AMA2222 Principles of Programming

Leung Man Kin, Adam
Instructor

adam.leung@polyu.edu.hk
TU720

Chapter 11: Simulation and Application
random number generation,
random shuffling, random simulation,
bisection, linear regression

Random number

Programs like generating a verification code or a gambling game usually involve randomness.

To generate a random number, use the `rand()` function in the `<cstdlib>` library. This function returns a random integer between 0 and RAND_MAX which is a platform-dependent constant. It is at least 32767.

Random integer from 0 to N: `rand() % (N+1)`

Random integer from 1 to N: `rand() % N + 1`

Random float point value from 0 to 1: `rand()/(RAND_MAX+0.0)`

However, when you run a simple program with `rand()` for several times, you will observe that every time it gives you the same number. The `rand()` function's algorithm uses a value called the seed to control how to generate the numbers. By default, the seed value is 1.

To change the seed, use the `srand()` function in the `<cstdlib>` library. To ensure that the seed value is different each time you run the program, use `time(0)` which returns the current time in seconds elapsed since the time 00:00:00 on January 1, 1970 GMT. This function is included in `<ctime>`.

Example program 11.1 Write a program that generates subtraction exercise questions randomly and do the marking.

```
 1 // eg11.1 Generating Random Numbers
 2 #include <iostream>
 3 #include <ctime> // for time function
 4 #include <cstdlib> // for rand and srand functions
 5 using namespace std;
 6 int main()
 7 {
 8          srand(time(0));
 9          int num1 = rand() % 10;
10          int num2 = rand() % 10;
11          if (num1 < num2)
12                    swap(num1, num2);
13          cout << "What is " << num1 << " - " << num2 << "? ";
14          int answer;
15          cin >> answer;
16          if (num1 - num2 == answer)
17                    cout << "You are correct!";
18          else
19                    cout << "Your answer is wrong." << endl << num1
20                    << " - " << num2 << " should be "
21                    << (num1 - num2) << endl;
22          return 0;
23 }
```

```
1  // eg11.1 Generating Random Numbers
2  #include <iostream>
3  #include <ctime> // for time function
4  #include <cstdlib> // for rand and srand functions
5  using namespace std;
6  int main()
7  {
8          srand(time(0));
9          int num1 = rand() % 10;
10         int num2 = rand() % 10;
11         if (num1 < num2)
12                 swap(num1, num2);
13         cout << "What is " << num1 << " - " << num2 << "? ";
14         int answer;
15         cin >> answer;
16         if (num1 - num2 == answer)
17                 cout << "You are correct!";
18         else
19                 cout << "Your answer is wrong." << endl << num1
20                 << " - " << num2 << " should be "
21                 << (num1 - num2) << endl;
22         return 0;
23 }
```

Generate two random single-digit integers

Make sure num1 is greater than or equal to num2 to ensure a non-negative answer. If not, swap them.

Prompt user to input the answer

Check the answer and display the result.

Classwork 11.1

Write a program that simulates a rolling die, then ask users to guess the number of dots.

Samples:

```
Guess the number of dots show up:5
You win!
```

```
Guess the number of dots show up:5
You lose!
It should be 2
```

```cpp
// cw11.1 guess the die
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;
int main()
{
        srand(time(0));
        int die = rand() % 6 + 1;
        cout << "Guess the number of dots shown up: ";
        int guess;
        cin >> guess;
        if (guess == die)
                cout << "You win!";
        else
                cout << "You lose!" << endl << "It should be "
                << die << endl;
        return 0;
}
```

We can further write a game which allow user to make multiple guesses before getting the correct answer. To make the game meaningful, hints are to be provided.

