Advanced Class Lab02 03/24

1. Monte Carlo Algorithm

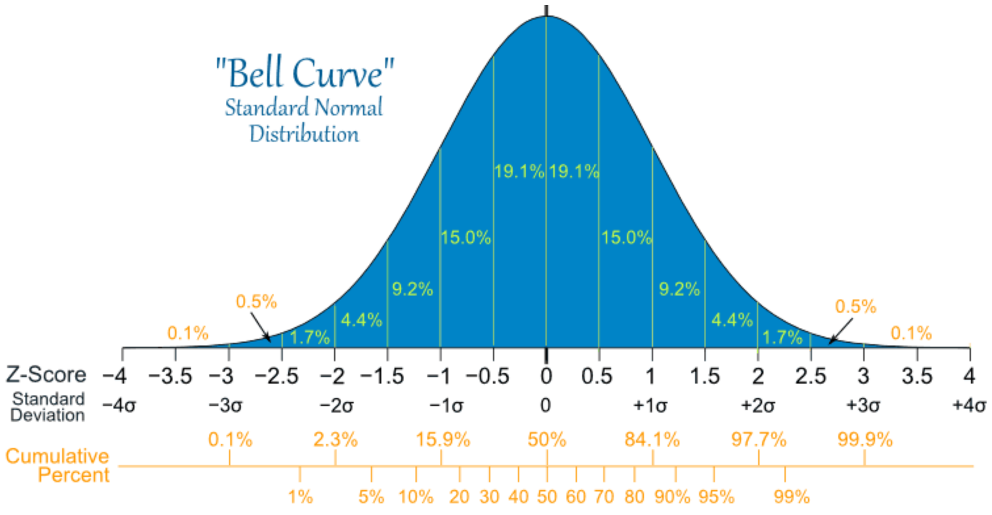
Please calculate area of ellipse $\frac{x^2}{4}+y^2=1$ by Monte Carlo method.

Hint: use random variable to solve this problem.

2. Standard Normal Distribution with Box-Muller Algorithm

Please generate 1000 normally distributed random numbers by Box-Muller method and do statistical analysis.

The equation of normally distribution is $N(m,\sigma^2)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-m)^2}{2\sigma^2}}$, of which m is mean and σ is standard deviation. The following figure is the standard normal distribution.



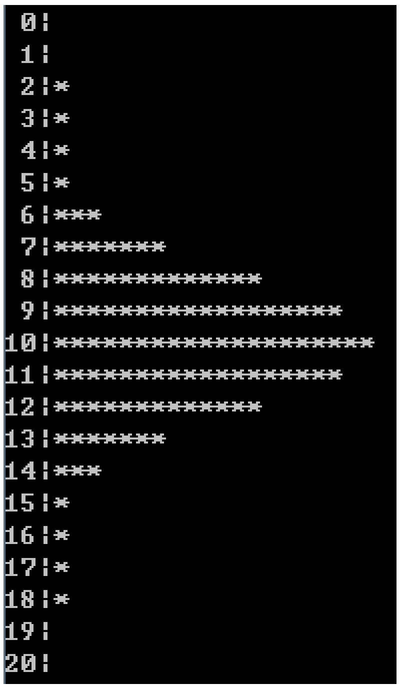
Suppose r_1 and r_2 are two independent random variables that are uniformly distributed, you can use following equations to generate standard normal distribution pairs with mean m and standard deviation σ.

$$x=\sigma\sqrt{-2\log r_1}\cos 2\pi r_2+m$$
$$y=\sigma\sqrt{-2\log r_1}\sin 2\pi r_2+m$$

Choose any of them to implement your normal random number.

