

Programs in C and Documentation



PRIFYSGOL CYMRU

Y Drindod Dewi Sant

UNIVERSITY OF WALES

Trinity Saint David

Siobhan Kelly Stephens

19/02/2019

Software Development

School of Applied Computing

Table of Contents

1. Discussion.....	3
1.1. Variables	4
1.2. Loops.....	5
1.3. Arrays.....	6
2. Exercise 1	8
2.1. Problem	9
2.1. Analysis of Solution.....	11
2.2. Annotated C Code Listing.....	13
2.3. Testing.....	15
2.4. Limitations of the Solution.....	25
3. Exercise 2	26
3.1. Problem	26
3.2. Analysis of Solution.....	28
3.3. Annotated C Code Listing.....	31
3.4. Testing.....	34
3.5. Limitations of the Solution.....	52
4. Exercise 3	53
4.1. Problem	53
4.2. Analysis of Solution.....	55
4.3. Annotated C Code Listing.....	57
4.4. Testing.....	59
4.5. Limitations of the Solution.....	68
5. Exercise 4	69
5.1. Problem	69
5.2. Analysis of Solution.....	70
5.3. Annotated C Code Listing.....	72
5.4. Testing.....	75
5.5. Limitations of the Solution.....	86
6. Reference List.....	87

Part A

1. Discussion

1.1. Variables

What is a variable and why are they important in writing programs?

A variable is a name given to a value used in a program to represent data of different types, such as integers, strings, booleans and arrays. During compilation or interpretation, variables correlate to memory locations that store the actual value, which may change numerous times during the execution of the program [1]. Variables can be accessed in high-level programming languages by calling them by their name, which is case-sensitive.

Unlike variables, constants should be used to store values which do not change during the course of the program, an example of this is defining a float with the value 0.2 to represent VAT, or 3.14159 for PI [2].

There are various different naming conventions to define variables, including camel case, which use title case beginning with a lower-case letter such as 'triangleAreaOne', or separating words using underscores, such as 'triangle_area_one'. The first character of a variable must only contain alpha characters or underscores and should then follow a suitable convention. Constants are often declared in CAPITAL letters.

Sensible and consistent naming conventions are of high importance for many reasons. Firstly, variables must have suitable names that describe their type, use or purpose in order for other programmers to identify the purpose of the variable easily. Secondly, variables must follow a consistent naming convention so that it can be followed and used by other programmers, in the case of collaboration or future maintenance [3].

Variables should be initialised before use, such as during declaration, to ensure they contain expected values. If not initialised, errors can occur in the program where variables point to memory locations that contain existing data.

Variables are essential to programming for many reasons. One, is that they allow for user input during execution, which can be manipulated in calculations etc. Variables allow for a specific memory location value to be altered during run-time such as for calculations to be performed and their value stored in its place, therefore allows programmers to create adaptable programs.

Variables are very useful when used as parameters and returns from functions, and this allows programmers to modularise their code for readability and re-usability. Another important use is using variables as environment variables, for example setting a mode for a program, such as an integer 0 for development mode, 1 for test mode, 2 for QA and 3 for production, all which set different variables throughout the code based on the value of test mode.

1.2. Loops

Explain why the for-loop can also be referred to as a counting-loop and give an example of when you would use it.

The for loop iterates through a block of code a known number of times and should be used when a programmer/program will know the number of iterations that must be made. The for-loop uses an incrementing variable, a counter often named '*i*', to iterate through the loop, therefore can be referred to as a counting-loop. The loop counter changes with each iteration, often increasing by 1. The loop must include a terminating condition, such as *i* < 10, so that outer program execution can continue.

An example use of a for-loop would be for printing arrays, where the length of the array is the terminating condition. Pseudocode for this is below.

```
For i = 0, i < length(array), i = i + 1:
    Print(array[i])
```

In the above example, *i* is initialised as 0, and the terminating condition states that the for loop will execute if *i* is less than the length of the array. If the array length was 15, the loop would iterate 15 times, terminating when the value of *i* is 14, as *i* begins at 0.

