Programiranje u C-u ili C++-u
Pseudo-slučajni brojevi; Dinamička alokacija memorije

Generisanje slučajnih brojeva

- Postoje dve metode koje se koriste za generisanje slučajnih brojeva. Prva metoda meri neku fizički pojavu za koju se očekuje da će biti slučajna, a zatim se u procesu merenja eliminiše ne-slučajnost. Izvorni primeri uključuju merenje atmosferskog šuma, termičkog šuma, i ostalih eksternih elektromagnetnih i kvantnih pojava. Najranije metode za generisanje slučajnih brojeva, kao što su kocke, bacanje novčića i rulet, se i dalje koriste danas, pre svega u igrama i kockanju, jer su prespore za većinu primena u statistici i kriptografiji.

- Druga metoda koristi računarske algoritme koji mogu da proizvedu duge nizove od navodno slučajnih rezultata, koji su u stvari u potpunosti određeni manjom početnom vrednošću, poznata kao semena vrednost ili ključ. Kao rezultat, ukupni pseudo-slučajni niz se može stvoriti ako je semena vrednost poznata. Ovaj tip generisanja slučajnih brojeva se često naziva pseudo-slučajni generator brojeva.

➢ Pseudo slučajni brojevi po svojoj suštini, odnosno načinu nastanka, nisu slučajni, već su posledica primene odgovarajućeg algoritma, ali poseduju osobine koje u potrebnoj meri odgovaraju nizu slučajnih brojeva. Te osobine podrazumevaju pre svega nezavisnost, saglasnost sa odgovarajućim raspodelama verovatnoća koja se dokazuje statističkim testovima, kao i zadovoljenje testa autokorelativnosti.

✓ Prvi od pristupa generisanju pseudoslučajnih brojeva predložio je Džon fon Nojman 1946, za potrebe korišćenja metoda Monte Karlo na računaru ENIAC. To je bio jednostavan algoritam zasnovan na metodu "sredine kvadrata". Ovaj metod podrazumeva da se za generisanje niza pseudoslučajnih izabere jedan početni broj (seme), te da se nakon kvadriranja tog broja, sledeći dobija kao niz središnjih cifara dobijenog kvadrata, nakon čega se postupak ponavlja. Ako je izabrani semeni broj 6457, njegov kvadrat iznosi 41692849, te je prvi "slučajni broj" predstavljen sa četiri središnje cifre 6928. Kvadriranjem ovog broja i ponavljanjem procesa dobija se niz čija su prva četiri člana: 6928, 9971, 4208 i 7072. Medjutim, kod ovog algoritma, period do ponavljanja za n-to cifrene brojeve ne može biti duži od 10^n, te je primena ovog pristupa ograničena.

❖ Pored ovoga razvijen je i niz drugih algoritama kao što su: linearni kongruentni generator, ili inverzni kongruentni generator, odnosno savremeniji, kao što su BlumBlumShub-ov generator (Blum L., et al. 1986) ili Mersenne twister (Matsumoto M., Nishimura T., 1998).

❖ **Linearni kongruentni generator** predstavlja jedan od najstarijih i najpoznatijih Algoritama. Ovaj algoritam, predložen pre više od pola veka (Lehmer, 1951) zasnovan je na primeni sledeće rekurentne jednačine:

$X_{j+1}=(aX_j+b) \bmod m$

gde su:

Xj – pseudo slučajni brojevi (X_0, broj seme čija se vrednost zadaje)

a,b,m – celobrojne konstante, a>0, b>0,m>0

mod – operator koji označava ostatak pri deljenju dva broja

Period linearnog kongruentnog generatora iznosi najviše m, ali je često manji, te primena ovog generatora u mnogome zavisi od izbora vrednosti za a,b i m. S obzirom da se u simulaciji koriste slučajni brojevi u intervalu (0,1), najčešće se koristi transformacija $X_j^{(0,1)} = X_j / m$

U tabeli je dat primer sekvence pseudo slučajnih brojeva dobijenih primenom linearnog kongruentnog generatora, za m=11123, a=1341, c=2977 i X0=8543.
Primer korišćenja linearnog kongruentnog generatora

REDNI BROJ	$aX_j + m$	X_j	$X_j^{(0,1)} = X_j / m$
1	11459140	2450	0.22026
2	3288427	7142	0.64209
3	9580399	3496	0.31430
4	4691113	8330	0.74890
5	11173507	6015	0.54077
6	8069092	4917	0.44206
7	6596674	735	0.06608

Opširnije
http://www.cplusplus.com/reference/random/linear_congruential_engine/
http://www.cplusplus.com/reference/random/mersenne_twister_engine/

Zadatak 3

C rešenje

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

void greska()
{
  fprintf(stderr, "-1\n");
  exit(EXIT_FAILURE);
}

/*Funkcija koja alocira niz
 * i vraca pokazivac na prvi
 * element niza
 */
```

```
int* alociraj1(int n){
  int* a = malloc(n * sizeof(int));
  if(a == NULL)
    greska();
  return a;
}

