Started on	Monday, 29 January 2024, 5:18 PM
State	Finished
Completed on	Monday, 29 January 2024, 6:53 PM
Time taken	1 hour 34 mins
Marks	40.00/40.00
Grade	**10.00** out of 10.00 (**100**%)

Question 1

Correct

Mark 10.00 out of 10.00

if and *else* are two of the most frequently used conditionals in C/C++, and they enable you to execute zero or one conditional statement among many such dependent conditional statements. We use them in the following ways:

1. *if:* This executes the body of bracketed code starting with $statement1$ if $condition$ evaluates to *true*.

```
if (condition) {
    statement1;
    ...
}
```

2. *if - else:* This executes the body of bracketed code starting with $statement1$ if $condition$ evaluates to *true*, or it executes the body of code starting with $statement2$ if $condition$ evaluates to *false*. Note that only *one* of the bracketed code sections will ever be executed.

```
if (condition) {
    statement1;
    ...
}
else {
    statement2;
    ...
}
```

3. *if - else if - else:* In this structure, dependent statements are chained together and the $condition$ for each statement is only checked if all prior conditions in the chain evaluated to *false*. Once a $condition$ evaluates to *true*, the bracketed code associated with that statement is executed and the program then skips to the end of the chain of statements and continues executing. If each $condition$ in the chain evaluates to false, then the body of bracketed code in the *else* block at the end is executed.

```
if(first condition) {
    ...
}
else if(second condition) {
    ...
}
.
.
.
else if((n-1)'th condition) {
    ....
}
else {
    ...
}
```

Given a positive integer n, do the following:

- If $1 \leq n \leq 9$, print the lowercase English word corresponding to the number (e.g., `one` for 1, `two` for 2, etc.).
- If $n > 9$, print `Greater than 9`.

Input Format

A single integer, n.

Constraints

- $1 \leq n \leq 10^9$

Output Format

If $1 \leq n \leq 9$, then print the lowercase English word corresponding to the number (e.g., `one` for 1, `two` for 2, etc.); otherwise, print `Greater than 9`.

Sample Input 0

```
5
```

Sample Output 0

```
five
```

Explanation 0

`five` is the English word for the number 5.

Sample Input 1

```
8
```

Sample Output 1

```
eight
```

Explanation 1

`eight` is the English word for the number 8.

Sample Input 2

```
44
```

Sample Output 2

```
Greater than 9
```

Explanation 2

$n = 44$ is greater than 9, so we print `Greater than 9`.

For example:

Input	Result
5	five
8	eight
44	Greater than 9

Answer: (penalty regime: 0 %)

Reset answer

```cpp
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);



int main()
{
    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    if (n==1){
        cout<<("one");
    }else if(n==2){
        cout<<("two");
    }else if(n==3){
        cout<<("three");
    }else if(n==4){
        cout<<("four");
    }else if(n==5){
        cout<<("five");
    }else if(n==6){
        cout<<("six");
    }else if(n==7){
        cout<<("seven");
    }else if(n==8){
        cout<<("eight");
    }else if(n==9){
        cout<<("nine");
    }else{
        cout<<("Greater than 9");
    }



    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
```

	Input	Expected	Got	
✔	5	five	five	✔
✔	8	eight	eight	✔
✔	44	Greater than 9	Greater than 9	✔

Passed all tests! ✔

▸ **Show/hide question author's solution (Cpp)**

Correct

Marks for this submission: 10.00/10.00.

Question 2

Correct

Mark 10.00 out of 10.00

A *for* loop is a programming language statement which allows code to be repeatedly executed.

The syntax is

```
for ( <expression_1> ; <expression_2> ; <expression_3> )
    <statement>
```

- *expression_1* is used for initializing variables which are generally used for controlling the terminating flag for the loop.
- *expression_2* is used to check for the terminating condition. If this evaluates to false, then the loop is terminated.
- *expression_3* is generally used to update the flags/variables.