Example program 11.2
A random number from 0 to 99 is generated. The user can guess the number indefinitely many times until the guess is correct. A hint on whether the guess is "too high" or "too low" will be given.

```
Guess a number between 0 to 99
Enter your guess: 50
Your guess is too high

Enter your guess: 20
Your guess is too high

Enter your guess: 12
Yes, the number is 12
```

```
1  // eg11.2 guess a number
2  #include <iostream>
3  #include <cstdlib>
4  #include <ctime>
5  using namespace std;
6  int main()
7  {
8           srand(time(0));
9           int number = rand() % 100;
10          cout << "Guess a number between 0 to 99";
11          int guess = -1;
12          while (guess != number)
13          {
14                    cout << "\nEnter your guess: ";
15                    cin >> guess;
16                    if (guess == number)
17                              cout << "Yes, the number is " << number << endl;
18                    else if (guess > number)
19                              cout << "Your guess is too high" << endl;
20                    else
21                              cout << "Your guess is too low" << endl;
22          }
23          return 0;
24 }
```

As long as the guess is different from the number, the loop will goes on. The initial value of guess is set to -1 as to guarantee it cannot equal to the number. Otherwise the loop will not be started.

In array processing, we can apply the random function to perform two tasks: initialization of an array with random numbers, and random shuffling of an existing array.

Using a for loop, we can access each element in an array and assign it with an random value. Notice that the value of each element are independent.

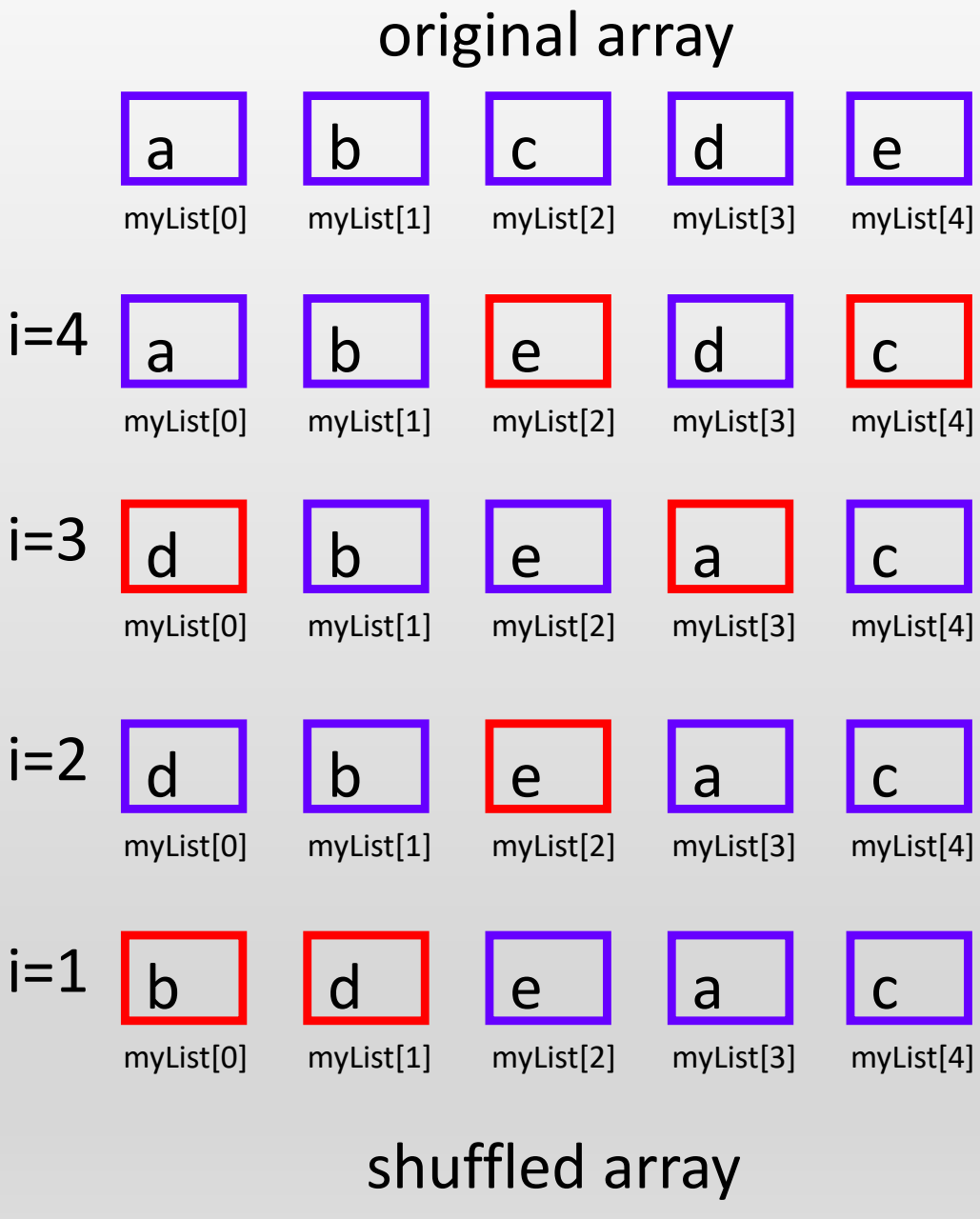
Initializing arrays with random values

```
int myList[arraySize];
for (int i = 0; i < arraySize; i++)
{
        myList[i] = rand() % 100;
}
```

