UNIVERSITÀ
degli STUDI
di CATANIA

Generazione di numeri pseudo-casuali in C/C++

Corso di programmazione I AA 2019/20

Corso di Laurea Triennale in Informatica

Prof. Giovanni Maria Farinella
Web: http://www.dmi.unict.it/farinella
Email: gfarinella@dmi.unict.it

Dipartimento di Matematica e Informatica

Simulazioni

Un **programma di simulazione** permette di usare il calcolatore per **simulare** una o più attività del mondo reale. Applicazioni:

- predictions/forecasting sulla base di un set di dati che rappresentano eventi passati.
- gestione di magazzino/scorte
- varie applicazioni di carattere scientifico o finanziario.

Simulazioni

Le simulazioni necessitano di dati in input che modellano alcuni aspetti del del mondo reale.

Ad esempio

il numero di clienti che mediamente si reca in negozio nell'arco di un'ora..

Per simulare un determinato aspetto della realtà, come l'arrivo dei clienti presso un negozio, si fa uso di **un generatore di numeri pseudo-casuali**.

Un bambino che effettua estrazioni con ripetizione da una urna è un generatore di numeri casuali.

Un generatore di numeri pseudo-casuali rappresenta una sorta di "approssimazione" del bambino che effettua estrazioni: output è rappresentato da una sequenza di numeri appartenenti ad certo intervallo $[a, b] \in \mathbb{N}$ che "sembrino" casuali.

Generatori di numeri pseudo-casuali

I generatori di numeri pseudo-casuali sono **algoritmi**!

Di conseguenza, fissato il set di parametri in input, otterremo in output lo **stesso risultato**, ovvero la sequenza di numeri **conforme alla logica dello algoritmo.**: in molti casi tale caratteristica è desiderabile (proprietà di controllabilità).

Generatori di numeri pseudo-casuali

Un generatore di numeri pseudo-casuali dovrebbe avere le segg. caratteristiche:

- **random**: il generatore dovrebbe essere capace di superare alcuni *test di randomicità* sulle sequenze prodotte;
- **controllabile**: il generatore dovrebbe generare la stessa identica sequenza con lo stesso input (parametri vari, come il seme);
- **portabile** su differenti architetture;
- **efficiente** in termini di risorse di calcolo occupate per la sua esecuzione;

Generatori di numeri pseudo-casuali

Esempio. Ottenere sequenze di numeri "realmente" casuali

Posizionare un sensore fuori dalla finestra per un giorno e misurare una o più grandezze (ad esempio l'intensità della luce ad intervalli di tempo fissati..)

Le funzioni rand() ed srand() di stdlib.h

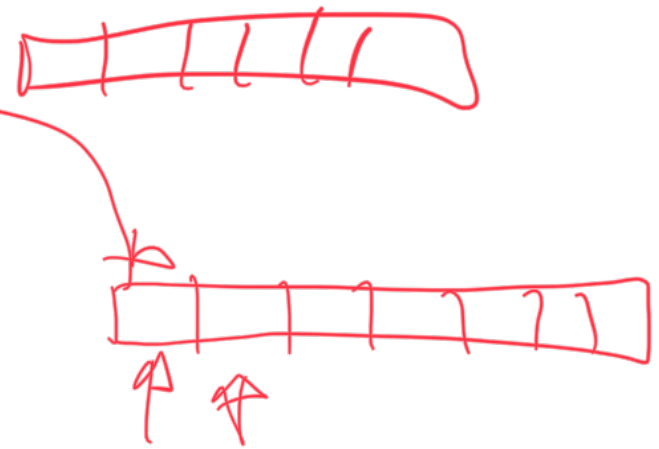
La libreria standard del linguaggio C/C++ include alcune funzioni di base per la generazione di numeri pseudo-casuali.

```
1  #include <cstdlib> // per rand() ed srand()
2  #include <ctime> // per la funzione time()
3
4  srand(111222333); // seme
5  //oppure
6  srand(time(0));
7
8  for(int i=0; i<1000; i++)
9     cout << rand() << endl;
```

Generazione di una sequenza di 1000 numeri "casuali".