A sample loop is

```
for(int i = 0; i < 10; i++) {
    ...
}
```

In this challenge, you will use a for loop to increment a variable through a range.

Input Format

You will be given two positive integers, a and b ($a \leq b$), separated by a newline.

Output Format

For each integer n in the inclusive interval $[a, b]$:

- If $1 \leq n \leq 9$, then print the English representation of it in lowercase. That is "one" for 1, "two" for 2, and so on.
- Else if $n > 9$ and it is an even number, then print "even".
- Else if $n > 9$ and it is an odd number, then print "odd".

Note: $[a, b] = \{x \in \mathbb{Z} \mid a \leq x \leq b\} = \{a, a+1, \ldots, b\}$

Sample Input

```
8
11
```

Sample Output

```
eight
nine
even
odd
```

For example:

Input	Result
8 11	eight nine even odd

Answer: (penalty regime: 0 %)

Reset answer

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

int main() {
    int a;
    int b;
    cin>>a;
    cin>>b;

    for(int n=a; n<b+1; n++){
    if (n==1){
        cout<<("one")<<endl;
    }else if(n==2){
        cout<<("two")<<endl;
    }else if(n==3){
        cout<<("three")<<endl;
    }else if(n==4){
        cout<<("four")<<endl;
    }else if(n==5){
```

```
        cout<<("five")<<endl;
    }else if(n==6){
        cout<<("six")<<endl;
    }else if(n==7){
        cout<<("seven")<<endl;
    }else if(n==8){
        cout<<("eight")<<endl;
    }else if(n==9){
        cout<<("nine")<<endl;
    }else{
        if(n%2==0){
            cout<<("even")<<endl;
        }else{
            cout<<("odd")<<endl;
        }
    }
}



    return 0;
}
```

	Input	Expected	Got	
✔	8 11	eight nine even odd	eight nine even odd	✔

Passed all tests! ✔

▸ **Show/hide question author's solution (Cpp)**

Correct

Marks for this submission: 10.00/10.00.

Question 3

Correct

Mark 10.00 out of 10.00

Objective

In this challenge, we practice reading input from stdin and printing output to stdout.

In C++, you can read a single whitespace-separated token of input using cin, and print output to stdout using cout. For example, let's say we declare the following variables:

```
string s;
int n;
```

and we want to use *cin* to read the input "High 5" from stdin. We can do this with the following code:

```
cin >> s >> n;
```

This reads the first word ("High") from stdin and saves it as string s, then reads the second word ("5") from stdin and saves it as integer n. If we want to print these values to stdout, separated by a space, we write the following code:

```
cout << s << " " << n << endl;
```

This code prints the contents of string s, a single space (" "), then the integer n. We end our line of output with a newline using endl. This results in the following output:

```
High 5
```

Task

Read 3 numbers from stdin and print their sum to stdout.

Input Format

One line that contains 3 space-separated integers: a, b, and c.

Constraints

- $1 \leq a, b, c \leq 1000$

Output Format

Print the sum of the three numbers on a single line.

Sample Input

```
1 2 7
```

Sample Output

```
10
```

Explanation

The sum of the three numbers is $1 + 2 + 7 = 10$.

For example:

Input	Result
1 2 7	10

Answer: (penalty regime: 0 %)

Reset answer

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    int a;
    int b;
    int c;
    cin>>a>>b>>c;
    int sum=a+b+c;
    cout<<sum;
    return 0;
}

```

	Input	Expected	Got	
✔	1 2 7	10	10	✔

Passed all tests! ✔

▸ **Show/hide question author's solution (Cpp)**

Correct

Marks for this submission: 10.00/10.00.

Question 4

Correct

Mark 10.00 out of 10.00

Sorting

One common task for computers is to sort data. For example, people might want to see all their files on a computer sorted by size. Since sorting is a simple problem with many different possible solutions, it is often used to introduce the study of algorithms.

