PROGRAMMAZIONE 2: SPERIMENTAZIONI

Lezione 4 – Numeri casuali con rand(), srand() e time()

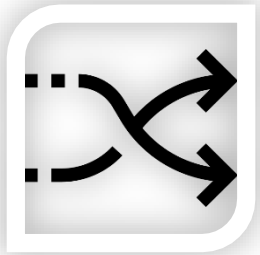
Agenda

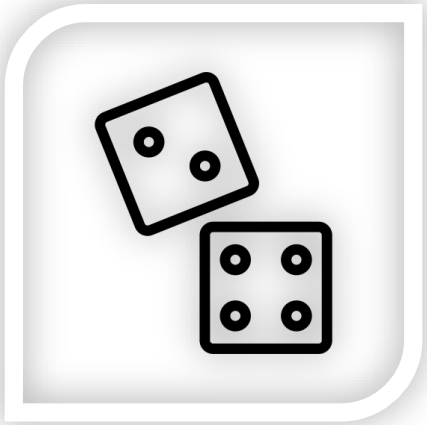
- Introduzione
- Generazione di numeri casuali
- La funzione `rand()`
- La funzione `srand()`
- La funzione `time()`
- Data e ora con `time.h`

Video lezione disponibile su YouTube: https://youtu.be/604jV6pfto8

Introduzione

- L'*elemento di casualità* può essere introdotto nelle applicazioni per computer usando la funzione `rand()` presente nella libreria standard del C `<stdlib.h>`.





Generazione di numeri casuali

- La funzione `rand()` genera un numero intero tra 0 e `RAND_MAX` (una costante simbolica definita in `<stdlib.h>`).
- Il **C standard** stabilisce che il valore di `RAND_MAX` deve essere almeno 32767, che è il valore massimo per un intero di due byte (16 bit → 2^{16} → 65536 numeri → -32767 ... 0 ... 32767).
- Se `rand` produce veramente numeri *a caso*, ogni numero tra 0 e `RAND_MAX` ha la stessa probabilità di essere scelto.

Generazione di numeri casuali

```
int x;
x = rand();
```

- L'intervallo di valori prodotto da `rand` (tra 0 e `RAND_MAX`) viene salvato in `x`.

stdlib.h

```
58  /* RAND_MAX is the maximum value that may be returned by rand.
59   * The minimum is zero.
60   */
61  #define RAND_MAX    0x7FFF
```

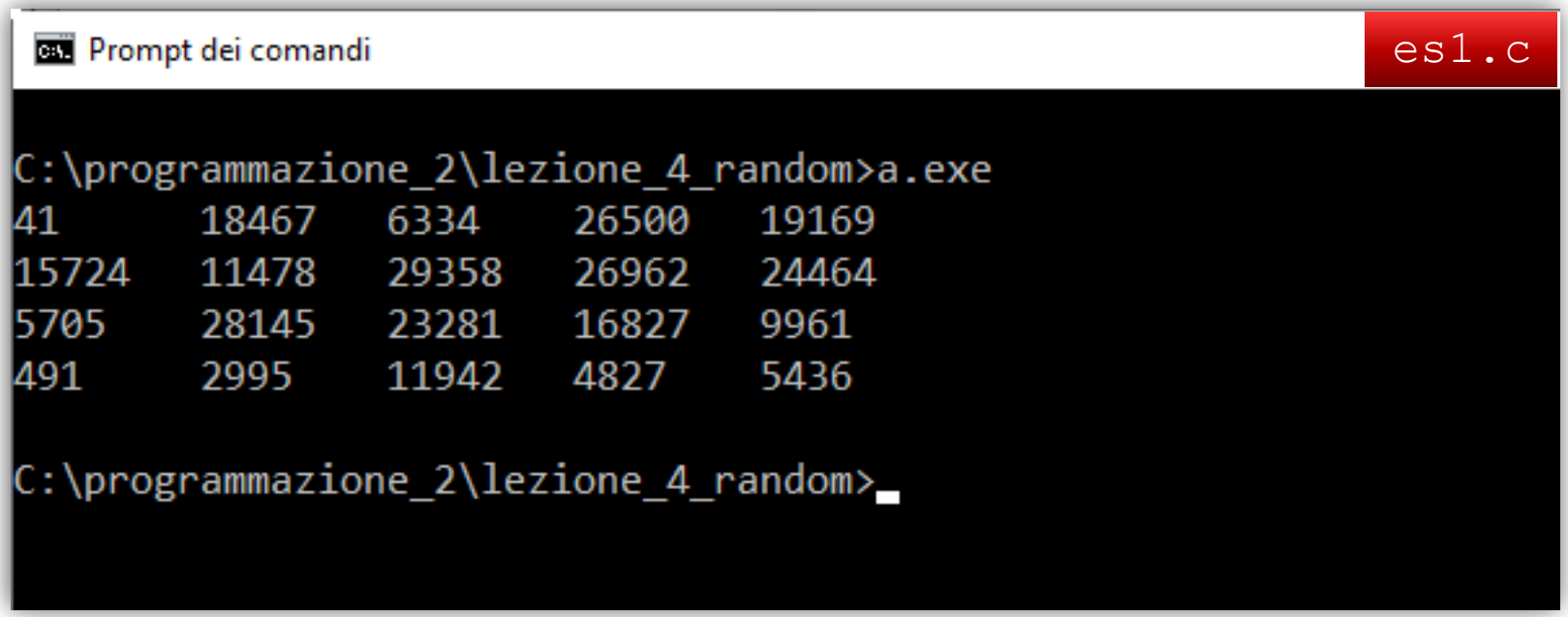
Notazione esadecimale (0x) → 7FFF → **32767** ($7*16^3 + 15*16^2 + 15*16^1 + 15*16^0$)