/*Funkcija koja alocira niz
 *cija se adresa prosledjuje
 * kao argument funkciji*/
void alociraj2(int** a, int n){
  *a = malloc(n * sizeof(int));
  if(*a == NULL)
    greska();
}

int main(){

  int i, n;
  int *a;

  //Ucitavamo dimenziju niza
  scanf("%d", &n);
  if(n <= 0)
    greska();

  //Alociramo prostor
  a = (int*)malloc(n*sizeof(int));
  if(a == NULL)
    greska();

  /*
   II nacin:
   a = alociraj1(n);
   */

  /*
   * III nacin:
     alociraj2(&a, n);
   */

  //Ucitavamo elemente
  for(i=0; i<n; i++)
    scanf("%d", &a[i]);

  //Ispisujemo rezultat
  for(i=n-1; i >= 0; i--)
```

```
   printf("%d ", a[i]);
  printf("\n");

  //Oslobadjamo prethodno alocirani niz
  free(a);

  return 0;

}
```

C++

I način

```
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

void greska()
{
  cerr << "-1\n";
  exit(EXIT_FAILURE);
}


int main(){

  int i, n;
  vector <int> a;

  //Ucitavamo dimenziju niza
  cin >> n;
  if(n <= 0)   greska();

  //Ucitavamo elemente
  for(i=0; i<n; i++)
  {  int x;
     cin >> x;
    a.push_back(x);
  }


  //Ispisujemo rezultat
  for(i=n-1; i >= 0; i--)
     cout <<a[i] <<" ";
  cout<<"\n";
```

```
  return 0;

}
```

II način

```
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

void greska()
{
  cerr << "-1\n";
  exit(EXIT_FAILURE);
}


int main(){

  int i, n;

  //Ucitavamo dimenziju niza
  cin >> n;
  if(n <= 0)
    greska();

  vector <int> a(n);

  //Ucitavamo elemente
  for(i=0; i<n; i++)
  {
     cin >> a[i];
  }


  //Ispisujemo rezultat
  for(i=a.size()-1; i >= 0; i--)
     cout <<a[i] <<" ";
  cout<<"\n";

  return 0;

}
```

III način (NIKAKO, variable length array)

```
#include <iostream>
#include <cstdlib>
```

```
using namespace std;

void greska()
{
  cerr << "-1\n";
  exit(EXIT_FAILURE);
}


int main(){

  int i, n;

  //Ucitavamo dimenziju niza
  cin >> n;
  if(n <= 0)
    greska();

  int a[n];  //OPREZ: Variable length array

  //Ucitavamo elemente
  for(i=0; i<n; i++)
  {
     cin >> a[i];
  }


  //Ispisujemo rezultat
  for(i=n-1; i >= 0; i--)
     cout <<a[i] <<" ";
  cout<<"\n";

  return 0;

}
```

IV način – dinamički niz u C++, operatori new, delete za alokaciju/dealokaciju

```
#include <iostream>
#include <cstdlib>
#include <new>
using namespace std;

void greska()
{
  cerr << "-1\n";
  exit(EXIT_FAILURE);
```

```
}


int main(){

  int i, n;
  int *a;

  //Ucitavamo dimenziju niza
  cin >> n;
  if(n <= 0)
    greska();

  //izbegavamo exception mehanizam, koristimo metod nothrow da analiziramo da li
je alokacija bila uspesna
  a= new (nothrow) int[n]; //alokacija operatorom new
  if (a == nullptr)
    greska();

  //Ucitavamo elemente
  for(i=0; i<n; i++)
  {
     cin >> a[i];
  }


  //Ispisujemo rezultat
  for(i=n-1; i >= 0; i--)
     cout <<a[i] <<" ";
  cout<<"\n";

  delete[] a;  //dealokacija operatorom delete
  return 0;

}
```

V način – metod reverse za vektor

```
#include <iostream>
#include <vector>
#include <algorithm>    // std::reverse
#include <cstdlib>
using namespace std;

void greska()
{
  cerr << "-1\n";
```

```
  exit(EXIT_FAILURE);
}


int main(){

  int i, n;

  //Ucitavamo dimenziju niza
  cin >> n;
  if(n <= 0)
    greska();

  vector <int> a(n);

  //Ucitavamo elemente
  for(i=0; i<n; i++)
  {
     cin >> a[i];
  }

  reverse(a.begin(),a.end());
     //Ispisujemo rezultat
 for (vector<int>::iterator it=a.begin(); it!=a.end(); ++it)
      cout  << *it << ' ';
     cout << '\n';
     //ILI: Ispisujemo rezultat
  for(i=0; i <= a.size()-1; i++)
     cout <<a[i] <<" ";
  cout<<"\n";

  return 0;

}
```