You can refer to the following blog which derive the above formulas.
http://peilingliu.pixnet.net/blog/post/16013748

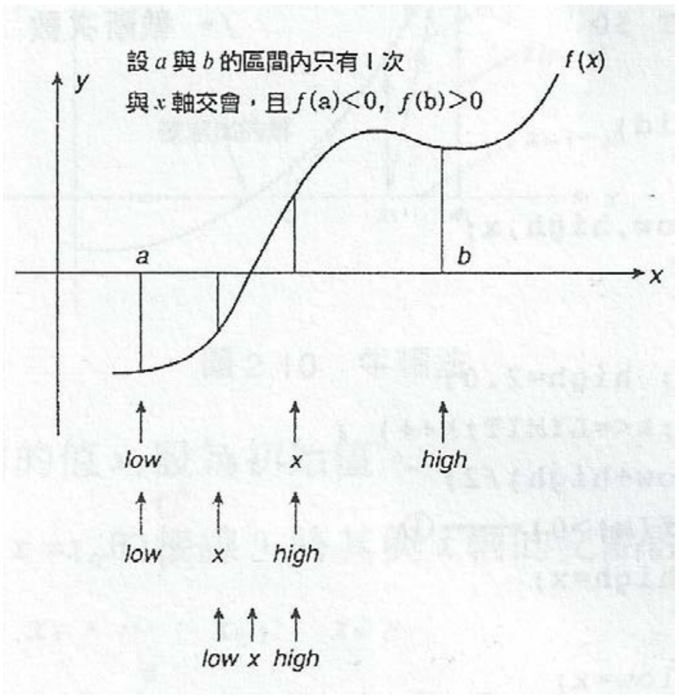
Please generate 100000 Normal random number with mean = 10.0 and sigma = 2.0 and print out “*” for every 1000 samples. The result should be in the following form:



```
 0¦
 1¦
 2¦*
 3¦*
 4¦*
 5¦*
 6¦***
 7¦*******
 8¦*************
 9¦******************
10¦********************
11¦******************
12¦*************
13¦*******
14¦***
15¦*
16¦*
17¦*
18¦*
19¦
20¦
```

3. Solution of Nonlinear Equation with Dichotomy Algorithm

Please calculate the root of equation $f(x) = x^3 - x + 1$ by dichotomy algorithm. Nonlinear equation is defined as that the order of a polynomial is higher than one. The dichotomy algorithm is illustrated as follows.



設 a 與 b 的區間內只有 1 次
與 x 軸交會，且 $f(a)<0$, $f(b)>0$
$f(x)$
y
a
b
x
low
x
high
low
x
high
low x high

1). Set initial values of low and high as any values of two points at left side and right side of root respectively.

2). Calculate central point x_C of low and high by $x_C = (low + high)/2$.

3). If $f(x_C) > 0$, the root is at left side of x_C, make high= x_C and the upper bound will be reduced to half.

If $f(x_C) < 0$, the root is at right side of x_C, make low= x_C and the lower bound will be reduced to half.

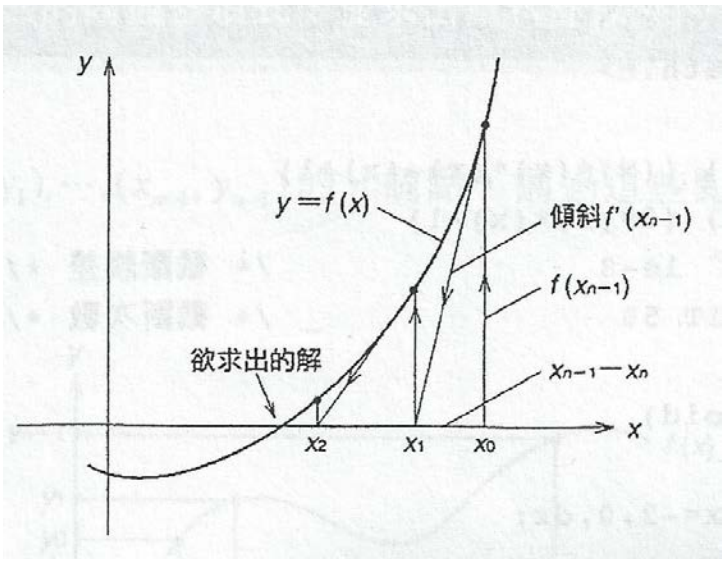
4). If $f(x_C) = 0$ or $|high - low| < EPS$, x_C is answer, or repeat step 2) to step 4).