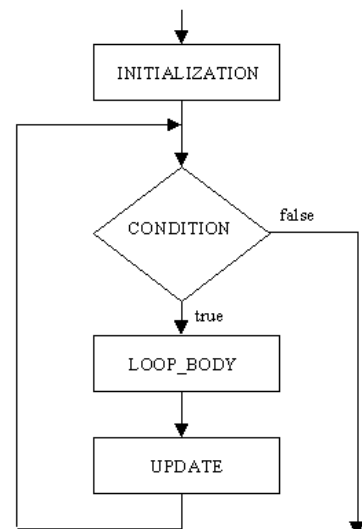


Figure 1. Flowchart of for-loop

Another example use of a for-loop is to sort data in an array, such as by using a bubble sort. The bubble sort uses nested-for-loops to iterate through each index and swap the values using a temporary variable until all values are sorted [4].

1.3. Arrays

Explain what an array is, what their features are, and give an example of when you would use one.

An array is a data structure that contains a group of elements of the same data type [5]. Items in an array are accessed using indexes, beginning from 0 upwards. An example of a 1D array that stores the first names of 4 employees is:

```
String array_firstnames[4] = {"Jim","John","Peter","Shaun"}
array_firstnames[0] = "Jim"
array_firstnames[1] = "John"
array_firstnames[2] = "Peter"
array_firstnames[3] = "Shaun"
```

2D and 3D arrays are commonly used to store related data. A 1D array may be thought of as being similar to a list, whereas a 2D array may be thought of as being similar to a table.

An example declaration of a 2D array, showing the sales for each week number is:

```
Float employee_sales[3][2] = {{ "32", "245.06"}, {"33", "568.76"}, {"34", "678.21"}}
```

This could also be displayed as:

Table 1. 2D array visual example

	$j = [0]$	$j = [1]$
	Week no	Sales amount
$i = [0]$	32	245.06
$i = [1]$	33	568.78
$i = [2]$	34	678.21

2D arrays are accessed using nested for-loops to access both rows and columns. The loop incrementors i and j are used to access the elements of the array. An example pseudocode of how to print the elements in a 2D array is below:

```
for (i = 0, i < array.rows, i++) //loop through rows
```

```
        for(j = 0, j < array.columns, j++) //loop through columns
            print(array[i][j] //print element at position [i][j]
        end for
    end for
```

Arrays are often used for sorting and searching, for example, a 2D array could be used to store and then sort the scores for players in a game, by passing the 2D array to a recursive quicksort function. A 1D array could be used to store a sorted list of permitted values and could be searched through using a binary search in order to find a certain value.

Arrays are useful where a programmer needs to gather data from a file and store the data in memory for use in the program, such as gathering data from a database, storing the data into a 2D array and then manipulating the values, such as to perform calculations on specific columns and output those columns and rows to the user.

Part B

1. Development and testing completed using the IDE Xcode (on Mac OSX 10.14)
2. Testing evidence consists of screenshots of the input/output from the CLI in Xcode

2. Exercise 1

Show how variables can contain different types of values by declaring three variables that contain a name, a measure of height (e.g. 1.81cm) and a person's age (e.g. 32). Input these values and display them on the screen.

2.1. Problem

The program must allow the user to enter three values, one for name, one for height, and one for age. It must then print these values back to the user.

In programming, there are five primary data types:

- Integer – which stores positive or negative whole numbers such as 5678
- Real – which stores decimal and integer values such as 366.77543
- Character – which stores a single character, such as a letter, number or symbol, for example “A”
- String – a string stores alphanumeric text, a combination of many characters, such as ‘Cat’
- Boolean – a Boolean can store only two values, True or False, which are represented as 0 for False, and 1 for True.

In C however, the primary data types consist of Integer, Float (real), Character and Void.

In order to declare variables of different types, it must first be determined which data type is suitable for a value. In the given problem, it is suggested that a name, measure of height and age should be declared as variables. Below are samples of data relating to these items.

