Bobcat Aerospace
Software Requirements Document

Software Name: Craft Designer Plus
Author: Gentry Atkinson
Version: 2.2.0

1. Introduction
Product Purpose: this product is being developed for the purpose of facilitating space vehicle design. This software will be able to take input from an engineer or designer and generate simple statistics about one vehicle which will be saved in a text file. The product will also be able to test these design files against some simple parameters.

2. Standards
Hardware: this product will be capable of running on any platform with a C++ compiler and a means of text I/O.

Schedule: completion of this product is expected no later than July 01.

Language: this project will be implemented using the C++ programming language

3. System Description

System Context: this will be a self-contained piece of software that will not rely on external platforms for its basic operation.

User Characteristics: users will be trained aerospace designers who are proficient with computer operation. This project will not be responsible for user training.

4. Functional Requirements

Menu: the user should be shown a simple text interface with the following options:

- **D**- design a new ship
- **T**- test a new ship
- **S-** simulate a flight
- **Q**- quit the program

This menu should allow the user to select one option and should print an error message if an invalid option is chosen. Based on user input one of the following subroutines should be run. Each subroutine should return to the menu after running, except for '**Q**'. The user should be able to enter a capital or lowercase letter as their input.

4.1 Design a New Ship (same as last program)

Input: the product will user collect the following information from the user through a simple text interface:

- **Ship Name:** a one word name for the vehicle
- **Designer Name**: the first and last name of the vehicle's designer
- **Ship Mass:** a numerical value representing the mass of a vehicle in kilograms
- **Engine Thrust:** a numerical value representing the force exerted by the vehicle's engines in newtons.

Output: the product should write the following values to a text file:

- **Ship Name**
- **Designer Name**
- **Ship Mass**
- **Acceleration:** calculated using (Engine Thrust/Ship Mass)
- **Change in Velocity:** calculated as (100,000 /Engine Thrust) * g0 * ln(2), where g0 is the constant gravitation value 9.8 and ln is the natural log function.
- **Maximum Altitude:** calculated as 100 + (Change in Velocity – 200)/10, with the result being the kilometers above sea level that the vehicle can safely travel.

Interface: the product should format the output as follows:

- "File written for [Ship Name]" should be printed on the **console** with [Ship Name] replaced by the actual Ship Name.
- The program's output should be written to a **file** named [Ship Name].txt with [Ship Name] replaced by the actual Ship Name.
- The file output should be formatted as follows:

```
##########[Ship Name]##########      //10 #s before and after
                                     //blank line
Ship mass:          [Ship Mass]kg    //Values in neat columns
Acceleration:       [Acceleration]m/s2  //
Impulse:            [Impulse]Ns/kg   //
Max Altitude:       [Max Alt.]km     //
                                     //blank line
File saved at:      [time stamp]     //
Designed by:        [Designer Name]  //
```

- All [Values] should be replace with correct values
- Comments should not appear in the file.
- Every item of output should have the correct unit as shown.

4.2 Test a Ship Design (same as last program)

Input: the product will prompt the user for the name of one space vehicle.

Output: the product will open one stored ship design file using the user's input as the file name. If that file does not exist the product will print an error message and return to the menu. Otherwise, all values should be read from the file and the following tests performed:

- **Test 1:** the Designer Name should not be "Thomas Harris". He was fired.
- **Test 2**: the ship mass should be less than 1000kg
- **Test 3**: the Acceleration should be more than 1 m/s2 but less than 10 m/s2
- **Test 4**: the Impulse must be at least 10.0 Ns/kg
- **Test 5:** if the Ship Mass is less than 100kg, then the Max Altitude must be at least 150km. Otherwise the Max Altitude must be at least 200km.

Interface: any ship that fails one of the tests should have the number of the first test that it fails printed to the console in this format:

[Ship Name] has failed test number [Test Number]

A ship that passes all of the tests should have the following message printed to the console:

[Ship Name] has passed all tests.

The output should have [Ship Name] and [Test Number] replaced with the appropriate values.

4.3 Simulate a Flight (same as last program)

Input: the product will prompt the user to enter an acceleration value for one space vehicle in meters per second squared and a number of second for the simulation to run.

Output: the product will compute the altitude and speed of the space vehicle at one second intervals for the number seconds input by the user using these equations:

- **Speed** = acceleration * time
- **Altitude** = (½) * acceleration * time^2

The space vehicle is always assumed to start motionless and on the ground (so with speed=0 and altitude=0).

Interface: the product should print text to the console starting with time=0 seconds and ending with the time entered by the user. The output should be formatted as follows:

At [time] seconds: [speed]m/s [altitude]m

[time], [speed], and [altitude] should all be replaced by the correct values. The three items on one line should be separated by one tab.