EPS is cut-off error which you can set by yourself, and will affect accuracy of answer.

In other words, Dichotomy Algorithm divides data into two parts, searches in which part root is, and then reduces search region toward root step by step.

4. Solution of Nonlinear Equation with Newton Algorithm

Please calculate the root of equation $f(x) = x^3 - x + 1$ by Newton algorithm. Nonlinear equation is defined as that the order of a polynomial is higher than one.



The Newton algorithm is illustrated as follows.

1). Set initial value as suitable point x_0 around root.

2). Draw tangents to $y = f(x)$ when $x = x_0$, and set intersection point of tangents and x-axis as x_1. Calculate $x_2, x_3, \ldots, x_{n-1}, x_n$ by the same step.

3). If $|x_n - x_{n-1}| < EPS$, x_n is answer, or repeat step 2) and step 3). EPS is cut-off

error which you can set by yourself.

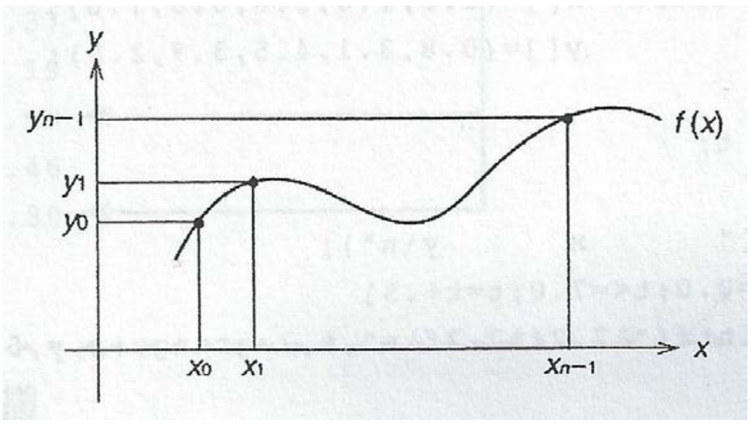
You can use following equation to calculate x_n**.**

$$f'(x_{n-1}) \approx \frac{f(x_{n-1})}{x_{n-1}-x_n} \quad \text{=>} \quad x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

The convergence speed of Newton Algorithm is faster than that of Dichotomy Algorithm.

5. Lagrange Interpolation Polynomial

There are several points data $(0.0,0.8),(1.0,3.1),(3.0,4.5),(6.0,3.9),(7.0,2.8)$**. Please calculate data points which represent polynomial passing above points data by Lagrange interpolation algorithm, and output the values of y when x=0.0, 0.5, 1.0, …, 6.5, 7.0.**



The Lagrange interpolation algorithm is illustrated as follows.

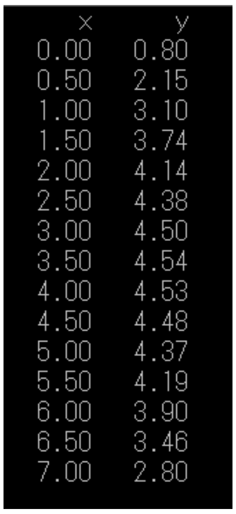
Suppose there are n points such as $(x_0,y_0),\ (x_1,y_1),\ \ldots,\ (x_{n-1},y_{n-1})$**, the polynomial**

$f(x)$ **can be obtained as follows.**

$$\begin{aligned} f(x) &= \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_{n-1})}y_0 \\ &+ \frac{(x-x_0)(x-x_2)\cdots(x-x_{n-1})}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_{n-1})}y_1 \\ &\cdots + \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-2})}{(x_{n-1}-x_0)(x_{n-1}-x_1)\cdots(x_{n-1}-x_{n-1})}y_{n-1} \\ &= \sum_{i=0}^{n-1}\left(\prod_{j=0}^{n-1}\frac{x-x_j}{x_i-x_j}\right)y_i \quad ,i \neq j \end{aligned}$$

Above equation is Lagrange interpolation polynomial of which order is n-1. You can use above equation to calculate other points data.