Table 2. Sample data for Exercise 1

Item	Sample Data
Name	Jim, John, Peter, Gareth, Chloe, Jeff, Milly
Measure of Height (in cm)	146.22, 168.5, 170, 172.55, 120
Age (in years)	12, 27, 42, 39, 55, 70, 98, 1

Using the sample data, it was decided that the following data types should be used.

Table 3. Data types chosen for exercise 1

Item	Data Type	Data Type in C
Name	String	Char (array of chars)
Measure of Height (in cm)	Real	Float
Age (in years)	Integer	Integer

As C does not have a string data type, it is suitable that an array of characters should be used to store the value of a name.

Simple input validation to check the input ranges of both age and height will be implemented, as shown in table 3.

Table 4. Validation rules for data input, exercise 1

Field	Data Type	Validation Rule
Name	Char []	N/a
Measure of Height (in cm)	Float	Range check: 50 - 300
Age (in years)	Integer	Range check: 1 - 110

Table 5. Examples of valid, invalid and erroneous data

Field	Valid	Invalid	Erroneous
Name	Bob	\$\$%	N/A
Measure of Height (in cm)	167.56	-1, -999, 301, 9999	A, %, \$ etc
Age (in years)	45	Range check: 1 - 110	A, ^, “ etc

2.1. Analysis of Solution

2.1.1. Problem Break Down

- a. Declare three variables: *Name*, *height* and *age*
- b. Initialise variable *name* to "", *height* to 0.00, and *age* to 0
- c. Ask user for input:
Print("Please enter your name.")
- d. User Input(name) and assign to variable *name*
- e. Print("Please enter your age.")
- f. Input(age) and assign to variable *age*
- g. Validate age while invalid ($\text{age} < 0$ and > 110)
 - a. If invalid: Print("Invalid, try again, please enter your age")
 - b. Input(age)
- h. Print("Please enter your height.")
- i. Input(height) and assign to variable *height*
- j. Validate height while invalid ($\text{height} < 50$ and > 300)
 - a. If invalid: Print("Invalid, try again, please enter your height")
 - b. Input(height)
- k. Print("Your name is " + name)
- l. Print("Your age is " + age)
- m. Print("Your height is " + height)

2.1.2. Pseudocode

```
Procedure is_range(low, high, input)
    If input < low or input > high
        Return false
    Else
        Return true
end

Procedure print_name()
    declare character name[] = ""
    declare integer age = 0
    declare float height = 0.00

    print("Please enter your name.")
    input(name)
    print("Please enter your age.")
    input(age)

    while is_range_int(1,110,age) == 0:
        print("Invalid age. Please enter your age.")
        input(age)

    print("Please enter your height.")
    input(height)
    while is_range_int(50,300,height) == 0:
        print("Invalid height. Please enter your height.")
        input(height)

    print("Your name is: " + name)
    print("Your age is: " + age)
    print("Your height is: " + height)
end

Function main
    print_name()
End
```

2.2. Annotated C Code Listing

```

#include <stdio.h>
//Declare prototype for procedures and functions
//Including their return types, parameters and parameter types
void input_print_variables(void);
int is_range_int(int low, int high, int input);
int is_range_float(int low, int high, float input);

//is_range_int is used to validate whether an input is within a specified range
//it uses parameters low for the minimum of the range,
//high for the maximum of the range, and input for the comparison
int is_range_int(int low, int high, int input){
    //if input is less than low or input is greater than high
    if(input < low || input > high){
        //return false
        return 0;
    }
    //otherwise, if the input is valid (within range), return true
    return 1;
}
//is_range_float follows the same principles as above however takes a different
data type for the input parameter
int is_range_float(int low, int high, float input){
    if(input < low || input > high){
        return 0;
    }
    return 1;
}
void input_print_variables(){
    //Declare and initialise variables to be used
    char name[] = {""}; //declare char array
    int age = 0; //declare and initialise integer to represent age
    float height = 0.00; //declare and initialise float to represent height

    printf("-----\n"); //'\n' is used to print a new line
    printf("Please enter your first name.\n"); //prompt user to input a name
    scanf("%s",&name); //scanf assigns the input value to the variable name

    printf("-----\n"); //used to neaten look of program in cli
    printf("Please enter your age.\n"); //prompt user to input an age
    scanf("%d",&age); //assign input value to the variable age

    //use return of is_range_int function
    //if return true - it is within the range, if false, it is not within the range
    while(is_range_int(1,110,age) == 0){
        printf("Invalid age, try again.\nPlease enter your age.\n"); //prompt user
to input an age
        scanf("%d",&age); //assign input value to the variable age
    }
    printf("-----\n");
    printf("Please enter your height.\n"); //prompt user to input height
    scanf("%f",&height); //assign input value to the variable height

    //use return of is_range_float function
    //if return true - it is within the range, if false, it is not within the range