Le funzioni rand() ed srand() di stdlib.h

```
1  #include <cstdlib>
2  #include <ctime>
3
4  srand(111222333); // seme
5  //oppure
6  srand(time(0));
7
8  for(int i=0; i<1000; i++)
9     cout << rand() << endl;
```



La funzione `rand()` estrae il "prossimo" numero della sequenza casuale (includere header C **stdlib.h**).

Le funzioni rand() ed srand() di stdlib.h

```
1  #include <cstdlib>
2  #include <ctime>
3
4  srand(111222333); // seme
5  //oppure
6  srand(time(0));
7
8  for(int i=0; i<1000; i++)
9     cout << rand() << endl;
```

Una invocazione della funzione `rand()` darà come risultato un `unsigned int` nel range $[0, \text{RAND\_MAX}]$

Le funzioni rand() ed srand() di stdlib.h

```
1  #include <cstdlib>
2  #include <ctime>
3
4  srand(111222333); // seme
5  //oppure
6  srand(time(0));
7
8  for(int i=0; i<1000; i++)
9     cout << rand() << endl;
```

NB: Una volta fissato il seme, la sequenza sarà sempre la stessa ad ogni esecuzione del programma (provare!).

Le funzioni rand() ed srand() di stdlib.h

```
1  #include <cstdlib>
2  #include <ctime>
3
4  srand(111222333);
5  //oppure
6  srand(time(0));
7
8  for(int i=0; i<1000; i++)
9     cout << rand() << endl;
```

La funzione `time()` restituisce il numero di secondi decorsi a partire dal 1 Gennaio 1970 (00:00) rispetto alla data e ora corrispondenti all'invocazione della funzione.

Le funzioni rand() ed srand() di stdlib.h

```
1  #include <cstdlib>
2  #include <ctime>
3
4  srand(111222333);
5  //oppure
6  srand(time(0));
7
8  for(int i=0; i<1000; i++)
9     cout << rand() << endl;
```

Dunque usare la funzione `time()` garantisce che a **differenti esecuzioni del programma** (che ovviamente avvengono in istanti differenti) corrispondano **differenti sequenze di numeri pseudo-casuali**.

Le funzioni rand() ed srand() di stdlib.h

Come generare numeri (interi) nel range $[a, b] \in \mathbb{N}$ $(a < b)$?

Dati due numeri naturali a **e** b **con** $a < b$**, ci sono** $(b - a + 1)$ **valori nell'intervallo** $[a, b]$**.**

Il risultato della chiamata `rand()%P`, con $0 < P <= RAND\_MAX$, sarà un numero $\mathbf{0} \leq \mathbf{r} < \mathbf{P}$.

Dunque la seguente istruzione

```
unsigned int r = rand()%(b - a + 1) + a
```

lascerà nella variabile r un numero compreso tra a e b (estremi inclusi).

Le funzioni rand() ed srand() di stdlib.h

Come generare numeri pseudo-casuali in virgola mobile? Ad esempio numeri compresi tra 0 e 1?

NB: bisogna fare attenzione a "forzare" uan divisione in virgola mobile.

```
double r = rand() / (RAND_MAX * 1.0);
```

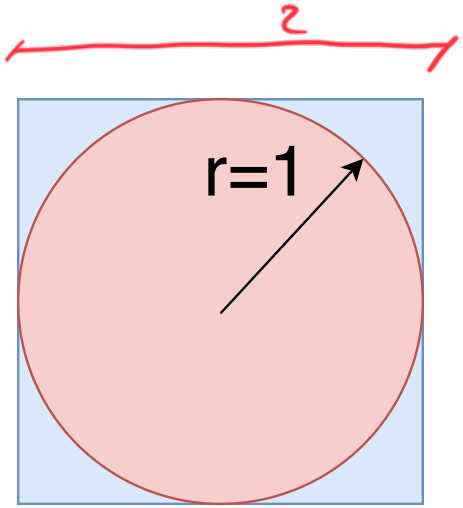
oppure

```
double r = ((double) rand()) / RAND_MAX;
```

