Repetition Application

Triangle Pattern

Homework 5: Due 10/25 noon

1. Triangle Printing Program:

You can only use

- cout << endl;
- cout << "*";
- cout << " ";
- Enter the size of the pattern, where size is the maximum number of asterisks in a line. For example, size of 5 looks like

Pattern 1a

(1a)

```
*
**
***
****
*****
```

Pattern 1b

(1b)

```
*****
 ****
  ***
   **
    *
```

Pattern 2

Enter the size of a pattern and print out a diamond. For example, the following is a diamond of size 5.

```
  *
 ***
*****
 ***
  *
```

Hint: cut the diamond into two pieces vertically (see red and blue parts). Divide and conquer.

Pattern of asterisks

Print out the following pattern with size 5, where size is the maximum of asterisks in a line.

```
*****
****
***
**
*
```

Pattern of asterisks

```
*****
****
***
**
*
```

(1) What different from one line to the next?

Print 5 asterisks in the first line.

Print 4 asterisks in the second line.

Print 3 asterisks in the third line.

Print 2 asterisks in the fourth line.

Print 1 asterisk in the fifth line.

(2) When do we stop?

Pattern of asterisks

```
*****
****
***
**
*
```

(1) What different from one line to the next?

Print 5 asterisks in the first line.

Print 4 asterisks in the second line.

Print 3 asterisks in the third line.

Print 2 asterisks in the fourth line.

Print 1 asterisk in the fifth line.

Q: What variable(s) are used to trace the above change?

Initialize related variable

```
*****
****
***
**
*
```

```
//initialize the variable.
int numAsterisks = 5;
```

What to do in each row?

Cursor is here

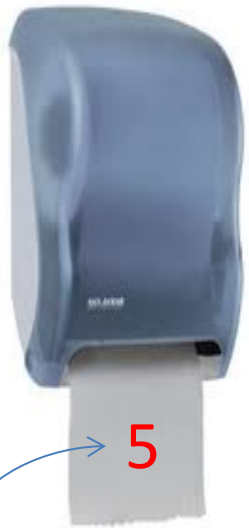
*****

****

***

**

*



int numAsterisks = 5;

Print numAsterisks *s; //use numAsterisks

Print a new line;

How to prepare to move to each row?

Cursor is here

*****

****

***

**

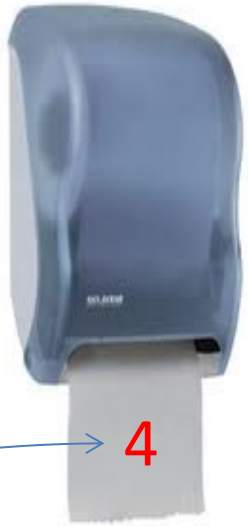
*

int numAsterisks = 5;

Print numAsterisks *s;

Print a new line;

numAsterisks--; //number of *'s to print in coming row



A structure w/o repetition statement

Cursor is here

*****

****

***

**

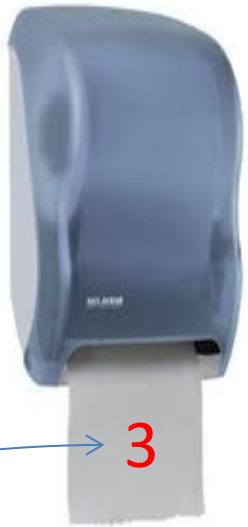
*

int numAsterisks = 5;

Print numAsterisks *s;
Print a new line.
numAsterisks--;

Print numAsterisks *s;
Print a new line;
numAsterisks--;

....



Rewrite in repetition statement

```
*****
****
***
**
*
```

```
int numAsterisks = 5;

while ( numAsterisk > 0 ) {
        Print numAsterisks *s;
        print a new line;
        numAsterisks--;
}
```

How to print numAsterisks asterisk

```
*****
****
***
**
*
```

```
int numAsterisks = 5;

while ( numAsterisk > 0 ) {
        Print numAsterisks *s;
        print a new line.
        numAsterisks--;
}
```

How to print numAsterisks asterisk

~~Print numAsterisks asterisks;~~

Say it in Java!