Random shuffling refers to permutation of the elements in an array. Despite the order, the set of elements is the same after random shuffling.

Random shuffling

```
srand(time(0));
for (int i = arraySize -1; i > 0; i--)
{
        int j = rand() % (i + 1);
        swap(mylist[i],myList[j]);
}
```



Example 11.3 Mark Six

Mark Six is a lottery in Hong Kong. 6 balls plus an special ball is drawn from 49 balls labelled 1 to 49. We will simulate this random process by the following:

(i) Assign 1 to 49 as an array of integers.
(ii) Random Shuffle the array.
(iii) Copy the first 6 elements of the shuffled array as the 6 drawn balls.
(iv) Assign the 7-th element of the shuffled array as the drawn special ball.

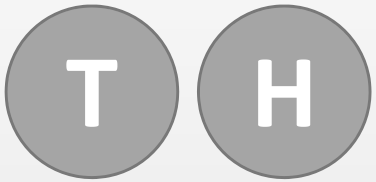
```cpp
// eg11.3 mark six
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;
int main()
{
	srand(time(0));
	int ball[49];
	for (int i=0; i<49; i++)
		ball[i] = i+1;
	for (int i = 48; i > 0; i--)
	{
		int j = rand() % (i + 1);
		swap(ball[i],ball[j]);
	}
	cout << "The drawn numbers are: ";
	for (int i=0; i<6; i++)
		cout << ball[i] << " ";
	cout <<endl << "The special number is: " << ball[6];
	return 0;
}
```

Random simulation

Random simulation refers to modelling a random event. This helps us to study the probability distribution or expected outcomes.

Example 11.4 St Petersburg Paradox

A game is described as follows. The player keep on flipping a fair coin, until the result is a Head. If the Head happens at the i-th trial, the player can win 2^i dollars. Simulate this game.



Win $4

Win $32

Notice that the expected value is infinity. But in reality, the probability of winning more than 2^n dollars is only $\frac{1}{2^n}$ which is very low as n goes up.

$$E(X) = \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i (2^i) = \frac{1}{2}(2) + \frac{1}{4}(4) + \frac{1}{8}(8) + \cdots = \infty$$

```
1  // eg11.4 St Petersurg paradox
2  #include <iostream>
3  #include <ctime>
4  #include <cstdlib>
5  #include <cmath>
6  using namespace std;
7  int main()
8  {
9           srand(time(0));
10          int flip = 0;
11          int result = -1;
12          while (result < 1)
13          {
14                  result = rand()%2;
15                  if (result==0) cout << 'T';
16                  else cout << 'H';
17                  flip++;
18          }
19          cout << endl;
20          cout << "You have flipped " << flip << " times." << endl;
21          cout << "You win " << pow(2,flip) << " dollars." << endl;
22          return 0;
23 }
```