```

```
    while(is_range_float(50,300,height) == 0){//validation to ensure height is
sensible
        printf("Invalid height, try again.\nPlease enter your height.\n");//prompt
user to input an age
        scanf("%f",&height);///assign input value to the variable height
    }
    printf("-----\n");

    printf("The name entered is: %s\n",name);//%s used to print a string
    printf("The age entered is: %d\n", age);//%d used to print integers
    printf("The height entered is: %.2f\n",height);//%.2f prints a float to two
decimal places
    printf("-----\n");

}
int main(){
    input_print_variables();//Call procedure input_print_variables
    return 0;
}
```

2.3. Testing

2.3.1. Test Plan

The test plan will outline tests to be performed on the system to ensure the system accepts valid inputs and produces the expected output. These will test against three main types of data: valid, invalid, and erroneous, plus both valid and invalid boundary data.

Test #	Test	Purpose	Test Data	Expected Result
1.1	Enter <u>valid data</u> for First Name, Age and Height	Ensure the acceptance of valid data	1.A	Pass
1.2	Enter <u>valid data</u> for First Name, Height, and erroneous data for Age	Ensure system does not accept erroneous data	1.B	Fail
1.3	Enter <u>valid data</u> for First Name, Age, and invalid data for Height	Ensure system can handle invalid data	1.C	Pass, prompt user to enter new data
1.4	Enter <u>valid boundary</u> data for Age	Ensure the acceptance of valid data	1.D	Pass
1.5	Enter <u>valid boundary</u> data for Height	Ensure the acceptance of valid data	1.E	Pass
1.6	Enter <u>invalid boundary</u> data for Age	Ensure system can handle invalid data	1.F	Pass, prompt user to enter new data s
1.7	Enter <u>invalid boundary</u> data for Height	Ensure system can handle invalid data	1.G	Pass, prompt user to enter new data
1.8	Enter <u>valid data</u> for First Name, Age, and erroneous data for Height	Ensure system does not accept erroneous data	1.H	Fail

2.3.2. Test Data

Test Data 1.A for Test 1.1

Field	Value	Type	Type of Data
First Name	John	String (char array)	Valid
Age	20	Integer	Valid
Height	179.52	Float	Valid

Test Data 1.B for Test 1.2

Field	Value	Type	Type of Data
First Name	John	String (char array)	Valid
Age	A	Integer	Erroneous
Height	168.50	Float	Valid

Test Data 1.C for Test 1.3

Field	Value	Type	Type of Data
First Name	John	String (char array)	Valid
Age	19	Integer	Valid
Height	-145.78	Float	Invalid

Test Data 1.D for Test 1.4

Field	Value	Type	Type of Data
First Name	John	String (char array)	Valid
Age	110	Integer	Valid Boundary
Height	168.50	Float	Valid

Test Data 1.E for Test 1.5

Field	Value	Type	Type of Data
First Name	John	String (char array)	Valid
Age	30	Integer	Valid
Height	300.00	Float	Valid Boundary