Related: print 5 asterisks using a repetition statement.

```
int i = 0;
while ( i < 5) {
        cout << "*";
        i++;
}
```

How to print numAsterisks asterisk

Print numAsterisks asterisk;

Related: print 6 asterisks using a repetition statement.

```
int i = 0;
while ( i < 6) {
        cout << "*";
         i++;
}
```

How to print numAsterisks asterisk

Print numAsterisks asterisk;

```
int i = 0;
while ( i < numAsterisks) {
        cout << "*";
        i++;
}
```

Pattern of asterisks

```
*****
****
***
**
*
```

```
int numAsterisks = 5;

while ( numAsterisk > 0) {
        Print numAsterisks asterisk;
        print a new line;
        numAsterisks--;
}
```

```
int i = 0;
while ( i < numAsterisks) {
        cout << "*";
        i++;
}
```

Pattern of asterisks

```
*****
****
***
**
*
```

```
int numAsterisks = 5;

while ( numAsterisk > 0) {
        Print numAsterisks asterisk;
        print a new line.
        numAsterisks--;
}
```

```
int i = 0;
while ( i < numAsterisks) {
        cout << "*";
        i++;
}
```

```
cout << endl;
```

Generalize to any size

```
*****
****
***
**
*
```

```
cout << ”Enter an int: ";
int size;
cin >> size;
```

size

```
int numAsterisks = 5;

while ( numAsterisk > 0) {
        Print numAsterisks asterisk;
        print a new line;
        numAsterisks--;
}
```

```
int i = 0;
while ( i < numAsterisks) {
        cout << "*";
        i++;
}
```

```
cout << endl;
```

Warning: size ≠ numAsterisks

- size is the maximum number of asterisks in all rows. It will not change from row to row.
- numAsterisks is the number of asterisks in each row. It will change from row to row.
- We may use size to initialize numAsterisks or in condition deciding whether we finish drawing the pattern.

```
*****
****
***
**
*
```

In this pattern, size = 5.

numAsterisks changes from 5 to 4 to 3 to 2 and 1, depending on which row we are talking.
So numAsterisks = inp.nextInt(); might not work for all problems, which does not store size.

Triangle of asterisks vs. sum of series

```
//Goal: Find the sum of 5 + 4 + 3 + 2 + 1.
int value = 5;
int sum = 0;
while (value > 0) {
        //add value to sum.
        sum += value;
        value --;
}
```

```
int numAsterisks = 5;
while (numAsterisks > 0) {
        //print numAsterisks *s

        //print a new line
        cout << endl;
        numAsterisks--;
}
```

```
int i = 0;
while ( i < numAsterisks ) {
        cout << "*";
        i++;
}
```

Can we move this line inside the above inner loop? Big NO NO!!!

Triangle of asterisks: nested-loop

```
int numAsterisks = 5;
while (numAsterisks > 0) {
        //print numAsterisks *s

        int i = 0;
        while ( i < numAsterisks ) {
                cout << "*";
                i++;
        }

        //print a new line
        cout << endl;

        numAsterisks--;
}
```

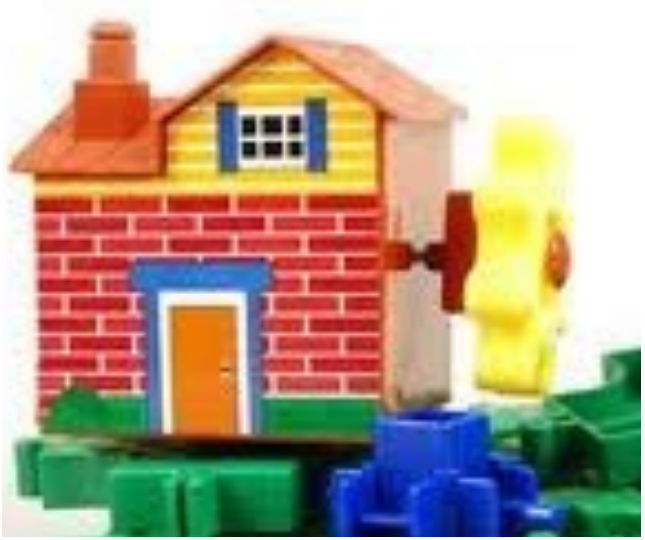
Building block: do something for x times

