

Lab 3 - While Loop & Functions

Directions

- You are encouraged to help your classmates, and to receive as much help as you like. Assignments, however, are individual work.

1. Going Loopy

A while loop allows you to execute one or more statements multiple times. For example

```
int i = 0;
while (i < 4) {
    println("Value: {}", i);
    i++;
}
```

- Create a new program inside your project named `lab3.cpp`
- Have your program ask the user for a number N (an integer), and then print all numbers from N down-to 0.
 - Hint: You read in N from the user; store it in a variable named something like `userN`, or even just `n` is fine here. Don't use " N " because that suggests its a constant.*
 - Hint: Once you have read in N , don't change that variable because later code will want it. You'll need to create a new variable for use in your loops.*
- Modify your loop to only print out even numbers from N down-to 0 inclusive. There are many ways you can make it do this; any way is fine.
- Independent of the above loop (i.e., write it at the end of `main()`, after your loop), sum all numbers from 0 up to N . Print the answer.
 - Hint: You're using the same N the user entered above. If the previous loop changed the value of N , you won't be able to use it here.*
 - Hint: You'll need a new variable to hold the sum of values.*
- Independent of the above work, multiply all numbers from 1 up to N . Print the answer. This is the value of N -factorial.
 - Hint: You'll need a new variable to hold the product of numbers.*
 - You'll want to start this variable at 1, so when you later multiply it by the numbers between 1 and N it correctly computes the result.*

6. Sample outputs:

```
Enter N: 2
All even numbers 2..0 = 2, 0,
Sum 0..2 = 3
Product 1..2 = 2
```

```
Enter N: 5
All even numbers 5..0 = 4, 2, 0,
Sum 0..5 = 15
Product 1..5 = 120
```

```
Enter N: 20
All even numbers 20..0 = 20, 18, 16, 14, 12, 10, 8, 6,
4, 2, 0,
Sum 0..20 = 210
Product 1..20 = -2102132736
```

7. Use the debugger in your IDE to single step through your program with N=4.
8. **Understanding: Answer the following question in a comment in your code**
Explain why N = 20 gives an unexpected value.

2. Output vs Return

1. Move all the code in `main()` into a new function name `runLoopPart()`; run this from `main()`. To do this, do the following:
 - In the same file, above `main()`, create a void function (i.e., a function with return type void) named `runLoopPart()`

```
void runLoopPart()
{
    // ...
}
```
 - Into this function move the code from the previous section (i.e., all the code you typed into `main()`).
 - Have `main()` now call your new function. `main()` will look like:

```
int main()
{
    runLoopPart();
}
```
 - After proving it works, you may want to comment out the call in `main()` to `runLoopPart()` so that when you run your program it does not print output from the previous section.
2. Above `main()`, create another function with the following header:

```
void displayWelcome() {
}
```

 - Make this function output a welcome message to the screen.
 - Have `main()` call this function and test your program.
 - Have `main()` call this function a **second time** and test your program.
 - Run the program. Output should look like (OK to have a different message!):

```
Hello and welcome to the lab on functions!
Hello and welcome to the lab on functions!
```

3. Run your program in the debugger and single-step through each statement:

- Set a breakpoint on the first line of `main()`
- Experiment with step-over and step-into; what is the difference?

4. Create a function with the following header:

```
void displaySumOfSquaresUptoN(int n) {
}
```

- Implement this function so that it adds up and prints out the square of each integer

from 1 to n . (i.e. display $\sum_{i=1}^n i^2$)

For example, if $n=4$, it adds $1^2+2^2+3^2+4^2 = 30$, and outputs the sum to the screen.

5. In `main()`, prompt the user for their favourite number (an integer), and pass it as an argument to the `displaySumOfSquaresUpToN()` function.

- Run the program. Your output should look like:

```
Hello and welcome to the lab on functions!
Hello and welcome to the lab on functions!
Enter your favourite number: 5
Sum of squares 1 .. 5 = 55
```

6. Add a new function with the following header:

```
int getSumOfSquaresUpToN(int n)
```

- Make this function do the same thing as `displaySumOfSquaresUpToN()`, except instead of printing to the screen, have it **return** the sum.
Hint: Copy and paste part of the `displaySumOfSquaresUpToN()` function.