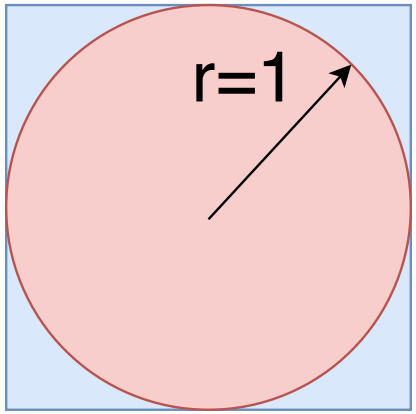
Esempio E16.1. Metodo Montecarlo per calcolare il numero π

I metodi Montecarlo si basano sul **campionamento casuale**, e sono concepiti per ottenere risultati numerici.

Si consideri il quadrato di lato 2 ed il cerchio di raggio unitario.



Esempio E16.1. Metodo Montecarlo per calcolare il numero π

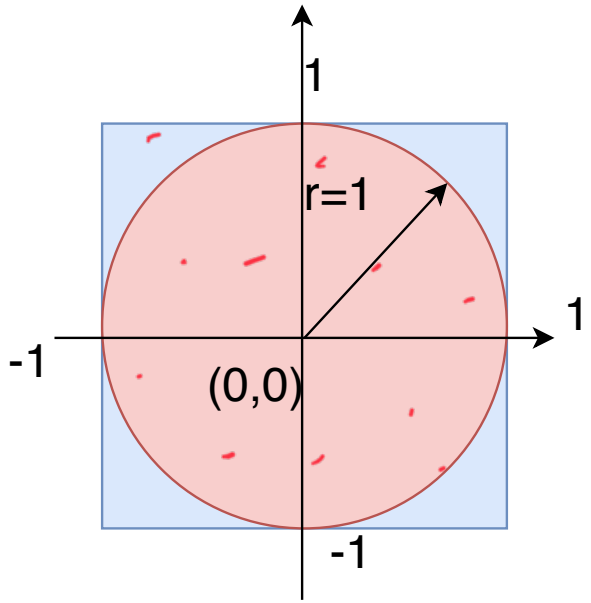


L'area del cerchio $A_c = \pi \times r^2 = \pi$

Area del quadrato $A_q = (2r)^2 = 4$

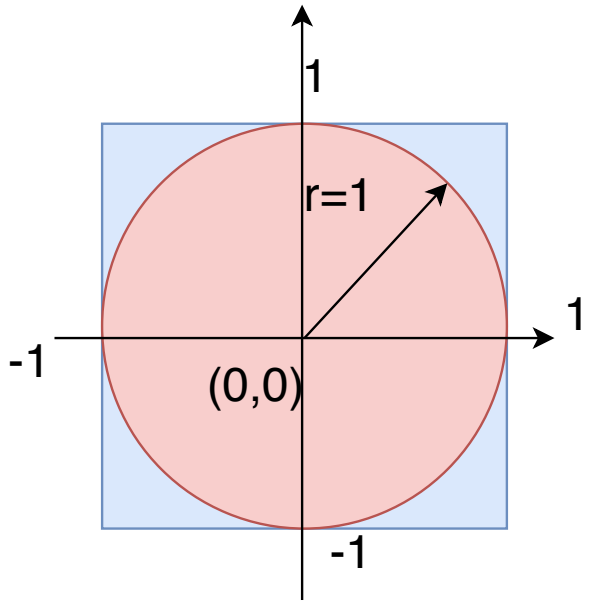
Il rapporto $\frac{A_c}{A_q} = \frac{\pi}{4}$

Esempio E16.1. Metodo Montecarlo per calcolare il numero π



Sistema di riferimento cartesiano avente origine coincidente con il centro del cerchio.

