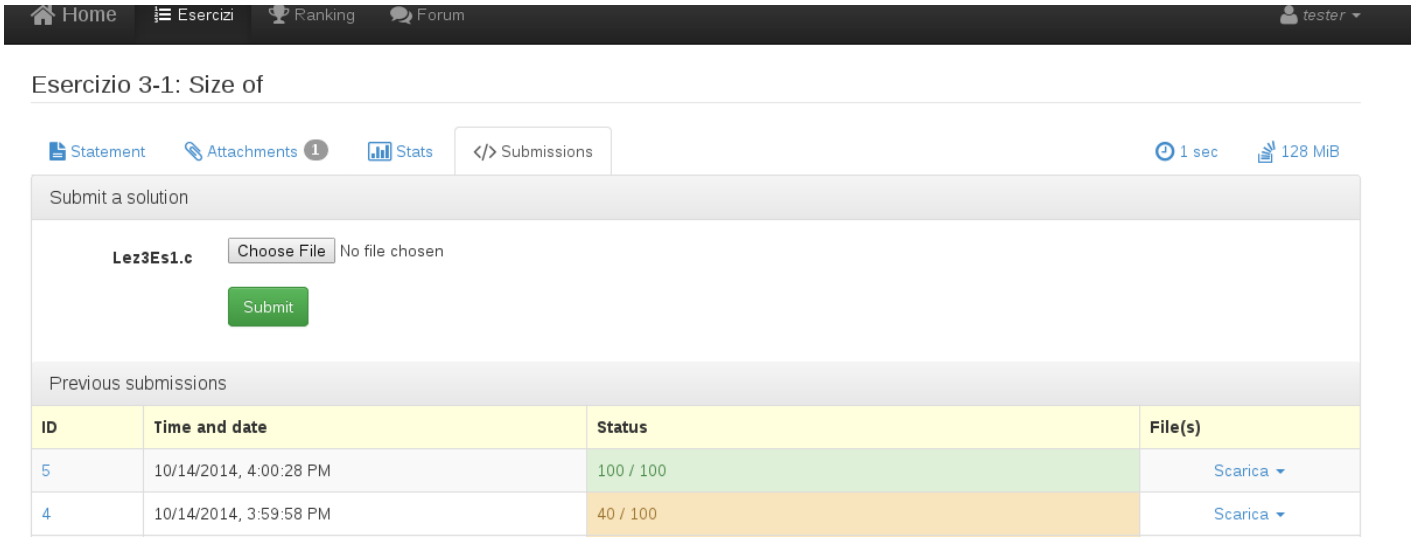


Le varie esercitazioni sono organizzate per lezione (a partire dalla 3)

Statement

Per ogni esercizio trovate sotto statement il testo (con alcuni esempi di input/output)

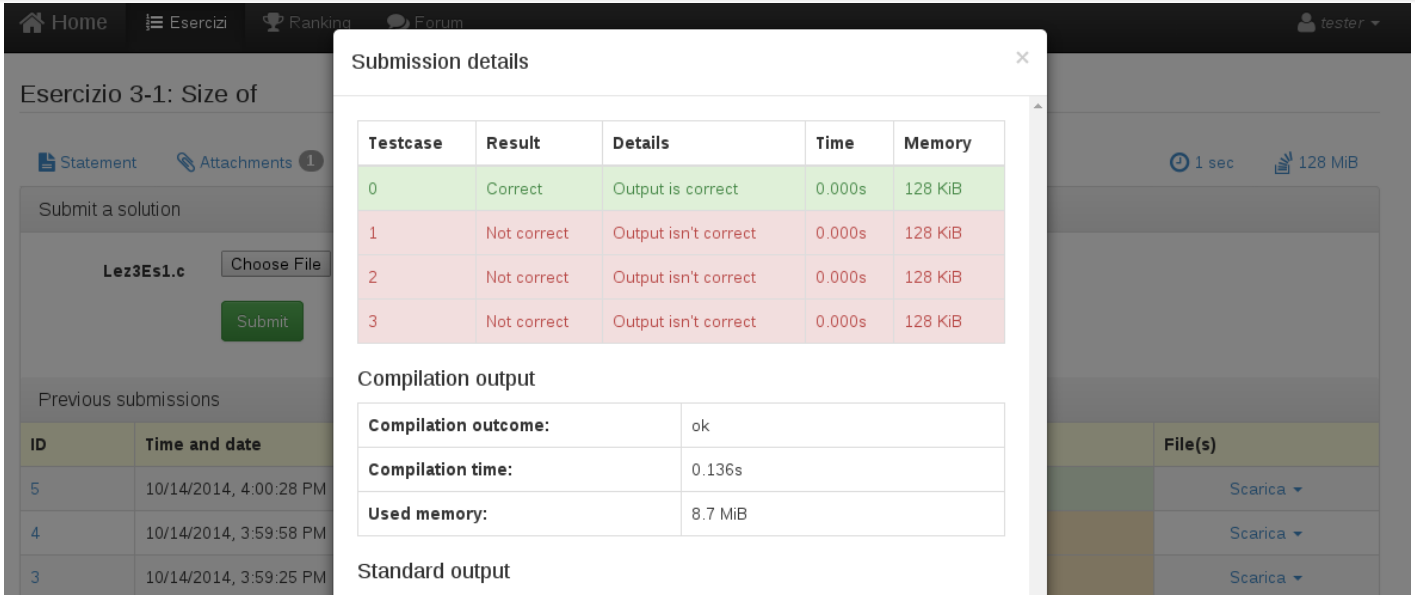
Submission



Potete sottoporre la vostra soluzione (il file `.c`) dalla scheda Submission.