Test Data 1.F for Test 1.6

Field	Value	Type	Type of Data
First Name	John	String (char array)	Valid
Age	0	Integer	Invalid Boundary
Height	168.50	Float	Valid

Test Data 1.G for Test 1.7

Field	Value	Type	Type of Data
First Name	John	String (char array)	Valid
Age	1	Integer	Valid
Height	49.99	Float	Invalid Boundary

Test Data 1.H for Test 1.8

Field	Value	Type	Type of Data
First Name	Peter	String (char array)	Valid
Age	45	Integer	Valid
Height	Bob	Float	Erroneous

2.3.3. Test Runs and Results

Test 1.1

Evidence:

```
-----  
Please enter your first name.  
John  
-----  
Please enter your age.  
20  
-----  
Please enter your height.  
179.52  
-----  
The name entered is: John  
The age entered is: 20  
The height entered is: 179.52  
-----  
Program ended with exit code: 0|
```

Figure 2. Input and Output / Result for Test 1.1

Result:

Pass: The program accepted valid data as expected, and outputted the correct values as inputted by the users.

Test 1.2

Evidence:

[illegible]

Figure 3. Input and Output / Result for Test 1.2

Result:

Fail: The program did not continue, as it was expecting an integer value, and therefore would not accept the erroneous data 'A'. This caused it to enter an infinite loop and crash.

Test 1.3

Evidence:

```
-----  
Please enter your first name.  
John  
-----  
Please enter your age.  
19  
-----  
Please enter your height.  
-145.78  
Invalid height, try again.  
Please enter your height.
```

Figure 4. Input and Output / Result for Test 1.3

Result:

Pass: The program caught the invalid height data (out of permitted range) and prompted the user to input valid data as expected.

Test 1.4.

Evidence:

```
-----  
Please enter your first name.  
John  
-----  
Please enter your age.  
110  
-----  
Please enter your height.  
168.50  
-----  
The name entered is: John  
The age entered is: 110  
The height entered is: 168.50  
-----  
Program ended with exit code: 0|
```

Figure 5. Input and Output / Result for Test 1.4

Result:

Pass: The valid boundary data was accepted as expected, and the correct values were outputted.

Test 1.5

Evidence:

```
-----  
Please enter your first name.  
John  
-----  
Please enter your age.  
30  
-----  
Please enter your height.  
300  
-----  
The name entered is: John  
The age entered is: 30  
The height entered is: 300.00  
-----  
Program ended with exit code: 0
```

Figure 6. Input and Output / Result for Test 1.5

Result:

Pass: The valid boundary data was accepted as expected, and the correct values were outputted.

Test 1.6

Evidence:

```
-----  
Please enter your first name.  
John  
-----  
Please enter your age.  
0  
Invalid age, try again.  
Please enter your age.  
|
```

Figure 7. Input and Output / Result for Test 1.6

Result:

Pass: The program caught the invalid boundary age data (out of permitted range) and prompted the user to input valid data as expected.

Test 1.7

Evidence:

```
-----  
Please enter your first name.  
John  
-----  
Please enter your age.  
0  
Invalid age, try again.  
Please enter your age.  
1  
-----  
Please enter your height.  
49.99  
Invalid height, try again.  
Please enter your height.  
|
```

Figure 8. Input and Output / Result for Test 1.7

Result:

Pass: The program caught the invalid boundary height data (out of permitted range) and prompted the user to input valid data as expected.

Test 1.8

Evidence:

```
Invalid height, try again.  
Please enter your height.  
Invalid height, try again.  
Please enter your height.  
Invalid height, try again.  
Please enter your height.  
Invalid height, try again.  
Please enter your height.  
Invalid height, try again.  
Please enter your height.  
Invalid height, try again.  
Please enter your height.  
Invalid height, try again.  
Please enter your height.  
Invalid height, try again.  
Please enter your height.  
Invalid height, try again.  
Please enter your height.
```

Figure 9.. Input and Output / Result for Test 1.8

Result:

Fail: The program did not continue, as it was expecting an integer value, and therefore would not accept the erroneous data 'Bob'. This caused it to enter an infinite loop and crash.