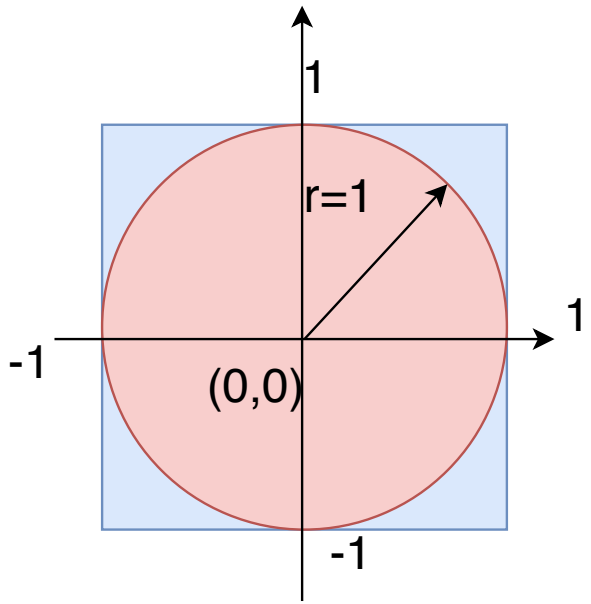
Esempio E16.1. Metodo Montecarlo per calcolare il numero π



Sequenza di campionamenti di coppie di numeri casuali (x, y), nel quale sia x che y potranno assumere valori in $[-1, 1]$.

Equivalente a **generare coordinate cartesiane** (x, y) distribuite nel quadrato di lato 2 che "ospita" il cerchio.

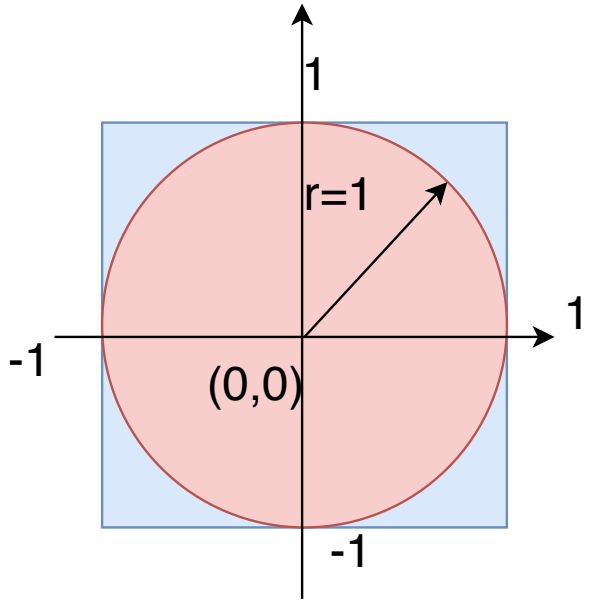
Esempio E16.1. Metodo Montecarlo per calcolare il numero π



Dopo ogni estrazione della coppia di valori (x, y), ci chiediamo se essa rappresenta un punto all'interno del cerchio, ovvero se