Generazione di numeri casuali

es1.c

```c
/* Lezione 4: numeri casuali */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i;
    int x;
    int n = 20;
    for (i=1; i<=n; i++){
        x = rand(); // genera numeri casuali
        printf("%d \t", x);
        if(i%5 == 0) // ogni 5 numeri generati crea un nuova riga
            printf("\n");
    }
    return 1;
}
```

Generazione di numeri casuali



Generazione di numeri casuali

- L'intervallo di valori prodotto da `rand` (tra 0 e `RAND_MAX`) **è spesso diverso da quello richiesto da una specifica applicazione**.
- Ad esempio, un programma per il lancio di dadi a sei facce richiede numeri casuali tra 1 e 6, mentre quello per testa o croce necessiterebbe solo di 0 (croce) e 1 (testa).
- Usando l'operatore `%` congiuntamente a `rand()` è possibile ridefinire e restringere l'intervallo di valori generabili.

Generazione di numeri casuali

- Supponendo di voler generare numeri tra 1 e 6:
  - l'operazione `rand() % 6` genera numeri interi tra 0 e 5 (**variazione di scala**) in quanto il resto di una divisione per 6 vale al massimo 5 (`es0.c`);
  - il numero 6 si dice **fattore di scala**;
  - si effettua quindi uno **spostamento** dell'intervallo dei numeri generati aggiungendo 1 al risultato ottenuto dalla `rand()` per ottenere quindi valori casuali tra 1 e 6.

```
int x;
x = 1 + rand() % 6;
```

Generazione di numeri casuali

- Sapendo che i valori generati da `rand()` sono compresi tra `0` e `RAND_MAX`, si può generalizzare la variazione di scala e lo spostamento dell'intervallo per i numeri casuali:

```
n = a + rand() % b;
```

- dove `a` è il **valore di spostamento** (cioè equivale al **primo numero desiderato** nell'intervallo di valori casuali) mentre `b` è il **fattore di scala** (che è uguale **all'ampiezza dell'intervallo** desiderato di valori casuali).
  - L'ampiezza dell'intervallo è data dal numero di valori presenti tra i valori minimo e massimo esprimibili.