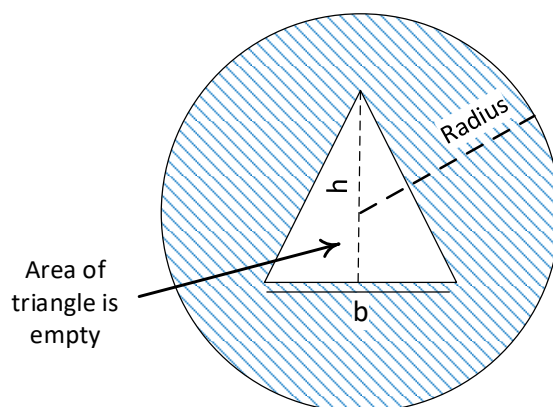
2.4. Limitations of the Solution

The solution does not currently validate against the input of data of conflicting types, such as the input of a char (character) where a float or integer is expected. This currently causes the program to crash and enter into an infinite loop, as evidenced in test 1.2 and 1.8.

Future improvements to the program would be to implement input validation to check the data type of the input, and handle it accordingly, by catching the error and prompting the user to enter valid data. This would improve the user experience as the program would continue its' execution and would simply ask for valid input data.

3. Exercise 2

Allow the user to enter the dimensions of shapes shown below, then calculate the area of the blue shaded part. Note: The triangular area in the centre is empty.



3.1. Problem

The program must allow the user to enter three values: circle radius, triangle base and triangle height. The program must then calculate the area of the circle using the following equation:

$$A = \pi r^2$$

$$PI = 3.14159265359$$

It must calculate the area of the triangle using the following equation:

$$A = (b * h) \div 2$$

In order to calculate the area of the shaded section, the program must subtract the area of the triangle from the area of the circle.

Below is a table of sample data that could be used to calculate the section area.

Table 6. Sample data for input, exercise 2

Item	Sample Data
Circle radius	45.67, 527.99, 76.00, 1.23, 891.01
Triangle height	45.67, 527.99, 76.00, 1.23, 891.01
Triangle base	45.67, 527.99, 76.00, 1.23, 891.01

Using the sample data, it was decided that the following data types should be used.

Table 7. Data types chosen for exercise 2

Item	Data Type	Data Type in C
Circle radius	Float	Float
Triangle height	Float	Float
Triangle base	Float	Float

The program must validate user input, to ensure the data entered is within a sensible range and must ensure that the triangle base and height are smaller than the circle diameter. The tables below show the validation that will be implemented, and what data is expected from the user.

Table 8. Validation rules for user input, exercise 2

Field	Data Type	Validation Rule
Circle Radius	Float	Range check: > 0 and < 999999
Triangle Base	Float	Range check: > 0 and < 999999
Triangle Height	Float	Range check: > 0 and < 999999

Table 9. Valid, invalid and erroneous data for exercise 2

Field	Valid	Invalid	Erroneous
Circle Radius	20.45	-1, -999, 0, 100000	A, %, \$ etc
Triangle Base	25.67	-1, -999, 0, 100000	A, %, \$ etc
Triangle Height	22.78	-1, -999, 0, 100000	A, %, \$ etc

3.2. Analysis of Solution

3.2.1. Problem break down

- a. Declare three variables: *Circle radius*, *triangle height* and *triangle base*
- b. Initialise variable *Circle radius* to 0.00, *triangle height* to 0.00, *triangle base* to 0.00
- c. Print("Please enter circle radius")
- d. Input(circle_radius)
- e. Validate circle_radius while invalid (value < 0 and > 999999)
 - a. Print("invalid data, please try again.")
 - b. Input(circle_radius)
- f. Print("Please enter triangle base")
- g. Input(triangle_base)
- h. Validate triangle_base while invalid (value < 0 and > 999999) and (value > circle diameter)
 - a. Print("invalid data, please try again.")
 - b. Input(triangle_base)
- i. Print("Please enter triangle height")
- j. Input(triangle_height)
- k. Validate triangle_height while invalid (value < 0 and > 999999) and (value > circle diameter)
 - a. Print("invalid data, please try again.")
 - b. Input(triangle_height)
- l. Calculate circle area
- m. Calculate triangle area
- n. Calculate section area
- o. Print("area of section is" + section area)