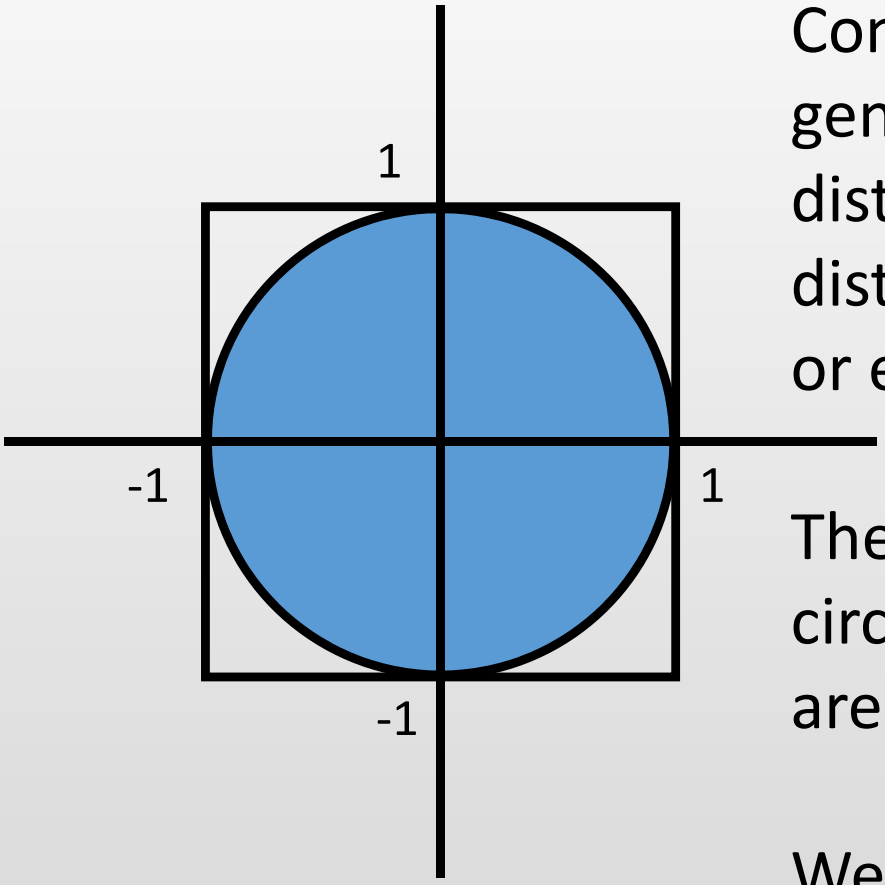
to ensure at least one flipping

the result being 0 or 1 represents "tail" and "head" respectively

Metropolis algorithm is also know as **Monte-Carlo Method** that uses random sampling to simulate situations in physics, finance, statistics, etc. It was published by Nicholas Metropolis in 1953.

In the following example, we will try to approximate the value of π by Monte-Carlo simulation.

The more the sample size, the more accurate result, but also more computation time.



Consider a unit circle. A point is generated with a 2D uniformly distribution on $[-1,1] \times [-1,1]$. If its distance from the origin is less then or equals 1, it lies within the circle.

The probability of a point lying in the circle should follow the ratio of the areas, $\pi : 4$.

We can first calculate the experimental probability, and then multiply this by 4 to get an approximate value of π.

Example program 11.5

```
 1 //eg11.5 approximating pi
 2 #include <iostream>
 3 #include <cstdlib>
 4 #include <ctime>
 5 using namespace std;
 6 int main ()
 7 {
 8          const int n = 100000; //number of trials
 9          int h; // number of hits
10          srand(time(0));
11          for (int i=0; i<n; i++)
12          {
13                   double x = rand() * 2.0 / RAND_MAX -1;
14                   double y = rand() * 2.0 / RAND_MAX -1;
15                   if (x*x + y*y <=1)
16                   h++;
17          }
18          double p = 4.0 * h / n;
19          cout << "The approximated value of pi is " << p;
20          return 0;
21 }
```