Generazione di numeri casuali

```
n = a + rand() % b;
```

- Esempi:

1) numeri casuali tra 1 e 6 (valore massimo compreso)
   - `a = 1; // valore iniziale (minimo)`
   - `b = 6; // da 1 a 6 ci sono 6 numeri`

     `(6 - 1 + 1)` → `rand() % 6` genera al massimo il valore 5
2) numeri casuali tra 0 e 1 (valore massimo compreso)
   - `a = 0; // valore iniziale (minimo)`
   - `b = 2; // da 0 a 1 ci sono 2 numeri`

     `(1 - 0 + 1)` → `rand() % 2` genera al massimo il valore 1
3) numeri casuali tra 18 e 30 (valore massimo compreso)
   - `a = 18; // valore iniziale (minimo)`
   - `b = 13; // da 18 a 30 ci sono 13 numeri`

     `(30 - 18 + 1)` → `rand() % 13` genera al massimo il valore 12
4) numeri casuali tra -20 e + 20
   - `a = -20; // valore iniziale (minimo)`
   - `b = 41; // da -20 a +20 ci sono 41 numeri`

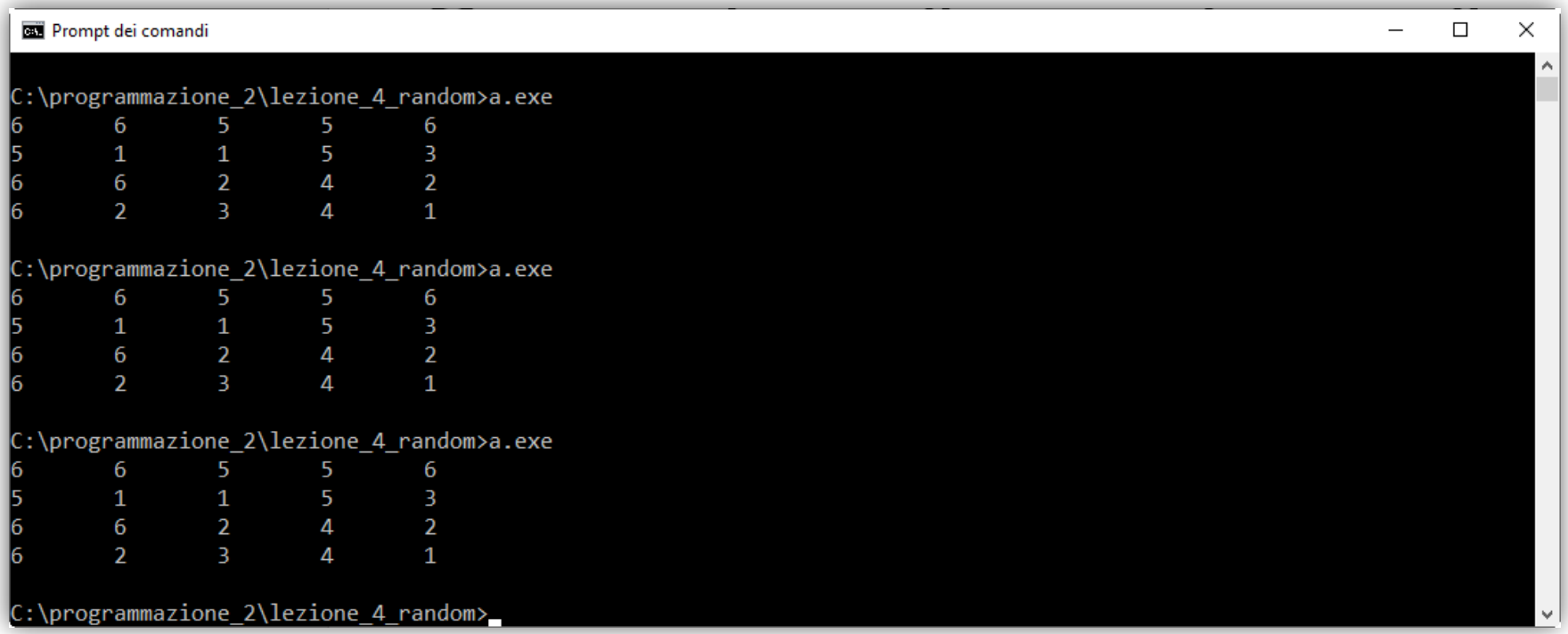
     `(20 - (-20) + 1` → `20 + 20 + 1` → `41)` → `rand() % 41` genera al massimo il valore 40