The result is as follows.



```
   x      y
 0.00   0.80
 0.50   2.15
 1.00   3.10
 1.50   3.74
 2.00   4.14
 2.50   4.38
 3.00   4.50
 3.50   4.54
 4.00   4.53
 4.50   4.48
 5.00   4.37
 5.50   4.19
 6.00   3.90
 6.50   3.46
 7.00   2.80
```

6. **Multiplying Large Number**

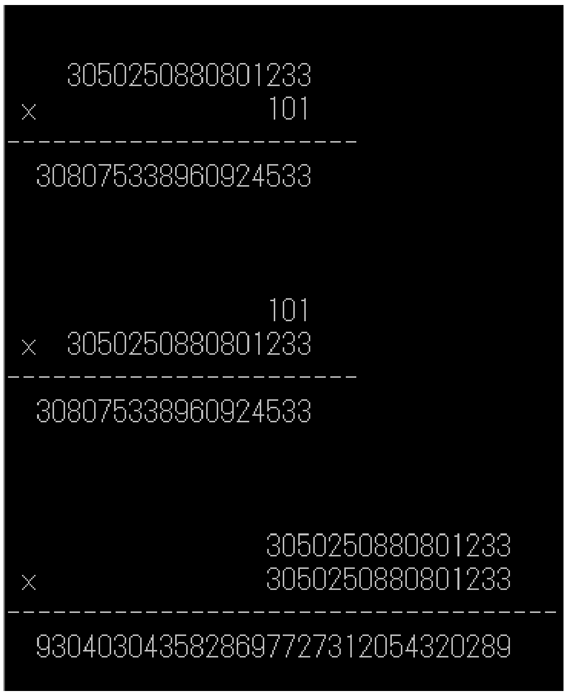
In C++, the largest int value is 2147483647. So an integer larger than this number cannot be stored and processed as an integer. Similarly, if the sum or product of two positive integers is greater than 2147483647, the result will be incorrect. One way to store and manipulate large integers is to store each individual digit of the number in an array.

Write a subroutine char *mul(char *num1, char *num2) that reads two positive integers of at most 40 digits and return the product of the numbers. Please use following code to verify your function.

```
void display(char *num1, char *num2, char *ans)
{
   int i, spa, spa1, length;

   printf("\n\n");

   spa = strlen(ans);
   //spa1 = strlen(num1) + strlen(num2);

   for (i=0; i<spa-strlen(num1); i++)
      printf(" ");
   printf("    %s\n", num1);
   printf(" x");
   for (i=0; i<spa-strlen(num2)-2; i++)
      printf(" ");
   printf("    %s\n", num2);

   for (i=0; i<strlen(ans)+5; i++)
      printf("-");
```

```c
    printf("\n");
    printf("    %s\n\n", ans);
}


int main()
{
    char num1[MAX_N] = "3050250880801233";
    char num2[MAX_N] = "101";
    char *ans;

    ans = mul(num1, num2);
    display(num1, num2, ans);

    ans = mul(num2, num1);
    display(num2, num1, ans);

    ans = mul(num1, num1);
    display(num1, num1, ans);

    getch();
    return 0;
}
```

Result:



```

    3050250880801233
 ×               101
-----------------------
  308075338960924533



                 101
 ×  3050250880801233
-----------------------
  308075338960924533



                 3050250880801233
 ×               3050250880801233
------------------------------------
  9304030435828697727312054320289
```