Print out * for numAsterisks times

```
int i = 0;
while ( i < numAsterisks ) {
        cout << "*";
        i++;
}
```

do something for x times

```
int i = 0;
while ( i < x ) {
        do something;
        i++;
}
```



Method to work pattern of asterisks

(1) What **changes** happen from one row to the next?

(2) What **variables** to describe those changes?

(3) What are **initial values** of those variables?

(4) What do we do in each row, especially, how to **use** those variables in each row?

(5) How to **update** those variables to prepare for the next row?

(6) When shall we **stop**?

What changes happen from one row to the next?

```
*
**
***
****
*****
```

Number of preceding spaces (spaces before the first *) is different from row to row.

```
*****
****
***
**
*
```

Number of preceding space is always 0. So no need to use it.

Use variables to describe them.

Print out **some** preceding spaces;

Print out **several** asterisks;

Print out one new line.

What variables to describe those changes?

```
    *
   **
  ***
 ****
*****
```

numPrecSpaces: number of preceding spaces

numAsterisks: number of asterisks

What are the initial values of the variables?

```
    *
   **
  ***
 ****
*****
```

numPrecSpaces = 4;

numAsterisks = 1;

How do the variables changes?

```
    *
   **
  ***
 ****
*****
```

numPrecSpaces	numAsterisks
4	1
3	2
2	3
1	4
0	5

What do we do in each row?

```
    *
   **
  ***
 ****
*****
```

What do we do each row?

Print numPrecSpaces spaces;
Print numAsterisks asterisks;
Print a new line.

precSpaces	numAsterisks
4	1
3	2
2	3
1	4
0	5

Update variables for next row

```
    *
   **
  ***
 ****
*****
```

numPrecSpaces	numAsterisks
4	1
3	2
2	3
1	4
0	5

numPrecSpaces--;
numAsterisks++;

When to stop?

```
    *
   **
  ***
 ****
*****
```

numPrecSpaces >= 0

Use **either** one of these two conditions. Variables numPrecSpaces and numAsterisks are not independent: one can be calculated from the other.

numAsterisks <= 5

numPrecSpaces	numAsterisks
4	1
3	2
2	3
1	4
0	5

Pattern of asterisks

Print out the following pattern with a given size, where size is the maximum of asterisks in a line.

The following is of size of 5.

```
    *
   **
  ***
 ****
*****
```

Pattern of asterisks

```
   *
  **
 ***
****
*****
```

```
//initialization
int numPrecSpaces = 4;
int numAsterisks = 1;
while (numPrecSpaces >= 0) {
        //Use variables for current row.
        print numPrecSpaces spaces;
        print numAsterisks *s;
        print out a new line;
        //prepare for the new row
        numPrecSpaces--;
        numAsterisks++;
}
```

As long as something needs to be done more than one time, use while-statement.