Generazione di numeri casuali

es2.c

```
int main(void)
{
  int i;
  // valore di spostamento (valore iniziale dell'intervallo)
  int a = 1;
  // fattore di scala (numero di valori nell'intervallo)
  int b = 6;
  int x;
  int n = 20;
  for (i=1; i<=n; i++){
     x = a + rand() % b;
     printf("%d \t", x);
     // ogni 5 numeri generati crea un nuova riga
     if(i%5 == 0)
        printf("\n");
  }
  return 1;
}
```

Generazione di numeri casuali



Generazione di numeri casuali

- Eseguendo più volte il programma si può notare che viene stampata *esattamente la stessa sequenza di valori*.
- **Come possono essere casuali questi numeri?**
  - Per ironia della sorte, questa *ripetibilità* è una caratteristica importante della funzione `rand`, utilizzata nelle fasi di *debugging* (“ricerca di errori e correzioni”).
  - Se è necessario correggere un programma è bene che produca valori apparentemente casuali, finché non è testato per verificarne il comportamento in tutti i casi.

Generazione di numeri casuali

- La funzione `rand` genera in verità numeri **pseudocasuali**.
- È possibile far sì che il programma produca una sequenza differente di numeri casuali attraverso la **randomizzazione**.
  - La **randomizzazione** avviene attraverso la funzione `srand` (definita in `<stdlib.h>`).
  - La funzione `srand` riceve come argomento un intero *unsigned* (senza segno) e fornisce un **seme** (**seed**) alla funzione `rand` per generare una sequenza diversa di numeri casuali ad ogni esecuzione del programma.

Generazione di numeri casuali

- Un metodo sicuro per randomizzare attraverso la `srand` è il seguente.

```
srand(time(NULL));
```

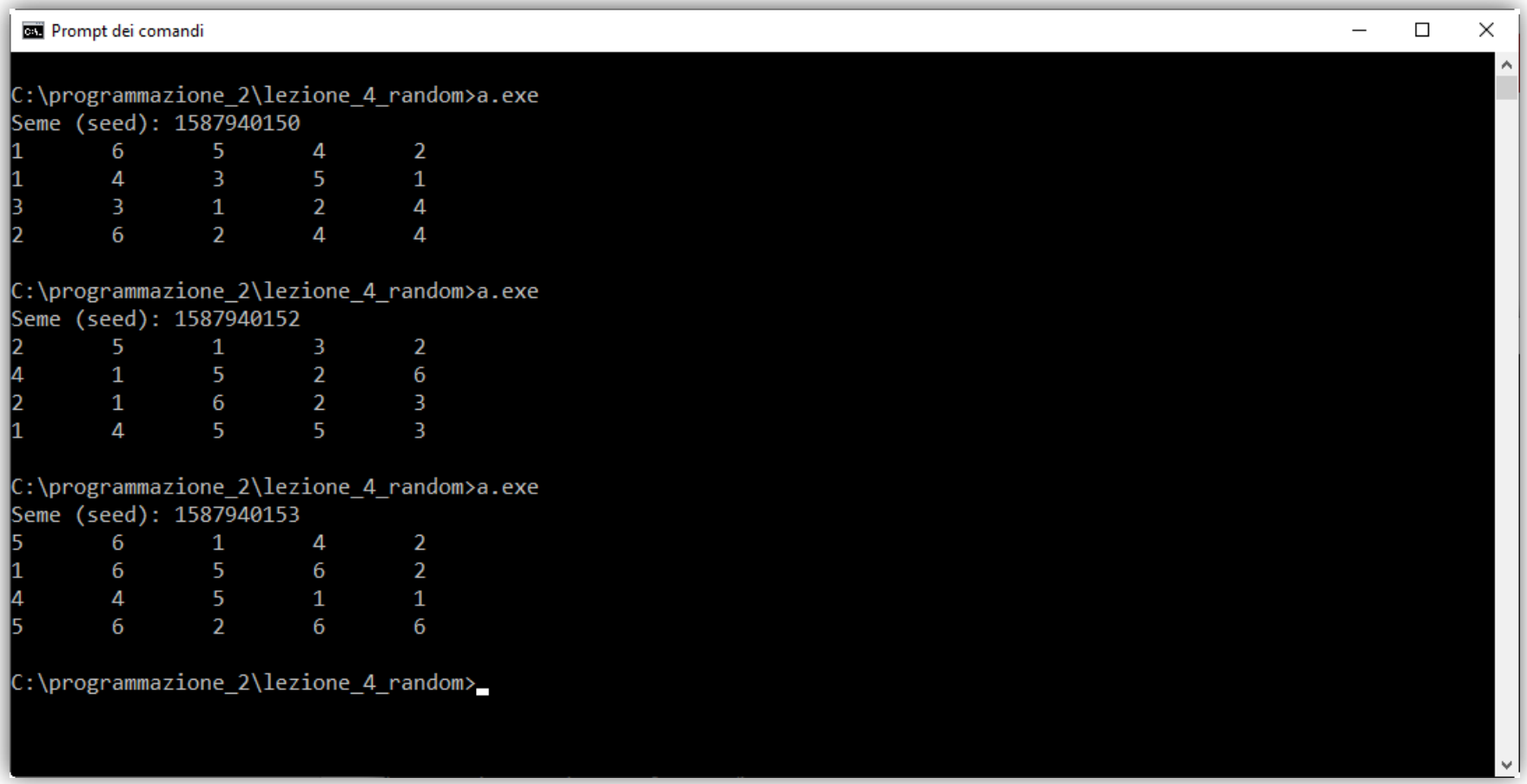
- Tale metodo fa sì che il computer legga il suo orologio interno per ottenere automaticamente un valore per il seme (seed).
- La funzione `time` (definita in `<time.h>`) restituisce un `time_t` ovvero il numero di secondi che sono trascorsi dalla mezzanotte del 1 Gennaio 1970.
- Utilizzando la `time` quindi, la `srand` conterrà ogni volta un valore diverso in input (essendo un tempo che cresce costantemente).