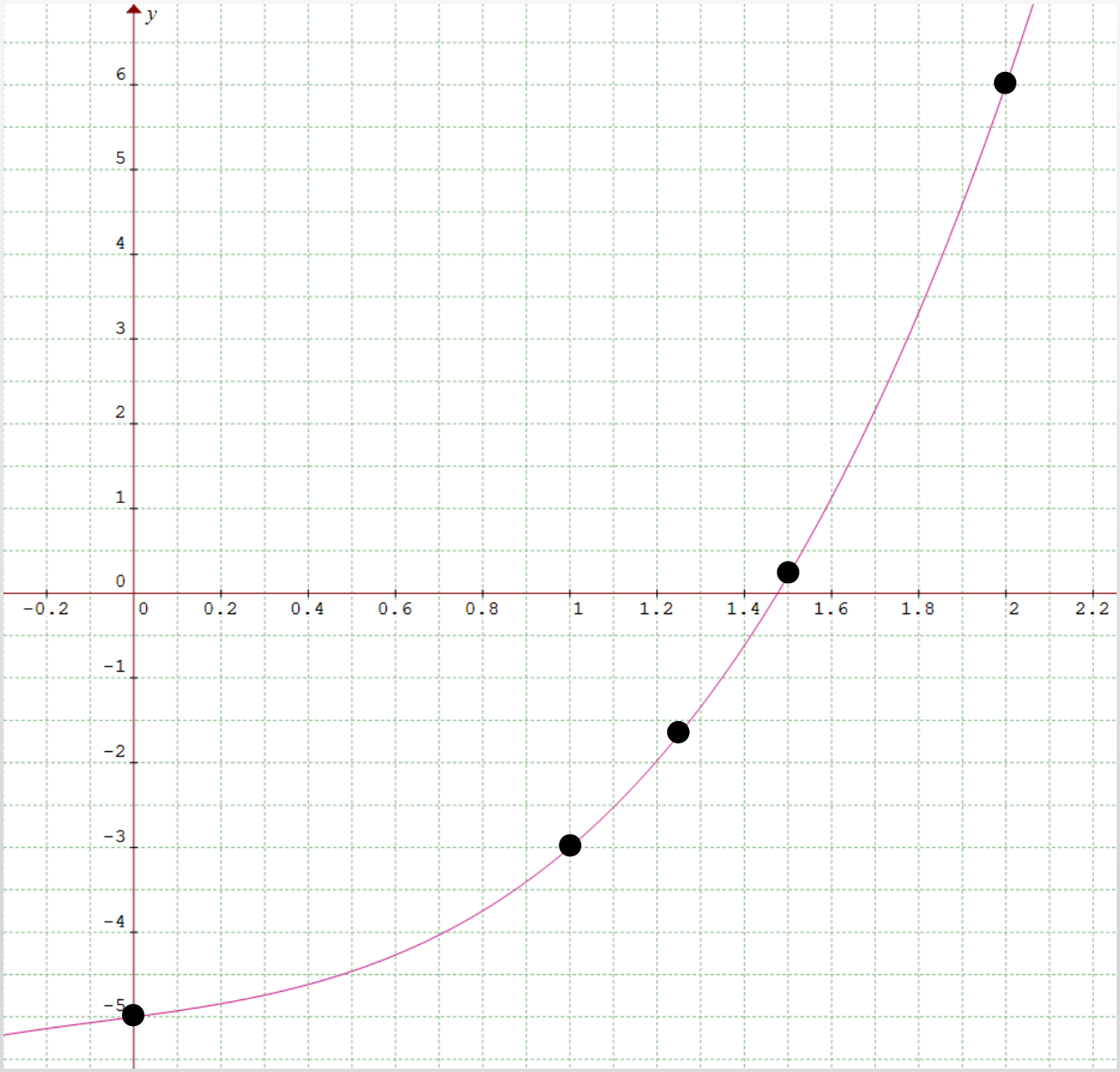
Bisection

In Mathematics, some equations might not have analytical solution. Numerical methods are used to solve such problems with an approximated answer.

Example program 11.6:
Solve the equation $2^x + x^3 = 6$ numerically.

Let $f(x) = 2^x + x^3 - 6$, it is continuous on $\mathbb{R}$.
$f(0) = -5 < 0$ and $f(2) = 6 > 0$

Therefore by Intermediate Value Theorem (IVT) we know that there exist at least a solution in $(0,2)$.