3.2.2. Pseudocode

Constant PI = 3.14159265359

```
function is_range_float(low, high, input)
    if(input <= low || input > high)
        return false
    end if
    return true
end

function calculate_circle_area(circle_radius)
    real area_circle = 0.00
```

```
        area_circle = PI * (circle_radius * circle_radius)
        print("Circle radius is " + circle_radius)
        print("Area of circle is " + area_circle)

        return area_circle
end

function calculate_triangle_area(triangle_base, triangle_height)
    real area_triangle = 0.00
    area_triangle = (triangle_base * triangle_height) / 2
    print("Triangle base is " + triangle_base)
    print("Triangle height is " + triangle_height)
    print("Area of triangle is " + area_triangle

    return area_triangle
end

procedure calculate_section_area()
    real circle_radius = 0.00
    real triangle_base = 0.00
    real triangle_height = 0.00
    real triangle_area = 0.00
    real circle_area = 0.00
    real section_area = 0.00

    print("Enter the outer circle radius.")
    input(circle_radius)

    while(is_range_float(0,999999,circle_radius) == 0)
        print("Invalid circle radius. Please try again.")
        input(circle_radius)
    end while

    print("Enter the inner triangle base.")
    input(triangle_base)

    while(is_range_float(0,999999,triangle_base) == 0)
        print("Invalid triangle base. Please try again.")
        input(triangle_base)
    end while

    while(triangle_base > (circle_radius * 2))
        print("Invalid triangle base. Please try again.")
        input(triangle_base)
    end while

    print("Enter the inner triangle height.")
    input(triangle_base)

    while(is_range_float(0,999999,triangle_height) == 0)
        print("Invalid triangle height. Please try again.")
        input(triangle_height)
    end while

    while(triangle_height > (circle_radius * 2))
        print("Invalid triangle height. Please try again.")
        input(triangle_height)
    end while
    triangle_area = calculate_triangle_area(triangle_base,triangle_height)
    circle_area = calculate_circle_area(circle_radius)
    section_area = circle_area - triangle_area;
```

```
        print("The area of the shaded section is " + section_area)
end

function main(){
    calculate_section_area()
}
```

3.3. Annotated C Code Listing

```

#include <stdio.h>
//define constant PI
#define PI 3.14159265359
//declare prototype functions to be used
float calculate_triangle_area(float, float);
float calculate_circle_area(float);
void calculate_section_area(void);

int is_range_float(int low, int high, float input);

//is_range_float is used to validate whether an input is within a specified range
//it uses parameters low for the minimum of the range, high for the maximum of the range, and
input for the comparison
int is_range_float(int low, int high, float input){
    //if input is less than or equal to low, or input is greater than height, return 0
    (false)
    if(input <= low || input > high){
        return 0;
    }
    //otherwise return true
    return 1;
}

//calculate_triangle_area function is used to calculate the area of a triangle
//it takes two float parameters, the triangle base and height
//the area is calculated using the equation triangle-area = (b * h) / 2
float calculate_triangle_area(float triangle_base, float triangle_height){
    float area_triangle = 0.00; //declare and initialise variable to store triangle area
    area_triangle = (triangle_base * triangle_height) / 2; //calculate area from parameter
values
    printf(" - Triangle base is %.3f.\n - Triangle height is %.3f.\n - Therefore the area of
the triangle is %.3f.\n", triangle_base, triangle_height, area_triangle); //print result, for
testing purposes
    return area_triangle; //return the result, float area_triangle
}

//calculate_circle_area is used to calculate the area of a circle
//The function takes one parameter, circle_radius
//The area is calculated using the equation a = PI * (radius^2)
float calculate_circle_area(float circle_radius){
    float area_circle = 0.00; //declare and initialise variable to store circle area
    area_circle = PI * (circle_radius * circle_radius); //calculate area from parameter value
and constant PI
    //print result for testing
    printf(" - Circle radius is %.3f.\n - Therefore the area of the circle is
%.3f.\n", circle_radius, area_circle);
    return area_circle; //return the result, float area_circle
}

void calculate_section_area(){
    //declare and initialise float variables to 0.00
    float circle_radius = 0.00;
    float triangle_base = 0.00;
    float triangle_height = 0.00;
    float triangle_area = 0.00;
    float circle_area = 0.00;
    float section_area = 0.00;
    printf("- - - - -\n Calculate area of section (exercise
2)\n- - - - -\n");
    //prompt user to enter the outer circle radius
    printf("1. Enter the outer circle radius:\n");
    printf("<          r----->\n");

    scanf("%f", &circle_radius);