Generazione di numeri casuali

es3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h> // contiene il prototipo per la funzione time

int main(void)
{
   int i;
   // valore di spostamento (valore iniziale dell'intervallo)
   int a = 1;
   // fattore di scala (numero di valori nell'intervallo)
   int b = 6; // da 1 a 6 ci sono 6 numeri
   int x;
   int n = 20;
   unsigned int t = time(NULL); // tempo dell'orologio di sistema in secondi
   printf("Seme (seed): %u \n", t); // da visualizzare solo in fase di debug
   srand(t); // oppure srand(time(NULL))
   for (i=1; i<=n; i++){
      x = a + rand() % b;
      printf("%d \t", x);
      if(i%5 == 0) // ogni 5 numeri generati crea un nuova riga
         printf("\n");
   }
   return 1;
}
```

Generazione di numeri casuali



Data e ora con `time.h`

- Come si può notare, la `time()` utilizzata all'interno di `srand()` ha un parametro `NULL`.
- Questo perché la `time()` richiede un parametro, costituito da un puntatore di tipo `time_t *`: se questo puntatore è valido, la stessa informazione che viene restituita viene anche memorizzata nell'indirizzo indicato da tale puntatore.
- Inserendo `NULL` come parametro, si indica che non si vuole passare alcun puntatore.

Data e ora con `time.h`

es4.c

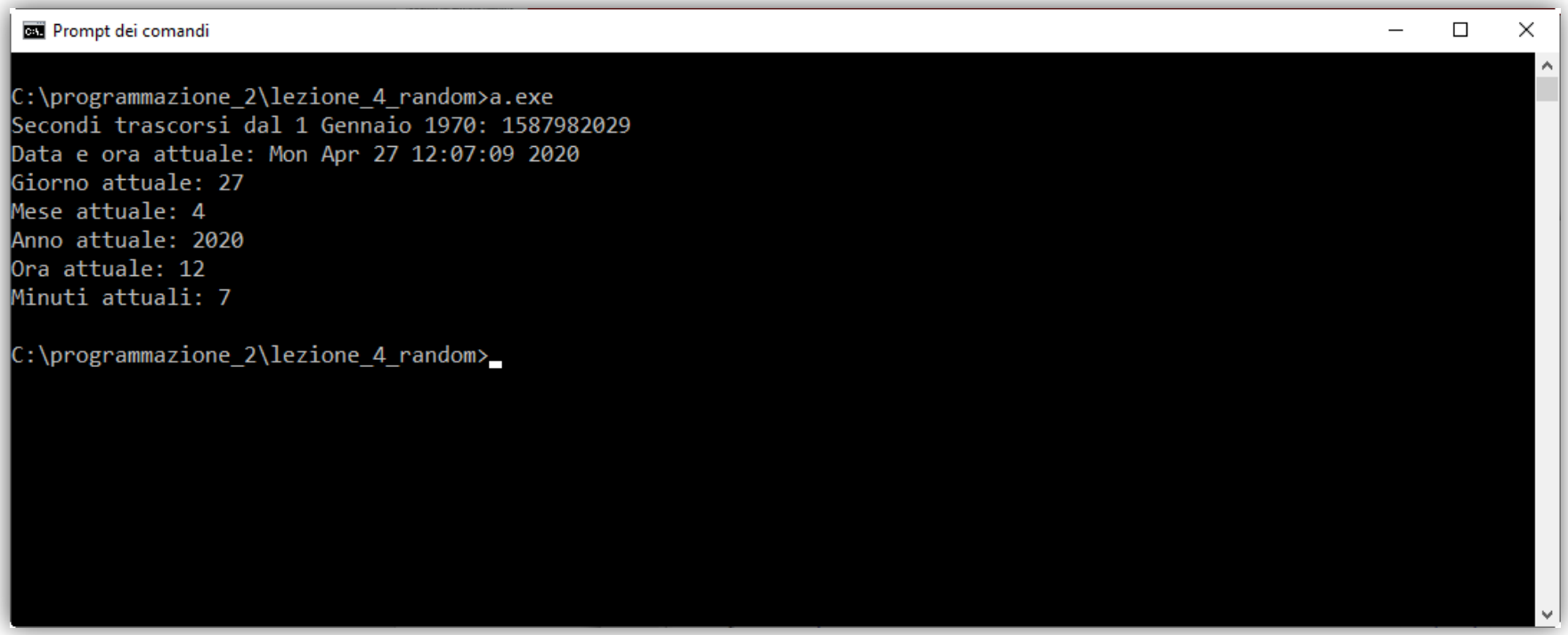
```
struct tm
{
 int tm_sec;  // Secondi: da 0 a 60.
 int tm_min;  // Minuti: da 0 a 59.
 int tm_hour; // Ora: da 0 a 23.
 int tm_mday; // Giorno del mese: da 1 a 31.
 int tm_mon;  // Mese dell'anno: da 0 a 11.
 int tm_year; // Anno dal 1900.
 int tm_wday; // Giorno della settimana: da 0 a 6
 // con lo zero corrispondente alla domenica.
 int tm_yday; // Giorno dell'anno: da 0 a 365.