1. il sistema valuta la vostra soluzione;
2. se corretta vi assegna il massimo del punteggio;

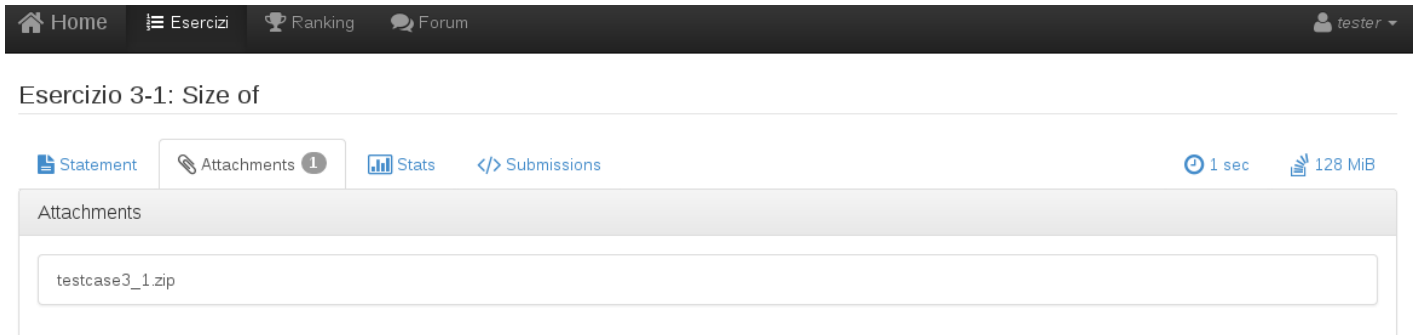
Submission



Potete sottoporre la vostra soluzione (il file `.c`) dalla scheda Submission.

1. il sistema valuta la vostra soluzione;
2. se corretta vi assegna il massimo del punteggio;
3. altrimenti vi segnala errore. Cliccando sull'ID della submission potete conoscere su quali test-case il vostro programma ha fallito (magari tutti :)).

Test-cases



Trovate i vari test-case sotto la scheda Attachments. Sono inseriti in un file .zip e per ognuno avete una coppia `input` e `output` (atteso). Per controllare la correttezza del vostro programma potete usare i file di input:

```
cat input.txt | ./<nome_eseguibile>
cat input.txt | ./<nome_eseguibile> | diff - output.txt
```

Nota: - in `diff` rappresenta standard input

Altro

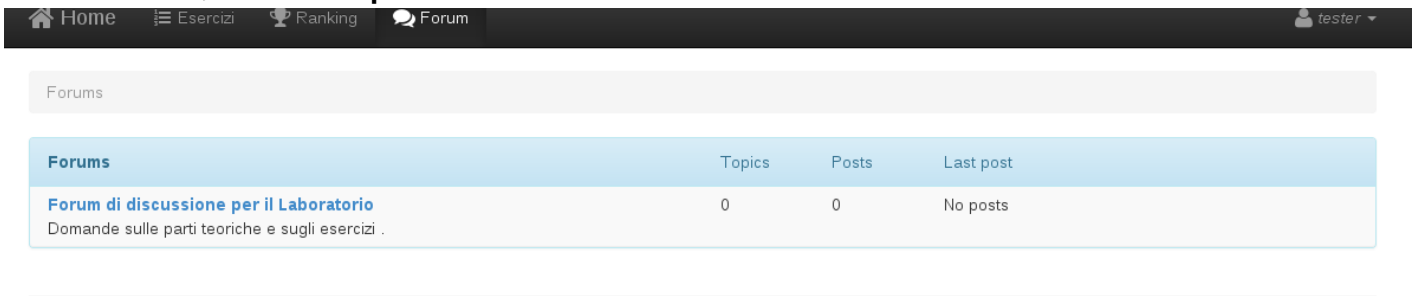
Ricordatevi di terminare con il `return 0`.

Altro

Ricordatevi di terminare con il `return 0`.

Cosa altro c'è:

- un forum, su cui potete discutere dei vari esercizi/soluzioni

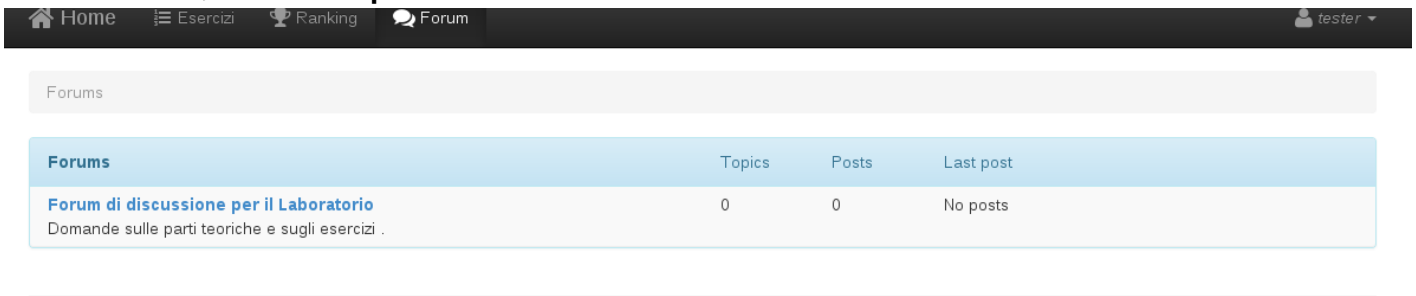


Altro

Ricordatevi di terminare con il `return 0`.

Cosa altro c'è:

- un forum, su cui potete discutere dei vari esercizi/soluzioni



- una classifica ottenuta considerando i punteggi che accumulate esercizio dopo esercizio.

Buon Lavoro!