CISS245: Advanced Programming
Assignment 1 (Review)

Objectives:

1. Review of CISS240.

The format of your program must look like this:

```
// Name: smaug
// File: a01q01.cpp

#include <iostream>

int main()
{
    *** YOUR WORK HERE ***
    return 0;
}
```

replacing "John Doe" with your name. In particular:

1. You must have your name and the name of the file at the top of each C++ source file as shown above.
2. The last thing printed must be a newline.

Read the questions carefully before diving in.

Note that you should create a new project for each question. For easy maintenance of your assignments, I suggest you have a folder `ciss245` somewhere in your `My Documents`, and in that you have a folder `a`, and in folder a you have a folder `a01`, and you have solutions folders `a01q01`, `a01q02`, etc. in the folder `a01`:

```
.
.
.
ciss245
|
+- a
   |
```

```
+- a01
   |
   +- a01q01
   |
   +- a01q02
```

All the relevant files (cpp and header files) for question 1 must be in folder `a01q01`. Etc.

Some test cases are included in the problems which are shown in a framed box after the statement of the problem. User input is underlined. Here's an example (see Q1):

```
1
+--------+
|        |
|  +--+  |
|  |     |
|  +-----+
```

You are strongly advised to add more testing on your own.

Q1. [Spiral] ASCII art. (Do not use arrays.)

The test cases explains what you need to do. Make sure the drawing is done by a function called with the following prototype:

```
void draw_spiral(int);
```

In other words, the skeleton code is

```
#include <iostream>

void draw_spiral(int n)
{
}

int main()
{
    int n;
    std::cin >> n;
    draw_spiral(n);

    return 0;
}
```

Test 1

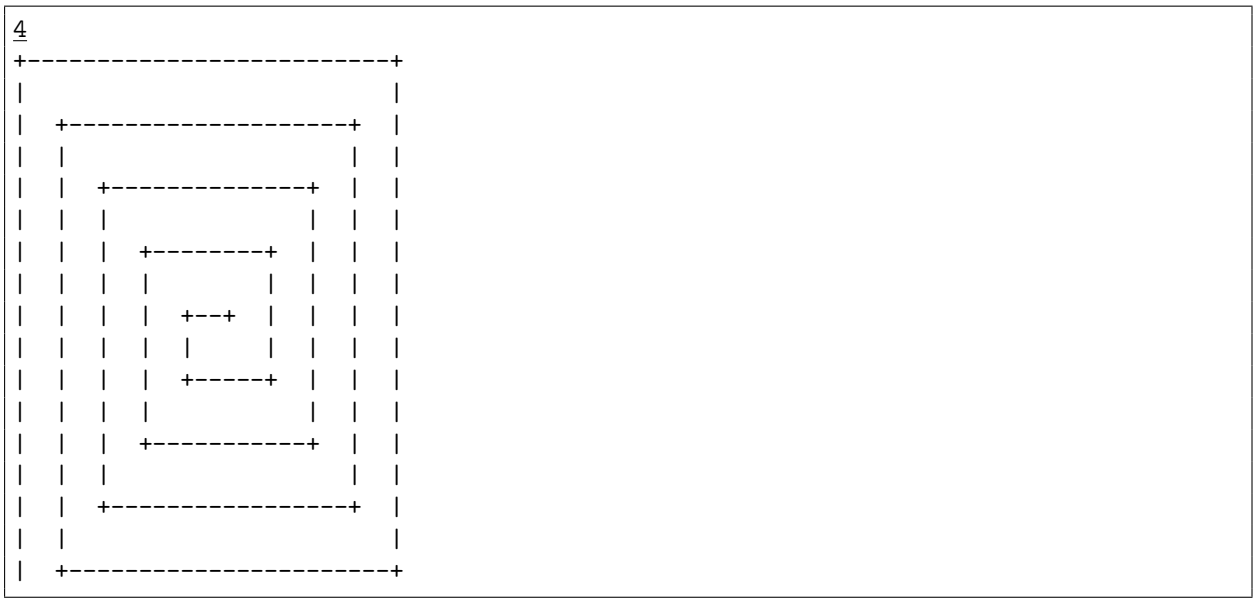
```
1
+--------+
|        |
|  +--+  |
|  |     |
|  +-----+
```

Test 2

```
2
+--------------+
|              |
|  +--------+  |
|  |        |  |
|  |  +--+  |  |
|  |  |     |  |
|  |  +-----+  |
|  |           |
|  +-----------+
```

Test 3

Test 4



WARNING: ... INCOMING SPOILERS ... Hints on next page. Use only if necessary.