 int tm_isdst;
 /* Contiene 1 se è in vigore l'ora "estiva" (legale);
 0  se l'ora è quella "normale" ovvero quella invernale
(solare); -1 se l'informazione non è disponibile. */
};
```

Data e ora con `time.h`

es4.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h> // contiene il prototipo per la funzione time

int main(void)
{
    // time_t (è un unsigned int)
    time_t tmppointer;

    // struct tm definita in time
    struct tm *timeinfo;

    unsigned int s; // time_t

    // time estrae la data dall'orologio del computer
    s = time(&tmppointer);

    printf("Secondi trascorsi dal 1 Gennaio 1970: %u \n", s);

    // localtime converte un time_t ricevuto in input in data e ora locale
    timeinfo = localtime(&tmppointer);

    // asctime converte la struttura struct tm in una stringa
    printf("Data e ora attuale: %s", asctime(timeinfo));
    // si può accedere agli elementi di struct tm
    printf("Giorno attuale: %d \n", timeinfo->tm_mday);
    // tm_mon indica i mesi partendo da 0 (gennaio è 0, febbraio è 1, ecc...)
    printf("Mese attuale: %d \n", timeinfo->tm_mon+1);
    // tm_year indica quanto è passato dal 1900 (se siamo nel 2020 indica 120)
    printf("Anno attuale: %d \n", timeinfo->tm_year+1900);
    printf("Ora attuale: %d \n", timeinfo->tm_hour);
    printf("Minuti attuali: %d \n", timeinfo->tm_min);

    return 1;
}
```

Data e ora con `time.h`



Data e ora con `time.h`

- È anche possibile generare una data in modo "manuale" senza ottenerla dall'orologio del computer.
- Ad esempio, potrebbe essere necessario creare una data a seguito di una stringa ricevuta in input da un utente o dopo aver letto tale stringa da un file di testo.

