Project 1

Programming and Algorithms II CSCI 211

Objectives

- Practice input and output
- Practice using arrays
- Practice implementing loops
- Practice implementing functions

Notes: You must complete Lab 1 before starting Project 1. This is an individual assignment. Do not share code or look at anyone else's code. You may share ideas with others, but not code. If you share code or collaborate too closely with others, your code is very likely to be flagged by our anti-plagiarism checker and you may not receive any credit for your submissions. If there are multiple instances of your submissions getting flagged, I have to fail you in the course. Please use common sense and avoid this common pitfall.

Overview

You will write a program named chart to read a sequence of integers from "standard input", and output a "bar chart" to "standard output", using asterisk and space characters ('*' and ' ').

The input specifies a sequence of up to 100 integers greater than 0. The last value in the input must be a 0, indicating that the input is complete.

For example, if the input contains the sequence

1 2 3 4 3 2 1 6 0

then your program must write the following output to "standard out":

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

Note: Space characters are used to output the whitespace in the chart above.

There are 8 integers in the input (ignoring the 0), so there are 8 vertical "bars" in the bar chart. The height of each veritcal "bar" in the chart above is controlled by each of the integers. The first "bar" is one asterisk tall because the first integer in the input is a 1. The last "bar" is six asterisks tall because the last integer in the input (ignoring the 0) is 6.

Note 1: The only characters output on each line are space (' ') and asterisk ('').*

Note 2: You must output the lines of the bar chart in from top to bottom (this is the only way you can write to "standard output"). Each line is output from left to right.

The input might have numbers on separate lines, or on the same line separated by "white space". The best way to handle input like this is to use the built-in C++ **>>** "extraction operator", which reads input delimeted by "white space". In computer input, "white space" means spaces, tabs and "end of line" characters. This type of input is also known as "token based" input.

...continued on next page...

Requirements

The maximum allowed number of inputs is 100, excluding the last 0. You must store each non-zero integer in the input into an array of integers. Do not use a vector.

Formatting and Style

Your program must be neatly organized and consistently indented. Each time you start a new block of code using a "begin curly brace" ('{'), use the tab key to indent to the right one additional level. After each end curly brace ('}'), use shift-tab to move one tab stop to the left.

For example:

```
int main()
{
    int x;
    while (cin >> x)
    {
        cout << x;
    }
    return 0;
}
```

You may also elect to minimize your line count by not putting the begin curly-brace character on a new line, like this:

```
int main() {
    int x;
    while (cin >> x) {
        cout << x;
    }
    return 0;
}
```

FYI: I prefer the second (more compact) way but you may decide which you prefer.

...continued on next page...

Comments

You must have informative comments throughout your program. There are three types of comments required in our 211 Projects:

1) File Header Comments: The top of each file must contain a "file header comment" in the following format:

```
// <Name of file>
// <Your Last Name>, <Your First Name>
// <Your CSUC Portal Username>
```

For example, the top of file chart.cpp authored by your instructor would start with the following header block:

```
// chart.cpp
// Herring, Brian
// bherring
```

2) Function & Method Header Comments: The top of each function or method implementation (a method is just a function that lives inside of a C++ Class) must have a "funtion / method header comment" in the following format:

```
// Purpose: <Function / Method Purpose Description>
// Input Parameters: <Input Parameter Descriptions (separate multiple with a semicolon) >
// Output Parameters: <Output Parameter Descriptions (separate multiple with a semicolon)>
// Return Value: <Description: Description of return value.>  (You may omit this if the function
// or method does not return a value).
```

For example:

```
// Purpose: Main program entry point.
// Input Parameters: None
// Output Parameters: None
// Return Value: Returns 0 on success.
int main() {
   return 0;
}
```

3) In-line comments. Within the implementation of each function or method that you write, you must add "in-line" comments every few to several lines of code that explains what the code is doing and why.

```
// <In-line comment>
```

For example:

```
// Write each element of input array to "standard output" on separate lines.
for (int i=0; i < count; i++) {
   cout << my_array[i] << endl;
}
```

...continued on next page...

Use Constants ('const' variables) to Avoid Duplication of Value References

The max allowed number of integers that can be input is 100, so you will create an array of 100 integers. It is good programming practice to define the 100 in a single place with a the symbolic name MAX, and share that symbol where needed:

```
const int MAX = 100;
int values[MAX];
...
for (int i = 0; i < MAX; i++) {
    ...
}
```

Note how MAX is defined in one place and used in multiple places. Imagine MAX being used in several more places in the code. This is good because if later you need to change MAX's value, you only need to change it in one place, rather than having to hunt down and modify each of the values manually.

...continued on next page...

Implementation Steps

Carefully read and follow the steps below exactly in order. These steps are provided to help you get Project 1 completed as efficiently as possible.

Some personal advice from your instructor: I want you all to succeed. Please don't rush or try to take short-cuts with these steps. They are provided to make your learning journey as smooth as possible. Over the years, I've noticed more and more students that seem to be unwilling or unable to read and follow step-by-step written directions, which places a much bigger load on your instructor and the TAs. I believe this is partially due to a trend towards a decreased ability to focus on linear tasks and decreased average attention spans. There are many possible causes of this trend: An overall decrease in reading books, along with a large increase in consuming social media, video games and other hyper-stimulating, non-linear forms of content. If you struggle with attention span, focus and/or concentration, I challenge you to improve in these areas. Read more books on topics you enjoy, reduce non-required screentime. Consider doing daily meditation. Get regular physical exercise to the extent possible. These things can make a huge difference in your ability to focus and succeed in this course and for the rest of your academic and professional career. Small changes can have large results if done consistently.