Insertion Sort

These challenges will cover *Insertion Sort*, a simple and intuitive sorting algorithm. We will first start with a nearly sorted list.

Insert element into sorted list

Given a sorted list with an unsorted number e in the rightmost cell, can you write some simple code to insert e into the array so that it remains sorted?

Since this is a learning exercise, it won't be the most efficient way of performing the insertion. It will instead demonstrate the brute-force method in detail.

Assume you are given the array $arr = [1, 2, 4, 5, 3]$ indexed $0 \ldots 4$. Store the value of $arr[4]$. Now test lower index values successively from 3 to 0 until you reach a value that is lower than $arr[4]$, at $arr[1]$ in this case. Each time your test fails, copy the value at the lower index to the current index and print your array. When the next lower indexed value is smaller than $arr[4]$, insert the stored value at the current index and print the entire array.

Example

$n = 5$
$arr = [1, 2, 4, 5, 3]$

Start at the rightmost index. Store the value of $arr[4] = 3$. Compare this to each element to the left until a smaller value is reached. Here are the results as described:

```
1 2 4 5 5
1 2 4 4 5
1 2 3 4 5
```

Function Description

Complete the *insertionSort1* function in the editor below.

insertionSort1 has the following parameter(s):

- *n*: an integer, the size of arr
- *arr*: an array of integers to sort

Returns

- *None:* Print the interim and final arrays, each on a new line. No return value is expected.

Input Format

The first line contains the integer n, the size of the array arr.
The next line contains n space-separated integers $arr[0] \ldots arr[n-1]$.

Constraints

$1 \leq n \leq 1000$
$-10000 \leq arr[i] \leq 10000$

Output Format

Print the array as a row of space-separated integers each time there is a shift or insertion.

Sample Input

```
5
2 4 6 8 3
```

Sample Output

```
2 4 6 8 8
2 4 6 6 8
2 4 4 6 8
2 3 4 6 8
```

Explanation

3 is removed from the end of the array.
In the 1^{st} line $8 > 3$, so 8 is shifted one cell to the right.
In the 2^{nd} line $6 > 3$, so 6 is shifted one cell to the right.
In the 3^{rd} line $4 > 3$, so 4 is shifted one cell to the right.
In the 4^{th} line $2 < 3$, so 3 is placed at position 1.

Next Challenge

In the next challenge, we will complete the insertion sort.

For example:

Input	Result
5 2 4 6 8 3	2 4 6 8 8 2 4 6 6 8 2 4 4 6 8 2 3 4 6 8

Answer: (penalty regime: 0 %)

Reset answer

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'insertionSort1' function below.
 *
 * The function accepts following parameters:
 *  1. INTEGER n
 *  2. INTEGER_ARRAY arr
 */

void insertionSort1(int n, vector<int> arr) {
    if(n==0)
        return;
    if(n==1)
        cout<<arr[n-1]<<endl;
    int curr = arr[n-1];
    int i=n-2;
    while(i>=0){
        if(arr[i]>=curr){
            arr[i+1]=arr[i];
        }
        else{
            arr[i+1]=curr;
            i=-1;
        }
        for(int j=0;j<n;j++)
            cout<<arr[j]<<" ";
        cout<<endl;
        if(i==0){
            arr[i]=curr;
            for(int j=0;j<n;j++)
                cout<<arr[j]<<" ";
            cout<<endl;
        }
        i--;
    }

}


int main()
{
    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));
```

	Input	Expected	Got	
✔	5 2 4 6 8 3	2 4 6 8 8 2 4 6 6 8 2 4 4 6 8 2 3 4 6 8	2 4 6 8 8 2 4 6 6 8 2 4 4 6 8 2 3 4 6 8	✔

Passed all tests! ✔

▸ **Show/hide question author's solution (Cpp)**

Correct

Marks for this submission: 10.00/10.00.