Data e ora con `time.h`

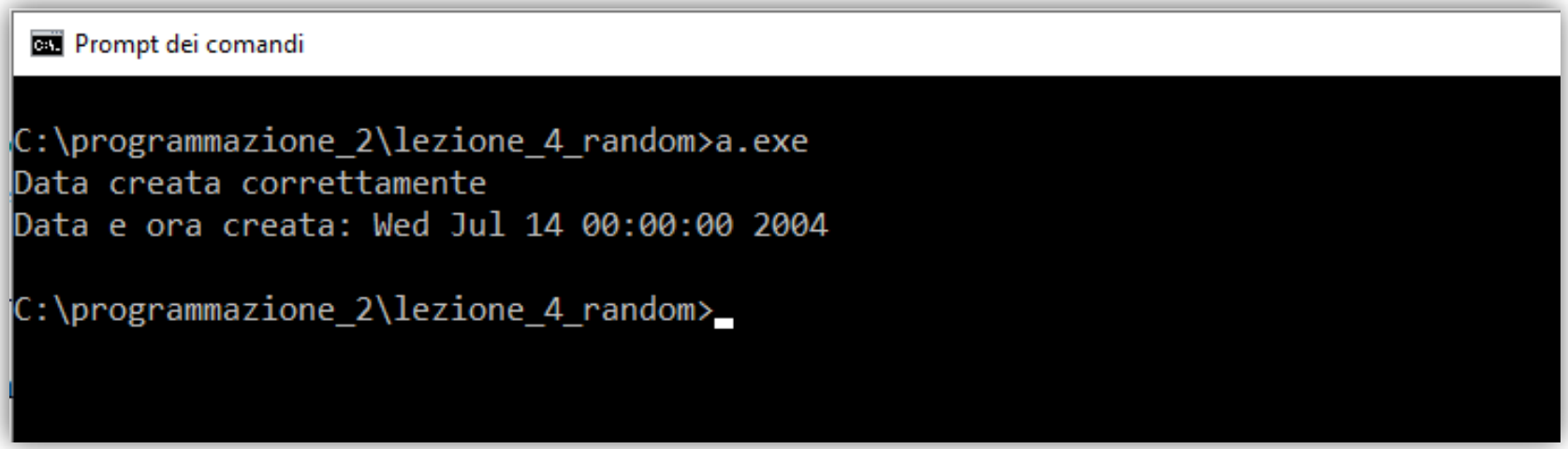
- La funzione `mktime()` (definita in `<time.h>`) riceve come argomento il puntatore a una variabile strutturata di tipo `struct tm`, contenente le informazioni sulla data e determina il valore di quella data secondo la rappresentazione interna, di tipo `time_t`.
- Se la funzione non è in grado di restituire un valore rappresentabile nel tipo `time_t` o, se non può eseguire il suo compito, restituisce il valore -1.

Data e ora con `time.h`

es5.c

```c
int main(void)
{
    // struttura per creare la data manualmente
    struct tm timeinfo;

    time_t tmpPointerA;

    struct tm *timeinfoBuilt;

    timeinfo.tm_year = 104; // 2007 - 1900
    timeinfo.tm_mon = 6; // mese: 0 gennaio, 1 febbraio, ecc...
    timeinfo.tm_mday = 14; // giorno
    timeinfo.tm_hour = 0;
    timeinfo.tm_min = 0;
    timeinfo.tm_sec = 0;

    tmpPointerA = mktime(&timeinfo);

    if (tmpPointerA == -1)
    {
        printf ("Errore! %d\n", tmpPointerA);
    }
    else
    {
        timeinfoBuilt = localtime(&tmpPointerA);
        printf("Data creata correttamente\n");
        // asctime converte la struttura struct tm in una stringa
        printf("Data e ora creata: %s", asctime(timeinfoBuilt));
    }

    return 1;
}
```

Data e ora con `time.h`



FINE PRESENTAZIONE