Building block: do something for x times

Print out * for numAsterisks times

```
int i = 0;
while ( i < numAsterisks ) {
        cout << "*";
         i++;
}
```

do something for x times

```
int i = 0;
while ( i < x ) {
        do something;
        i++;
}
```

Pattern of asterisks

```
   *
  **
 ***
****
*****
```

```
//initialization
int numPrecSpaces = 4;
int numAsterisks = 1;
while ( numPrecSpaces >= 0 ) {
        //Use variables for current row.
        print numPrecSpaces spaces;
        print numAsterisks *s;
        print out a new line;
        //prepare for the new row
        numPrecSpaces--;
        numAsterisks++;
}
```

```
int i = 0;
while ( i < numPrecSpaces ) {
        cout << " ";
        i++;
}
```

```
int i = 0; // i declared; need init.
while ( i < numAsterisks ) {
        cout << "*";
        i++;
}
```

Pattern of asterisks

```
   *
  **
 ***
****
*****
```

```
//initialization
int numPrecSpaces = 4;
int numAsterisks = 1;
while (numPrecSpaces >= 0 ) {
        //Use variables for current row.
        print numPrecSpaces spaces;
        print numAsterisks *s;
        cout << endl;
        //prepare for the new row
        numPrecSpaces--;
        numAsterisks++;
}
```

```
int i = 0;
while ( i < numPrecSpaces ) {
        cout << " ";
        i++;
}
```

```
int i = 0; // i declared; need init.
while ( i < numAsterisks ) {
        cout << "*";
        i++;
}
```

Pattern of asterisks: generalize to any size

```
    *
   **
  ***
 ****
*****
```

```
cout << "Enter size of a triangle: ";
int size;
cin >> size;
//initialization
int numPrecSpaces = 4;
int numAsterisks = 1;
while (numPrecSpaces >= 0 ) {
        //Use variables for current row.
        …
        cout << endl;
        //prepare for the new row
        numPrecSpaces--;
        numAsterisks++;
}
```

size-1

```
int i = 0;
while ( i < numPrecSpaces ) {
        cout << " ";
        i++;
}
```

```
int i = 0; //i declared; need init.
while ( i < numAsterisks ) {
        cout << "*";
        i++;
}
```

Optional: simplify: numPrecSpaces & numAsterisks

numPrecSpaces & numAsterisks are dependent:
numPrecSpaces + numAsterisks = size, or
numAsterisks = size – numPrecSpaces or
numPrecSpaces = size – numAsterisks

```
    *
   **
  ***
 ****
*****
```

```
//TODO: Enter an int size
//initialization
int numPrecSpaces = size -1;
int numAsterisks = 1;
while ( numPrecSpaces >= 0 ) {
        //Use variables for current row.

        cout << endl;
        //prepare for the new row
        numPrecSpaces--;
        numAsterisks++;
}
```

```
int i = 0;
while ( i < numPrecSpaces ) {
        cout << " ";
        i++;
}
```

```
i = 0; // i declared; need init.
while ( i < numAsterisks ) {
        cout << "*";
        i++;
}
```

Optional: simplify: numPrecSpaces & numAsterisks

```
    *
   **
  ***
 ****
*****
```

Without loss of generality, get rid of numAsterisks by
numAsterisks = size – numPrecSpaces

```
//TODO: enter size
//initialization
int numPrecSpaces = size -1;
int numAsterisks = 1;
while ( numPrecSpaces >= 0 ) {
        //Use variables for current row.

        cout << endl;
        //prepare for the new row
        numPrecSpaces--;
        numAsterisks++;
}
```

```
int i = 0;
while ( i < numPrecSpaces ) {
        cout << " ";
        i++;
}
```

```
i = 0; // i declared; need init.
while ( i < numAsterisks ) {
        cout << "*";
        i++;
}
```

Optional: simplify: numPrecSpaces & numAsterisks

```
    *
   **
  ***
 ****
*****
```

Without loss of generality, get rid of numAsterisks by
numAsterisks = size – numPrecSpaces

```
//TODO: enter size
//initialization
int numPrecSpaces = size -1;
int numAsterisks = 1;
while ( numPrecSpaces >= 0 ) {
        //Use variables for current row.

        cout << endl;
        //prepare for the new row
        numPrecSpaces--;
        numAsterisks++;
}
```

```
int i = 0;
while ( i < numPrecSpaces ) {
        cout << " ";
        i++;
}
```

```
i = 0; // i declared; need init.
while ( i < numAsterisks ) {
        cout << "*";
        i++;
}
```