Hints

Look at the case when the input is 4:

```
4
+--------------------------+
|                          |
|  +--------------------+  |
|  |                    |  |
|  |  +--------------+  |  |
|  |  |              |  |  |
|  |  |  +--------+  |  |  |
|  |  |  |        |  |  |  |
|  |  |  |  +--+  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  +-----+  |  |  |
|  |  |  |           |  |  |
|  |  |  +-----------+  |  |
|  |  |                 |  |
|  |  +-----------------+  |
|  |                       |
|  +-----------------------+
```

Notice that the first line of output is "more similar" to the third and fifth than the second and fourth. So if you look at the 1-st, 3-rd, 5-th, ... lines of output the picture looks like this:

```
4
+--------------------------+
|  +--------------------+  |
|  |  +--------------+  |  |
|  |  |  +--------+  |  |  |
|  |  |  |  +--+  |  |  |  |
|  |  |  |  +-----+  |  |  |
|  |  |  +-----------+  |  |
|  |  +-----------------+  |
|  +-----------------------+
```

In other words the pseudocode should look like this:

```
for i = 1, 2, 3, 4, ..., 17:
    if i is odd:
        draw it in a certain way
    else:
        draw it in another way
```

This is similar to the problem of drawing alternativing characters, something like

```
5
*
@
*
@
*
```

Going back to our problem, I suggest you focus on the odd case first, i.e.,:

```
for i = 1, 2, 3, 4, ..., 17:
    if i is odd:
        draw it in a certain way
```

Of course later you have to make the program work for any input n. When the input is `4`, the number of lines printed is `17`. You have to experiment to see what happens when the input is 1, 2, 3, 5. You should be able to figure out a formula for the number of lines printed in terms of the user input n (look at the `??` below):

```
for i = 1, 2, 3, 4, ..., ??:
    if i is odd:
        draw it in a certain way
```

If you look at Test 1, Test 2, Test 3, Test 4, you'll see that

- when $n = 1$, the number of lines printed is 5.
- when $n = 2$, the number of lines printed is 9.
- when $n = 3$, the number of lines printed is 13.
- when $n = 4$, the number of lines printed is 17.

So what is `??` in terms of n?

The output lines of the top half are similar but slightly different from the bottom half. The top half looks like this:

```
4
+-------------------------+
|  +--------------------+  |
|  |  +--------------+  |  |
|  |  |  +--------+  |  |  |
|  |  |  |  +--+  |  |  |  |
```

and the bottom half looks like this:

```
4
|  |  |  |  +-----+  |  |  |
|  |  |  +-----------+  |  |
|  |  +-----------------+  |
|  +-----------------------+
```

So the pseudocode should look like this:

```
for i = 1, 2, 3, 4, ..., ??:
    if i is odd:
        if i < 10:
            draw it in a certain way (for top half)
        else:
            draw it in a certain way (for bottom half)
```

I suggest you focus on the top half first, i.e.,

```
for i = 1, 2, 3, 4, ..., ??:
    if i is odd:
        if i < 10:
            draw it in a certain way (for top half)
```

Here's the output again for the top half when `i` is odd – I've included the value of `i` on the left:

```
i
1  +-------------------------+
3  |  +--------------------+  |
5  |  |  +--------------+  |  |
7  |  |  |  +--------+  |  |  |
9  |  |  |  |  +--+  |  |  |  |
```

You see that for each value of `i` in the above, you have to

- print a bunch of `"|  "`,
- followed by `'+'`,
- followed by a bunch of `'-'`,
- followed by `'+'`,
- followed by a bunch of `"  |"`.

So the pseudocode now becomes:

```
for i = 1, 2, 3, 4, ..., ??:
    if i is odd:
        if i < 10:
            draw a bunch of "|  "
            draw '+'
            draw a bunch of '-'
            draw '+'
            draw a bunch of "  |"
            draw newline
```

In fact it's easy to count the number of times to draw the things in the bunches:

```
i
1  +-------------------------+    0"|  ", 1'+', 26'-', 1'+', 0"  |", 1'\n'
3  |  +--------------------+  |    1"|  ", 1'+', 20'-', 1'+', 1"  1", 1'\n'
5  |  |  +--------------+  |  |    2"|  ", 1'+', 14'-', 1'+', 2"  1", 1'\n'
7  |  |  |  +--------+  |  |  |    3"|  ", 1'+',  8'-', 1'+', 3"  1", 1'\n'
9  |  |  |  |  +--+  |  |  |  |    4"|  ", 1'+',  2'-', 1'+', 4"  1", 1'\n'
```

You now have to relate the values of `i` = 1,3,5,7,9 to the number of `"|  "` to draw:

```
i
1  +-------------------------+    0"|  "
3  |  +--------------------+  |    1"|  "
5  |  |  +--------------+  |  |    2"|  "
7  |  |  |  +--------+  |  |  |    3"|  "
9  |  |  |  |  +--+  |  |  |  |    4"|  "
```

Do you see that: (9 - 1)/2 = 4, (7 - 1)/2 = 3, etc.

```
for i = 1, 2, 3, 4, ..., ??:
    if i is odd:
        if i < 10:
            draw (i - 1)/2 "|  "
            draw '+'
            draw a bunch of '-'
            draw '+'
            draw a bunch of "  |"
```

In other words

```
for i = 1, 2, 3, 4, ..., ??:
    if i is odd:
        if i < 10:
            for k = 1, 2, 3, ..., (i - 1)/2:
                draw "|  "
            draw '+'
            draw a bunch of '-'
            draw '+'
            draw a bunch of "  |"
```

The above should get you going.