- Have `main()` call this function, passing in the user's favourite number and have `main` print its return value to the screen using the following code:

```
int sum = getSumOfSquaresUpToN(favNum);
println("Returned sum of squares is: {}", sum);
```

- Your output should now be:

```
Hello, and welcome to the lab on functions!
Hello, and welcome to the lab on functions!
Enter your favourite number: 5
Sum of squares 1 .. 5 = 55
Returned sum of squares is: 55
```

7. In `main()`, write code to outputs a number of stars (*) equal to the sum of squares from 1 to n .

- For example, if $n=3$, the sum of squares from one to three is 14, so output 14 *'s:

```
*****
```
- Use one of the functions you wrote in this section to help accomplish this.
*Hint: Does **displaying** the sum from 1 to n on the screen help with this? Does getting (**returning**) the sum from 1 to n help with this?*

- *Hint: Any time a function returns a value that you need, remember to store it in a variable.*

- Output:

```
Hello, and welcome to the lab on functions!
Hello, and welcome to the lab on functions!
Enter your favourite number: 5
Sum of squares 1 .. 5 = 55
Returned sum of squares is: 55
*****
```

8. Move the code that generates the row of '*'s into its own function. Pass in the number of stars to draw as an argument.
 - What should this function be named? Pick a name which best describes it.
 - What should the return type be for this function? Recall it just prints to the screen.
 - What arguments are needed for this function? What type and name should they have?
 - How can it generate a bunch of stars? (*Hint: loop!*)
9. (Optional) Try simplifying your `displaySumOfSquaresUpToN()` function. Think about the rest of the code you have written for this lab and figure out how to reduce the duplication of code.
 - *Hint: You can call one function from inside another function.*
 - *Hint: You may need to reorder your functions so that the function being called appears the function calling it.*

10. Understanding

- What is the *fundamental* difference between a function *returning a value*, and a function *displaying a value*?
- For outputting the stars to the screen, will either one of the functions `displaySumOfSquaresUpToN()` or `getSumOfSquaresUpToN()` help? Why or why not?
- What is the difference between passing in an argument and using `cin` in the function?
- Can one function call another function?

3. Number Checking - Optional

If you would like more practice with if-statements then create a new function named `numberChecking()`, call it from `main()`, and implement the following.

1. Prompts the user to enter their favourite number (as an integer).
2. If the number is equal to 42, then using **two print statements**, print “Awesome choice!” on one line, and “That's mine too!” on a second line.
3. Independent of previous checks (i.e., as its own separate if/if-else/else.. statement), if the number is..
 - negative: write “How negative of you!”
 - zero: write “That's nothing!”
 - positive: write “How positive of you!”
4. Independent of previous checks, if the number is:
 - even: write “Now you are just getting even with me.”
 - odd: write “How odd!”
5. Independent of previous checks, if the number is:
 - less than 10, write “Small.”,
 - between 10 and 50 inclusive, write “Medium.”,
 - greater than 50 and less than 100, write “Large.”,
 - between 100 and 1000 inclusive, write “Huge.”,
 - greater than 1000, write “Massive!”

HINT: Use an if-else if... structure to check non-overlapping regions (like above). Such as (different numbers, but similar idea):

```
if (x < 5) {
} else if (x <= 30) {
} else if (x < 80) {
} else {
}
```

6. Independent of previous checks, if the number is greater than 10,000 then:
 1. Print the number of thousands in the number. For example, if given the number 21456, print “That's 21k!”.

Hint: what math operator can you apply that will extract just the thousands?

 2. Using a **nested if** statement (inside your check for greater than 10,000), if the number is:
 - evenly divisible by 1000, print “With nothing left over.”
 - otherwise, print the number left over once the thousands are stripped away. For example, given the number 21456, print “With 456 left over.”
7. Sample Outputs:
42¹:

¹ If only we knew the question...

```
Enter your favourite number: 42  
Awesome choice!  
That's mine too!  
How positive of you!  
Now you are just getting even with me.  
Medium.
```

Negative:

```
Enter your favourite number: -5  
How negative of you  
How odd!  
Small.
```

Massively odd:

```
Enter your favourite number: 12345  
How positive of you!  
How odd!  
Massive!  
That's 12k!  
With 345 left over.
```

Zero:

```
Enter your favourite number: 0  
That's nothing!  
Now you are just getting even with me.  
Small.
```

Larger than 10,000:

```
Enter your favourite number: 21456  
How positive of you!  
Now you are just getting even with me.  
Massive!  
That's 21k!  
With 456 left over.
```

```
Enter your favourite number: 951000  
How positive of you!  
Now you are just getting even with me.  
Massive!  
That's 951k!  
With nothing left over.
```

4. Extra Challenge Tasks - Optional

These are not required to complete the lab, but interesting exercises to try if you have time:

- Change all the magic numbers in this lab to be named constants.
- Check if your code matches the suggested coding style guide (see assignments section of course website for link).
- Review a friend's implementation of both parts of the lab and look for suggestions you can give them about their code. Here are some things to look for:
 1. Correct indentation.
 2. Correct logic on test cases.
 3. Boundary cases on their checks.
 4. Inconsistent use of { ... }

5. Lab credit

- Complete above steps in one file. OK to skip any “optional” sections.
- Comment your code correctly: comment for every ~2-5 lines of code.
- Submit your one lab file to CourSys.