If you get stuck on a step, that's ok, you can always ask for help in lab or office hour, but you should be able to tell us what step you are stuck on. If you don't know what step you are on, this is a red flag and you need to go back and follow the steps.

Here we go, good luck! Be sure to save your work in progress after each step.

1. Use the nano editor to create and open file **chart.cpp** within your **~/csci211/projects/p1** directory.
cd ~/csci211/projects/p1
nano chart.cpp

2. Enter your file header comment block (as specified under **Additional Requirements: Comments**, above) at the top of **chart.cpp** and save the file.

3. Add the following two lines after the file header comment block:

```
#include <iostream>
using namespace std;
```

Note: The first line above includes the C++ interfaces needed for doing the necessary input and output. The second line above is the conventional way to tell the C++ compiler that we will be using the "std" namespace as the default. This will help keep our code more compact by not having to explicitly specify namespace prefixes for some of the C++ library names like cin (used for reading from "standard input") and cout (used for reading "standard input"). Without the 'using namespace std' specifier, we'd have to type std::cin and std::cout everywhere in our program, instead of just cin and cout.

...continued on next page...

4. Create your **main** function at the bottom of chart.cpp (make sure the **main** function is always the very last function specified in the file). Note: The purpose of the **main** function is to provide an "entry point" for your program. It is where your program begins execution. Be sure to include the function header block above the implementation of the main function, like this:

```
// Purpose: Program entry point.
// Input Parameters: None.
// Output Paramters: None.
// Return value: Always returns 0 (success).
int main() {
    return 0;
}
```

5. Define a constant for the number of elements needed in your input array. Define this at the top of your main function:

```
const int MAX = 100;
```

6. Declare your input array, on the next line of main after MAX is declared:

```
int input_array[MAX];
```

7. Just after the declaration of the input_array, add the code to read all of the input from "standard input" into the input_array. Use an integer counter to keep track of the number of inputs read and the next available array element index. Stop reading elements when you read the value 0. Here is some code to get you started:

```
int value;
int count = 0;
while (cin >> value) {
    // Add code here to exit the while loop (break) if the value just read equals 0.
    // Add code here to insert the current value into the next available
    // array slot (count).
    // Add code here to increment count.
}
```

8. Write a function that scans all of the values in the input_array and returns the maximum value found:

```
int find_largest(int values[], int size) {
   int largest = 0;
   /* code to find the largest integer goes here */
   return largest;
}
```

Create the function above the **main** function. Don't forget to create the required comments for this function.

9. Add code in main() to call the **find_largest** function. Pass in the **input_array** as the first parameter. The second parameter needs to be the integer varaible that contains the number of elements that were inserted into the array (you need to determine which one that is). Declare a new integer variable named **largest** in the **main** function to receive the value returned by the call to **find_largest.**

...continued on next page...

10. Define a new function above the **find_largest** function. This function will output the chart to "standard output". The input arguments to this function are: 1) The input array, 2) the number of elements in the input array, 3) the largest value in the input array. You will determine the name of the new function, and the names and types of the input parameters, and you will write the code to call this function from your **main** function. The logic inside of this function will output the chart line by line, from top to bottom, and left to right on each line. Hints: Create a doubley-nested **for** loop. The outer loop variable is an integer value that starts with the largest array value and decrements (decreases by one) on each loop iteration, until the value reaches 1. The inner loop is used to output the current line of the chart, one character at a time, from left to right. Each inner loop step will write either an asterisk character ('*') or a space character (' ') to standard output. You will use the input array and the inner loop variable to determine which character to output at each step. Hint: Is the current input array value large enough to output an asterisk for the current chart row? If not, output a space. Step 10 is the hardest. If you get stuck, use a piece of scratch paper to draw the chart for the example data given in the first section, using the techniques described above. Spend some time trying to solve this on your own. Ask for help if you get stuck.

Compile & Test

Use the **run_tests** program (that you set up during Lab 1) to validate that all tests are passing locally before submitting your final solution to TurnIn. See **Guides > Advanced Local Testing** if you don't know how to use the run_tests program.

Make sure you output an "end of line" (using '<< endl' or '\n') with your last line of output. If you forget this, you will not pass the project 1 test set.

Submit to Turnin

Submit your completed program to https://turnin.ecst.csuchico.edu/ under Project 1.
Note: Submit only your C++ source file (**chart.cpp**). Do not submit your object file, Makefile or your executable program.

Each student will complete and submit this assignment individually. Do not work with anyone else on this or any other CSCI 211 project. An anti-plagiarism tool will be used to ensure that all student submissions are unique. DO YOUR OWN WORK. DO NOT ALLOW ANYONE ELSE TO SEE OR USE YOUR SOLUTION. Submit your work before the due date to receive maximum credit, or at least before the late cut-off date to receive partial credit (see Course Content > Syllabus for info on the CSCI 211 lateness policy).