Q2. [Rising hills] ASCII art. (Do not use arrays.)

The test cases explains what you need to do. Make sure the drawing is done by a function with the following prototype:

```
void draw_rising_hills(int);
```

Test 1

```
1
*
```

Test 2

```
2
  *
****
```

Test 3

```
3
      *
  *  ***
*********
```

Test 4

```
4
           *
      *    ***
  *  ***  *****
***************
```

Test 5

```
5
                  *
           *      ***
      *    ***    *****
  *  ***  *****  *******
************************
```

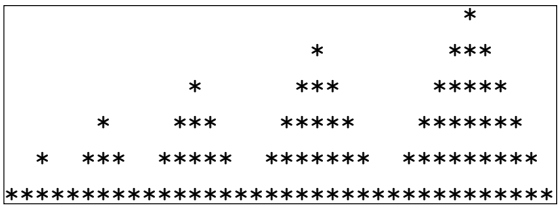
Test 6

```
6
                                *
                    *          ***
              *       ***       *****
       *     ***     *****     *******
   *   ***   *****   *******   *********
***********************************
```

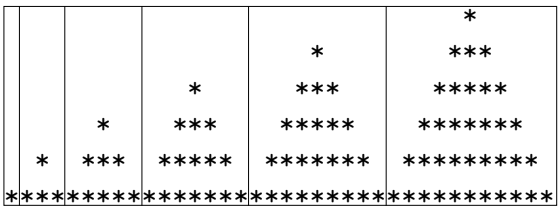
WARNING: ... INCOMING SPOILERS ... Hints on the next page.

Hints

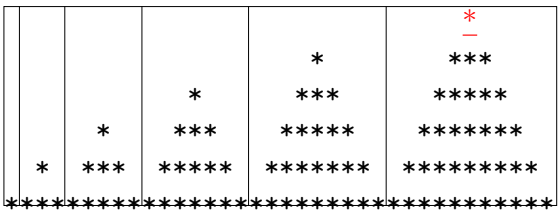
Look at the test case when the input is $n = 6$:



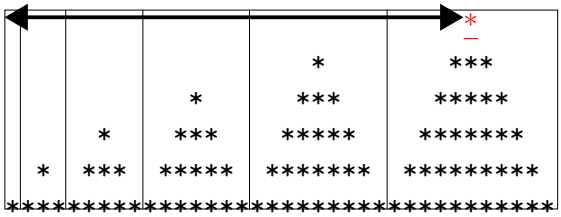
You might want to think of this:



Why is this helpful? Because if you want to draw the first star (at the first line of output):



you would need to print a bunch of spaces:

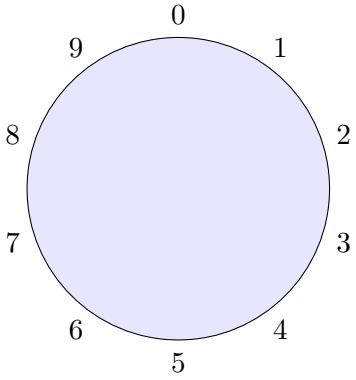


How many spaces? For the case when the user input is $n = 6$, the number of spaces is the base of the rectangles containing the hills before the largest hill #6, plus roughly half of the base of hill #6. Right? The first hill has base of length 1, the second hill has base of length 3, the third hill has base of length 5, etc.

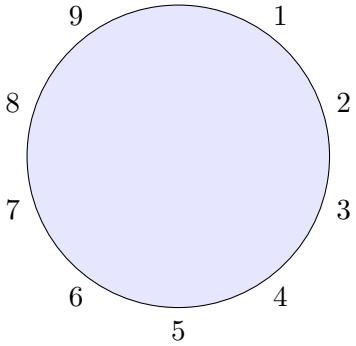
For the printing of the second line of output, you will also need to use roughly the same idea when

you print the top of the second-to-last hill.

