650615017 Nat Kempet - Driver 50%
6506150030 Ritthanupahp Sitthananun - Navigator/Documentation 50%

First we start with writing an input from the user and print it out.

```
char letter;

cout << "Enter a letter: ";
cin >> letter;
cout << letter << endl;
```

This turns out to work fine so we're moving on to the next requirement. The next test case we're going to try is to print the letters of the alphabet that comes before the letter input from the user.

We figured that we need a for loop that repeats 'n' amount of times depending on how far the letter input is from the letter a so we tried making a for loop that prints that. Before that we need to test if the calculation works.

```
int n;
n = letter - 'A';
cout << n << endl;
```

By subtracting the character 'A' we could find the difference in integer position from input letter to 'A'.
However, the problem here is that the print isn't the expected outcome

```
Enter a letter: A
0
```

```
Enter a letter: B
1
```

```
Enter a letter: D
3
```

There seems to be a problem where the difference between the letter A - A will result in the integer of 0.
After thinking for a bit, we figured out that all we had to do was add a 1 to the letter n which will give us the result we want!

```
Enter a letter: A
1
```

```
Enter a letter: B
2
```

```
Enter a letter: D
4
```

Up until this point, code from part [1] and [2] seems to work together without issue.
Next, we're going to try and use a for-loop for the letters according to the input.

```
for(int i = 0; i < n; i++){
    if(i%2 == 0){
        cout << char('A' + i) << " ";
    }else if(i%3 == 0){
        cout << endl;
    }

}
```

We made a mistake, we have endl in the else() clause. At this point we are stuck on how to create a for loop that prints out the top half of the diamond.

We shifted gears to figuring out the number of spaces in between letters which we found out to be odd numbers. Having the space of 1, 3, 5, 7, … for B, C, D, E respectively (except for A which is the top of the diamond). This means that with each new line the space is incremented by 2.

Our next test case would be trying to put ‘A’ into the middle of the diamond. Knowing the above information, ‘i%n’ will determine the number of spaces for ‘A’ to be in the middle of the diamond.

```
for(int i = 0; i < n; i++){
    if(i%n == 0){
        cout << char('A') << " ";
}else{
    cout << " ";
}
```

This doesn't go as planned. The issue might be because the spaces aren't printed.

We decided to introduce a new component which is strings. The reason being is to save the output from the for-loop unlike a char variable.

```
for(int i = 0; i < n; i++){
    if(i%n == 0){
        output = output + char('A');
}else{
output = output + " ";
}
cout << output << endl;
```

We added a ‘(i+1)%(n-1)’ to make the ‘A’ in the center of the diamond

```
for(int i = 0; i < n; i++){
    // Places 'A' in middle of the first row
    if((i+1)%n == 0){
        output = output + char('A');
}else{
output = output + " ";
```

We changed the modulator so that it works now only for ‘A’ and places the ‘A’ in the middle.

```
for (int i = 0; i < n; i++)
{
    // Places 'A' in middle of the first row
    if ((i + 1) % n == 0)
    {
        output = output + char('A');
    }
    else
    {
        output = output + " ";
    }

output = output + "\n";

}
for (int i = 1; i < n; i++)
{
    output = output + char('A' + i);
}
```

The next test case to tackle is the rows after the ‘A’. Our test case is to make the rest of the rows. By storing the letters into the string and displaying it in output after the for-loop is completed.

```
for (int i = 1; i < n; i++)
{
    while(counter != 2){

    }
}
```

We suspect that using a while-loop can help make only 2 letters pre row.

```
char letter;
int counter = 0;
int spaces1 = 0;
int spaces2 = 0;
string output = "";
```

Creating two more integers to validate the spaces between the letters after the first row. (This is temporary and might be removed if not needed).

```
for (int i = 0; i < n; i++)
{
    // Places 'A' in middle of the first row
    if ((i + 1) % n == 0)
    {
        output = output + char('A');
    }
    else
    {
        output = output + " ";
        ++spaces1;
    }
}
```

The ‘++spaces1’ is used to test the increment of the spaces between the letters. The print validates that it’s correct. We decided that we should be using more for-loop to make the rest of the rows.

```
// Top half of the diamond
for (int i = 1; i < n; i++)
{
    while(counter != 2){
        --spaces1;
        spaces2 = spaces2 + 2;
        for(i = 0; i < spaces1; i++)
        {
            output = output + " ";
        }
        output = output + char('A' + 1);
    }
}
```

We are creating a for-loop outside of the first for-loop. This is mainly focused on creating the letters after ‘A’. ‘spaces1’ is used for the spaces before the letter and ‘spaces2’ is used in between the two letters. ‘spaces1’ decreases as it gets closer to the input letter from the user to form the diamond shape.

```
// Top half of the diamond
for (int i = 1; i < n; i++)
{
    spaces2 = spaces2 + 2;
    --spaces1;
    for(i = 0; i < spaces1; i++)
        {
            output = output + " ";
        }
        output = output + char('A' + 1);
    for(i = 0; i < spaces2; i++)
        {
            output = output + " ";
        }
        output = output + char('A' + 1);
        output = output + "\n";
}
cout << output;
```

We ended up creating 2 for-loops in the main for-loop for the top half of the diamond (excluding the first row). We increment the alphabet by '+1' to the letter 'A'. The result is as follows.

```
Enter a letter: D
   A
  C  C
 B    E
```

We got the top half shape of the diamond but the letters are not what we wanted. We decided to change the '+1' into 'j' for each for-loop here:

```
// Top half of the diamond
for (int i = 1; i < n; i++)
{
    spaces2 = spaces2 + 2;
    --spaces1;
    for(int j = 0; j < spaces1; j++)
        {
            output = output + " ";
        }
        output = output + char('A' + i);
    for(int j = 0; j < spaces2; j++)
        {
            output = output + " ";
        }
        output = output + char('A' + i);
        output = output + "\n";
}
cout << output;
```

The following is the result:

```
Enter a letter: D
   A
  B  B
 C    C
D      D
```

Success! We use the same i for each nested for-loop to represent the characters on each row and it works well.

Next we are working on the bottom half of the diamond. Our test case is to duplicate our previous code and make the function do the opposite of the top half function.

```
for (int i = n - 1; i > 0; i--)
{
    if(i == 1){
        output = output + char('A' + (i - 1));
        break;
    }
    spaces2 = spaces2 - 2;
    ++spaces1;
    for (int j = 0; j < spaces1; j++)
    {
        output = output + " ";
    }
    output = output + char('A' + (i - 1));
    for (int j = 0; j < spaces2; j++)
    {
        output = output + " ";
    }
    output = output + char('A' + (i - 1));
    output = output + "\n";
}
```

```
Enter a letter: D
   A
  B  B
 C    C
D      D
 C    C
  B  B
A
```

The idea of reversing the function worked except for the letter ‘A’. The problem lied in the if clause, we should have put in the middle of the code and write a break in the if clause to store the A in the middle of the for-loop while keeping the spaces on both sides the same.

```
for (int i = n - 1; i > 0; i--)
{
    spaces2 = spaces2 - 2;
    ++spaces1;
    for (int j = 0; j < spaces1; j++)
    {
        output = output + " ";
    }
    if(i == 1){
        output = output + char('A' + (i - 1));
        break;
    }
    output = output + char('A' + (i - 1));
    for (int j = 0; j < spaces2; j++)
    {
        output = output + " ";
    }
    output = output + char('A' + (i - 1));
    output = output + "\n";
}
```

Hurray! Here's the final result:

```
Enter a letter: D
   A
  B  B
 C    C
D      D
 C    C
  B  B
   A
```