Pattern of asterisks: simplify

```
    *
   **
  ***
 ****
*****
```

```
//TODO: enter size
//initialization
int numPrecSpaces = size -1;
while ( numPrecSpaces >= 0 ) {
        //Use variables for current row.

        cout << endl;
        //prepare for the new row
        numPrecSpaces--;
}
```

```
int i = 0;
while ( i < numPrecSpaces ) {
        cout << " ";
        i++;
}
```

```
i = 0; // i was declared; need init.
while ( i < size – numPrecSpaces )
{
        cout << "*";
        i++;
}
```

Hint for diamond shape

Enter the size of a pattern and print out a diamond. For example, the following is a diamond of size 5.

```
  *
 ***
*****
 ***
  *
```

Hint: cut the diamond into two pieces vertically (see red and blue parts). Divide and conquer.

Hint for diamond shape: II

```
  *
 ***
*****
```

numPreSpaces: preceding of each row

numAsterisks: asterisks of each row

Initial values of related variables

```
  *
 ***
*****
```

Enter size from keyboard.

int numAsterisks = 1;

int numPreSpaces = ?; //not so obvious

If size = 5, then numPreSpaces = 2.

If size = 7, then numPreSpaces = 3.

So, numPreSpaces depends on size.

Note: numPreSpaces + numAsterisks + numSuccSpaces = size.

while numSuccSpaces = numPreSpaces, by symmetry

numPreSpaces is (size-numAsterisks)/2 and is **initialized** as (size-1)/2.

Use variables for current row

```
  *
 ***
*****
```

//Print numPreSpaces spaces;

… // you fill in the code

//Print numAsterisks *s;

… // you fill in the code

Do not forget to print a new line to indicate the end of current row.

Update variables when moving to next row

```
  *
 ***
*****
```

```
numAsterisks += 2;
numPreSpaces --;
```

When to stop?

```
  *
 ***
*****
```

Stop when numAsterisks > size.

Equivalently, stop when numPreSpaces < 0.

while (numAsterisks <= size) {

1. Draw pattern **on current row**.
2. Prepare variables **for next row**.

}

Initialize variable for bottom diamond

size = 5:

```
  *
 ***
*****
 ***
  *
```

size = 7:

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

numAsterisks = size -2;

numPreSpaces = 1;

Toss coin for 1000 times

Throw a coin for 1000 times, report how many times we get heads and how many times we get tails.

- Hint: We can generate a random number with values 0 and 1 to simulate the throwing of coins, where 0 is the head and 1 is the tail.

Toss coin for 1000 times

Throw a coin for 1000 times, report how many times we get heads and how many times we get tails.

```
int numTosses = 0;
while (numTosses < 1000) {
        //What to do in each round?
        toss a coin;
        tally the result;

        //number of tosses tallied so far
        numTosses ++;
}
```

Toss coin for 1000 times

Throw a coin for 1000 times, report how many times we get head and how many times do we get tail.

```
//rand generates a stream of pseudorandom numbers.
Random rand = new Random();
int numTosses = 0;
while (numTosses < 1000) {
        //What to do in each round?
        //toss a coin
        tossResult = ?; //generate a random int 0 or 1

        tally the result;

        //number of tosses tallied so far
       numTosses ++;
}
```

Toss a coin can be simulated as generating two random numbers: 0 and 1.

Toss coin for 1000 times

Throw a coin for 1000 times, report how many times we get head and how many times do we get tail.

```
int numHeads = 0;
int numTails = 0;
//rand generates a stream of pseudorandom numbers.
Random rand = new Random();
int numTosses = 0;
while (numTosses < 1000) {
        //toss a coin
        tossResult = ?;
        //tally the result;
        if (tossResult == 0)
          numHeads ++;
        else numTails++;
        //number of tosses tallied so far
       numTosses ++;
}
```

Tally the coin toss result.