$f(0) < 0, f(2) > 0$

$f(1) < 0, f(2) > 0$

$f(1) < 0, f(1.5) > 0$

$f(1.25) < 0, f(1.5) > 0$

```
1  //eg11.6 Bisection
2  #include <iostream>
3  #include <cmath>
4  using namespace std;
5  double f(double x)
6  {
7           double y;
8           y = pow(2,x)+pow(x,3)-6;
9           return y;
10 }
11 int main()
12 {
13          double a = 0;
14          double b = 2;
15          double c = (a+b)/2;
16          double e = 0.0001;
17          while (abs(f(c)) > e)
18          {
19                   if (f(c)*f(a) < 0) b = c;
20                            else a = c;
21                   c = (a+b)/2;
22          }
23          cout << "The approximated answer is " << c << endl;
24          return 0;
25 }
```

define the single variable real function $f(x)$

Initial interval (a, b) with $f(a)$ and $f(b)$ opposite signs. Let c be their midpoint

Error acceptable of $f(x)$ from 0

While $|f(c)|$ is bigger than the error, do the bisection

Update the interval to either (a, c) or (c, b)

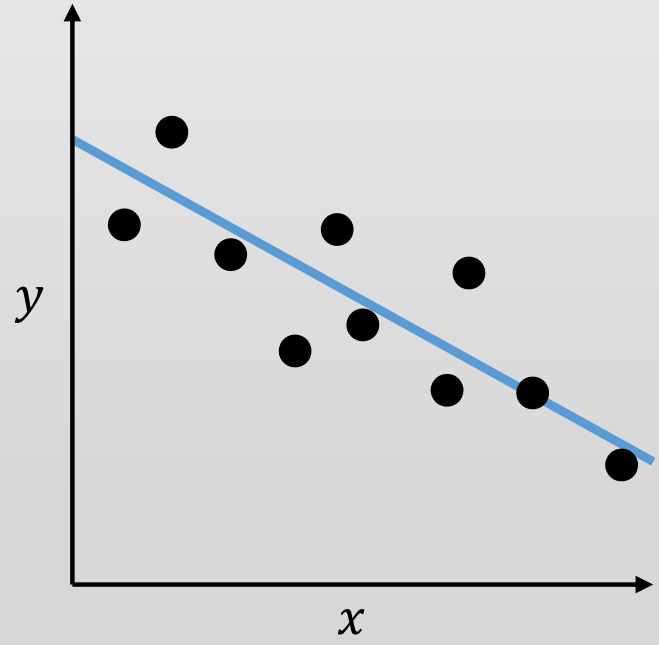
Update the midpoint c

Linear regression

In statistics, linear regression refers to finding a straight line $\hat{y} = \alpha + \beta x$ which best fit into a set of bivariate data $\{(x_i, y_i)\}$. The linear regression model can also be used for making prediction. The formula of its coefficients is is given by:

$$\beta = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2}$$

$$\alpha = \frac{\sum y_i - \beta \sum x_i}{n}$$



Our target is to regard a linear regression model as an object.
Such an object should contain the following:

Data field:

- alpha - y intercept
- beta - slope

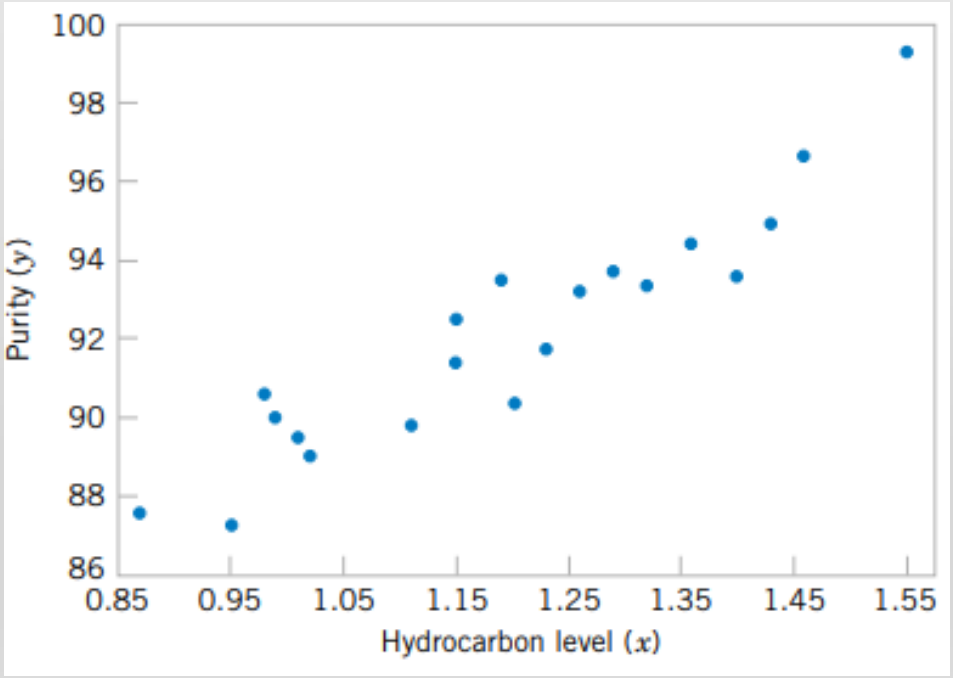
Function:

- default constructor - to create a new object
- predict - make prediction given an input value x
- print - display the linear regression formula
- fit - read two arrays and also the array size, then evaluate

```
 5 class lr
 6 {
 7 public:
 8           double a;
 9           double b;
10           lr(){
11                     a = 0;
12                     b = 0;
13           }
14           double predict(double x){
15                     return a + b * x;
16           }
17           void print()
18           {
19                     cout << "y = " << a << " + " << b << "x" << endl;
20           }
21           void fit(double* x, double* y, int n){
22                     double sx=0, sy=0, sxy=0, sxx=0;
23                     for (int i=0; i<n; i++){
24                               sx += *(x+i);
25                               sy += *(y+i);
26                               sxx += *(x+i) * *(x+i);
27                               sxy += *(x+i) * *(y+i);
28                     }
29                     b = (n*sxy - sx*sy)/(n*sxx - sx*sx);
30                     a = (sy - b*sx)/n;
31           }
32 };
```

read two arrays as pointers

Example 11.7 As shown in the table below, y is the purity of oxygen produced in a chemical distillation process, and x is the percentage of hydrocarbons that are present in the main condenser of the distillation unit. Write a problem to build a linear regression model to study the relationship between these two variables.



Observation Number	Hydrocarbon Level x(%)	Purity y(%)
1	0.99	90.01
2	1.02	89.05
3	1.15	91.43
4	1.29	93.74
5	1.46	96.73
6	1.36	94.45
7	0.87	87.59
8	1.23	91.77
9	1.55	99.42
10	1.40	93.65
11	1.19	93.54
12	1.15	92.52
13	0.98	90.56
14	1.01	89.54
15	1.11	89.85
16	1.20	90.39
17	1.26	93.25
18	1.32	93.41
19	1.43	94.98
20	0.95	87.33

```
 1 // 11.7 linear regression
 2 #include <iostream>
 3 #include <fstream>
 4 using namespace std;
   // insert class lr
33 int main()
34 {
35          ifstream file;
36          file.open("data.txt");
37          double hc[20], ox[20];
38          for (int i=0; i<20; i++)
39                    file >> hc[i] >> ox[i];
40          file.close();
41          lr model1;
42          model1.fit(hc, ox, 20);
43          model1.print();
44          double x, y;
45          cout << "Enter the hydrocarbon level(%): ";
46          cin >> x1;
47          y1 = model1.predict(x1);
48          cout << "Predicted oxygen purity(%): " << y1;
49          return 0;
50 }
```

```
0.99 90.01
1.02 89.05
1.15 91.43
1.29 93.74
1.46 96.73
1.36 94.45
0.87 87.59
1.23 91.77
1.55 99.42
1.40 93.65
1.19 93.54
1.15 92.52
0.98 90.56
1.01 89.54
1.11 89.85
1.20 90.39
1.26 93.25
1.32 93.41
1.43 94.98
0.95 87.33
```

```
y = 74.2833 + 14.9475x
Enter the hydrocarbon level(%): 1.2
Predicted oxygen purity(%): 92.2203
```