1. **Problem Statement**

Create a recursive function to perform a factorial calculation and return the result

1. **Requirements**
   1. **Assumptions**
      1. We deal with integer numbers throughout the program
      2. User enters only integers as input that are positive and greater than 0
   2. **Specifications**
      1. Ask user to give a number
      2. Method status variable
      3. Function for factorial to be called from main
         1. Returns factorial as long as user input is positive. Else returns negative
2. **Decomposition Diagram**

|  |  |  |
| --- | --- | --- |
| **Main** | | |
| **Input** | **Process** | **Output** |
| Number to be used to calculate factorial | Calculate factorial | Print the calculated factorial to screen |
|  |  | Print “Invalid input. Negative number not allowed” if the user input is negative |

1. **Test Strategy**
   1. Valid Data
   2. Invalid Data
2. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User inputs positive number |  |  |  |  |
| Valid | 2 | User inputs zero |  |  |  |  |
| Invalid | 1 | User inputs negative number |  |  |  |  |

1. **Initial Algorithm**
   1. In main function
      1. Create variable to store user input
      2. Ask user for a number and store it in user input variable
      3. Create a Boolean variable for method status and set it equal to true
      4. Call factorial function and pass in the user input and method status
         1. If this results in a negative number, print out error message
         2. Else print out the calculated factorial
   2. In *factorial()* function
      1. If value passed in is negative, return -1 and assign method status variable to false
      2. Else return the value times the recursive function factorial
         1. This time pass in value – 1 and method status variable to the recursive function factorial
2. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User inputs positive number | 5 | 120 |  |  |
| Valid | 2 | User inputs zero | 0 | 1 |  |  |
| Invalid | 1 | User inputs negative number | -5 | “Invalid input. Negative number not allowed” |  |  |

1. **Code**

// CIS200\_Lab5\_Program2\_Home.cpp : Defines the entry point for the console application.

//

#include <iostream>

#include <string>

int factorial(int, bool &);

using namespace std;

int main()

{

int userInput;

cout << "Give me a number: " << endl;

cin >> userInput;

bool methodStatus = true;

if (factorial(userInput, methodStatus) >= 0)

{

cout << factorial(userInput, methodStatus) << endl;

}

else {

cout << "Invalid input: negative number not allowed" << endl;

}

system("pause");

return 0;

}

int factorial(int value, bool &methodStatus)

{

if (value < 0)

{

return -1;

methodStatus = false;

}

else if (value == 0)

{

return 1;

}

else

{

return value \* factorial(value - 1, methodStatus);

}

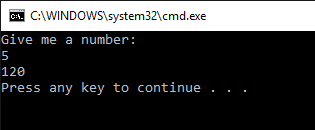
}

1. **Updated Algorithm**
   1. In main function
      1. Create variable to store user input
      2. Ask user for a number and store it in user input variable
      3. Create a Boolean variable for method status and set it equal to true
      4. Call factorial function and pass in the user input and method status
         1. If this results in a negative number, print out error message
         2. Else print out the calculated factorial
   2. In *factorial()* function
      1. If value passed in is negative, return -1 and assign method status variable to false
      2. Else if value passed in equals 0, return 1
      3. Else return the value times the recursive function factorial
         1. This time pass in value – 1 and method status variable to the recursive function factorial
2. **Test Plan Version 3**

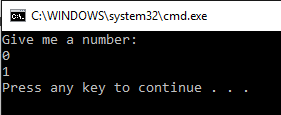
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User inputs positive number | 5 | 120 | 120 | Pass |
| Valid | 2 | User inputs zero | 0 | 1 | 1 | Pass |
| Invalid | 1 | User inputs negative number | -5 | “Invalid input. Negative number not allowed” | “Invalid input. Negative number not allowed” | Pass |

1. **Screenshots**

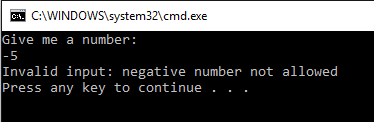
Valid Test Case 1:



Valid Test Case 2:



Invalid test Case 1:



1. **Status**

Program works perfectly with assumptions in mind