For altitudes greater than 1000m, your program should change the units to kilometers, and adjust the output value accordingly.

The first line of output will always be:

At 0 seconds: 0m/s 0m

The time should be displayed without a decimal place. Speed and altitude should both be printed with one decimal place of precision.

4.4 Quit (same as last program)

Input: the product will take no input

Output: the product will print a parting message on the console.

Interface: the parting message should “Thank you for using Craft Designer Plus”.

5. Non-functional Requirements

- The program should be submitted as a .cpp file with the name [Author’s Last Name]_assignment4.cpp, with [Author’s Last Name] replaced by the author’s last name.
- The first two lines of the program should be the author’s name and the date that the assignment was written.
- The author is expected to follow the Style Guidelines.
- Numbers should be formatted to one decimal place of accuracy.
- All columns of text output should be neatly aligned.

5.1 Code Refactor (updated)

Your code should be refactored to improve its readability and re-usability. The main operations (Design a New Ship and Test a Ship Design) should be re-written as functions. The menu in the **main** function should call the appropriate function.

You should include a prototype for each function that you write other than **main**. The **main** function should be the first function definition in your program.

Update for version 2.2:

- The Test a Ship Design function should be split into two functions.
- Use one function to read the ship file and store the values in a Ship variable (described in 5.2).
- Use a second function to run the tests and print the results.
- You can decide how the Ship is passed between the two functions. The first function could return a Ship or be passed a Ship by reference. But all file operations should be done in one function and all tests (described in another).

You can call both from from main or only call one of them.

Function Comments

You should also include thorough documentation in the form of comments for each function. These comments should define the use, parameters, and return values of each function.

//Design a Ship
//Write one ship file after taking user input
//Parameters: None
//Returns: Nothing
//Side Effects: writes one file to disk
void designShip();

5.2 Custom Data Types

To improve the performance and readability of your code, add a custom data type using a **struct.** Give this **struct** the name **Ship**. The **Ship** structure should have the following members:

- **string shipname**
- **string designerName**
- **(optionally designer name can be stored as two strings)**
- **float mass**
- **float thrust**
- **float acceleration**
- **float impulse**
- **float maxAltitude**

Every function should be updated to incorporate this new datatype. “Design a New Ship” should save every piece of user input in the members of a **Ship** as it is being input by the user and then write those values to file. “Test a Ship Design” should read the values from the ship file into the members of a **Ship**, and should perform the 5 tests using the members of this structure.

“Simulate a Flight” does not need to use a Ship **struct.**

You can pass **Ships** to your functions as parameters but are not required to.

6. Sample Test Cases:

Input
```
d
Robert Goddard
Zoomer1
10000
1000
```
Console
```
File written for Zoomer1
```
Zoomer1.txt
```
##########Zoomer1##########

Ship Mass:          1000.0kg
Acceleration:       10.0m/s2
Impulse:            22.6Ns/kg
Max Altitude:       461.3km

File saved at:      1620926125
Deigned By:         Robert Goddard
```

Input
```
d
Sergei Korolev
Boko
200
250
```
Console
```
File written for Boko
```
Boko.txt
```
##########Boko##########

Ship Mass:          200.0kg
Acceleration:       1.2m/s2
Impulse:            2.2Ns/kg
Max Altitude:       53.7km

File saved at:      1620956317
Designed by:        Sergei Korolev
```

Input
```
T
Zoomer1
```
Console
```
Zoomer1 has failed Test 2
```

Input
```
T
Boko
```
Console
```
Boko has passed all tests.
```

Input
```
s
1
9
```
Console
```
At 0 seconds:  0.0m/s  0.0m
At 1 seconds:  1.0m/s  0.5m
At 2 seconds:  2.0m/s  2.0m
At 3 seconds:  3.0m/s  4.5m
At 4 seconds:  4.0m/s  8.0m
At 5 seconds:  5.0m/s  12.5m
At 6 seconds:  6.0m/s  18.0m
At 7 seconds:  7.0m/s  24.5m
At 8 seconds:  8.0m/s  32.0m
At 9 seconds:  9.0m/s  40.5m
```

7. Guidance

- The **<iomanip>** library can be used to set the precision of decimal numbers using the **fixed** and **setprecision** keywords.
- The log function from the <**cmath**> library can be used to calculate a natural log.
- The function time(NULL) from the <**ctime**> can be used to produce a timestamp formatted in
- computer time. Printing a timestamp on the screen would look like:
    - **cout << time(NULL) << endl;**
- When reading a text file, you will need to discard some string values.
- You will need to use a loop to keep the program running until the user enters ‘q’ or ‘Q’ as a menu option.