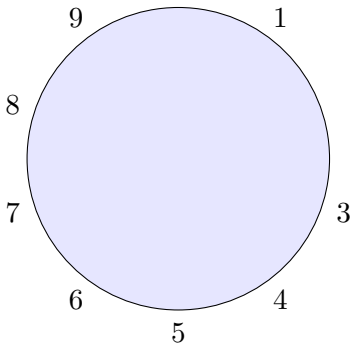
Q3. A group of 10 rebels were trapped in a cave. Very soon an army of invading soldiers will be upon them. Instead of surrendering themselves, they decided to sit in a circle, and starting with someone and going around the circle, every other person will take his own life. Here's the initial setup.



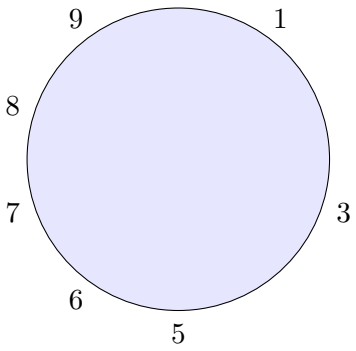
First the rebel at 0 will take his own life:



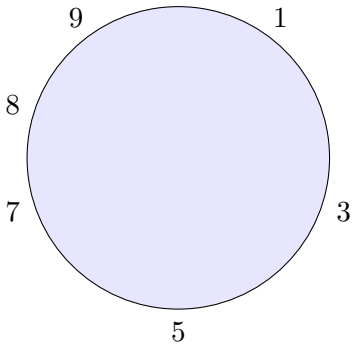
The next to take his life is the person at 2, i.e., going clockwise, it's the second person alive after the person at 0 who just died:



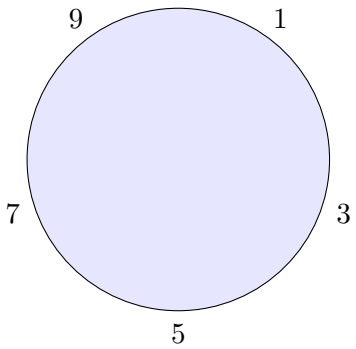
This is followed by 4:



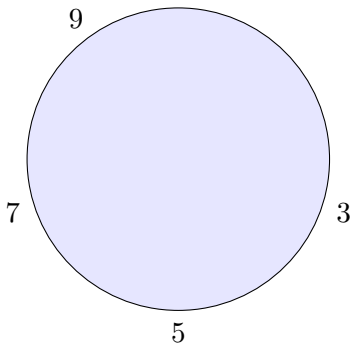
and then 6:



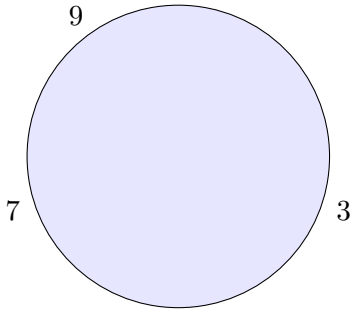
Next is 8:



Now note that the among the remaining rebels, the second to the left of where 8 was is the rebel at position 1. Therefore the next to go is 1:



After 1, the next to be removed is 5:



then 9, followed by 7. Note that 3 is the last rebel.

In general given n rebels labeled 0 to $n-1$, with the same algorithm for removing the rebels, who is the last surviving rebel? In the case of $n = 10$, the last surviving rebel is rebel number 3.

Write a function with prototype

```
int last_alive(int n);
```

that returns the index of the last surviving rebel. For instance, `last_alive(10)` returns `3`. Your program must allow a maximum number of 1000 rebels.

Test 1

```
10
3
```

Test 2

```
5
1
```

WARNING: INCOMING SPOILER ... Hint is on the next page. Use only if necessary.

Hint

For Test 1, think about using an array of 10 boolean values. Say the array is called `alive`. Set all 10 values of `alive` to `true`. Go through the array just like going through the circle. Say you use an index variable `i`. Once `i` goes beyond `9`, i.e., when `i` reaches 10, you want to set it to `0`. In other words the values taken by `i` is `0`, `1`, `2`, `3`,... `7`, `8`, `9`, `0`, `1`, `2`, `3`,... `7`, `8`, `9`, `0`, `1`, `2`, etc.

Starting with `i = 0`, set `alive[i]` to `false`. You then advance `i` to the next `alive[i]` that is `true`, and you keep going until you reach the *second* `alive[i]` value that is `true`, you stop and set that `alive[i]` to `false`. Repeat. You stop when the *second* `alive[i]` that is `true` is where you started. That tells you that there's only one alive.

The above only works for an array of 10 boolean values. The problem states that it must work for different input. For instance the user can enter `150`. The problem does say that you only need to consider a maximum input of `1000`. So declare a boolean array of size `1000`. If the user enters, say, `15`, you just use `x[0]`, ..., `x[14]` and ignore the rest. If the user enters `42`, then just use `x[0]`, ..., `x[41]` and ignore the rest.