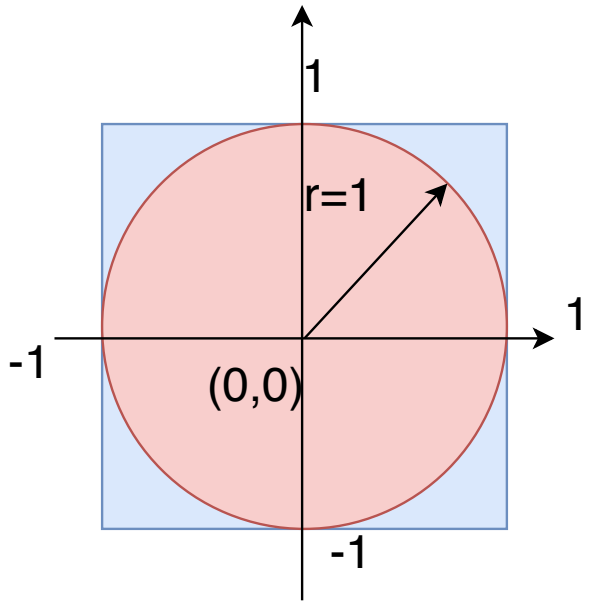
$$\mathbf{x^2 + y^2 \leq 1}$$

Esempio E16.1. Metodo Montecarlo per calcolare il numero π



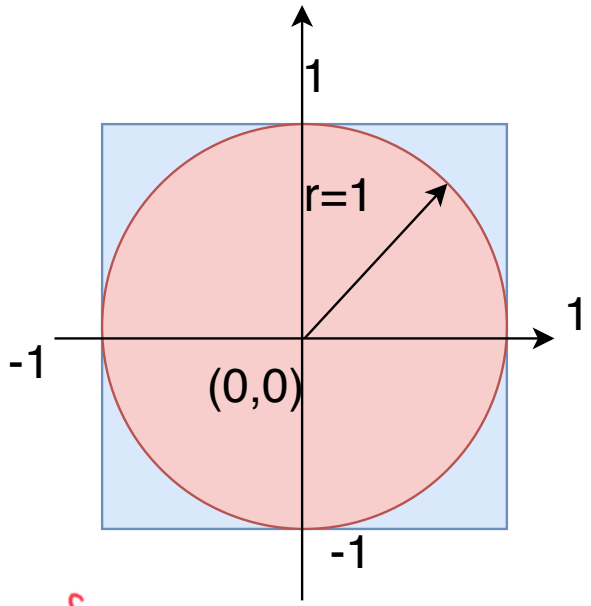
Controllare che la relazione ($\mathbf{x^2 + y^2 \leq 1}$) sia verificata equivale a **simulare il tiro delle freccette all'interno del quadrato e poi verificare che la freccetta sia caduta dentro il cerchio!**

Esempio E16.1. Metodo Montecarlo per calcolare il numero π



La probabilità relativa alla estrazione del singolo numero dell'intervallo $[-1, 1]$ è identica per tutti i numeri.

Esempio E16.1. Metodo Montecarlo per calcolare il numero π



$\pi = \frac{S}{T} 4$

Di conseguenza il **rapporto tra il numero di successi (freccetta dentro il cerchio) e il numero totale di lanci** sarà

$$\frac{S}{T} = \frac{A_c}{A_Q} = \frac{\pi}{4};$$ (nella quale S, T sono note).

S denota il numero totale di successi, Tot il numero totale di lanci.

Homework

Homework H16.1

Scrivere un programma in C++ che permetta di simulare una sequenza di N lanci di una coppia di dadi, dove N è un numero scelto dall'utente oppure una costante scelta a tempo di compilazione. Il programma dovrà stampare le sequenze dei due numeri (output primo dado e output secondo dado) in due colonne separate. ES:

1 6
3 4
1 2
6 3

Homework

Homework H16.2

Codificare in C++ un programma per il metodo montecarlo per il calcolo (approssimato) del numero π sulla base che si basi sull'esempio E16.1.

NB: Il numero di campionamenti deve essere scelto dall'utente a tempo di esecuzione.

Homework H16.3

Codificare in C++ un programma *battaglia navale* in cui l'utente gioca contro il calcolatore.

Il programma chiede all'utente le seguenti informazioni:

- il nome del giocatore
- la dimensione del campo di gara
- la dimensione della tabella di gara.
- la dimensione minima e massima delle navi, intesa come numero di celle occupate da ogni nave.

Homework H16.3

Il programma dispone in modo casuale un certo numero di navi nella tabella del giocatore e nella tabella dell'avversario (il calcolatore).

Il programma visualizza sempre e solo le due tabelle del giocatore, i) quella che contiene le navi del giocatore e ii) quella che contiene le mosse contro l'avversario. In questa ultima tabella saranno visibili solo i tiri a vuoto e i tiri a segno, mentre nella prima tabella saranno visibili le navi e i tiri a segno.

Dopo l'inizio della battaglia, il programma chiede all'utente le coordinate dell'obiettivo (cella) da colpire, di conseguenza aggiorna le tabelle del giocatore.

Homework H16.3

La mossa del calcolatore avverrà dopo ogni mossa del giocatore. Anche in questo caso il giocatore vedrà le sue tabelle aggiornate.

Il giocatore può chiedere al programma di interrompere il gioco in qualsiasi momento, in questo caso il programma offre al giocatore la possibilità di salvare lo stato del gioco su un file, e di scegliere un nome.

Inoltre, ad ogni avvio del programma, l'utente può scegliere se iniziare una nuova sessione di gioco oppure caricare una sessione di gioco salvata precedentemente in un file.

FINE