```

```

//validate user input - circle radius must be greater than 0 and less than 999999
while(is_range_float(0,999999, circle_radius)==0){
    //if invalid, ask for new input
    printf("*Invalid circle radius, must be greater than 0 and less than 999999, try
again.*\n");
    printf("1. Enter the outer circle radius:\n");
    printf("<          r----->\n");
    scanf("%f",&circle_radius);
} //continue to ask for input while the circle_radius is invalid

//prompt user to enter the inner triangle base
printf("2. Enter the inner triangle base:\n");
printf("<-----b----->\n");
scanf("%f",&triangle_base);
/** input validation
//1. check that triangle_base is greater than 0 and less than 999999
while(is_range_float(0,999999, triangle_base)==0){

    //if invalid, ask for new input
    printf("*Invalid triangle base, must be greater than 0 and less than 999999, try
again.*\n");
    printf("2. Enter the inner triangle base:\n");
    printf("<-----b----->\n");
    scanf("%f",&triangle_base);
} //continue to ask for input while the triangle base is invalid

//check that triangle_base is smaller than the circle diameter
while(triangle_base > (circle_radius * 2)){
    printf("Invalid: Inner triangle base must be less than circle diameter. Try
again.\n\n"); //print error message and prompt user to enter the value again
    printf("2. Enter the inner triangle base:\n");
    printf("<-----b----->\n");
    scanf("%f",&triangle_base);
}
//prompt user to enter the inner triangle height
printf("3. Enter the inner triangle height:\n");
printf("<          ^h          >\n");
scanf("%f",&triangle_height);

//input validation for input range - value must be greater than 0 and less than 999999
while(is_range_float(0,999999, triangle_height)==0){
    //if input invalid, ask for new input
    printf("*Invalid triangle height, must be greater than 0 and less than 999999, try
again.*\n");
    printf("3. Enter the inner triangle height:\n");
    printf("<          ^h          >\n");
    scanf("%f",&triangle_height);
}
/** input validation
//check that triangle height is smaller than the circle diameter
while(triangle_height > (circle_radius*2)){
    printf("Invalid: Inner triangle height must be less than circle diameter. Try
again.\n\n"); //print error message
    //prompt user to enter value again
    printf("3. Enter the inner triangle height:\n");
    printf("<          ^h          >\n");
    scanf("%f",&triangle_height);
}
//call calculate_triangle_area function with the parameters triangle_base and
triangle_height
//set the variable triangle area to be the return value of the call
triangle_area = calculate_triangle_area(triangle_base, triangle_height);
//follow same principle as above for circle area
circle_area = calculate_circle_area(circle_radius);

```

```
    //calculate the area of the section by subtracting the triangle area from the circle area
    section_area = circle_area - triangle_area;
    //print the result to the console
    printf("-----\n");
    printf("The area of the section is %.3f.\n",section_area);
    printf("-----\n");
}

int main()
{
    //call starting procedure
    calculate_section_area();
}
```

3.4. Testing

A test plan has been devised to test against three main types of data: valid, invalid, and erroneous, plus both valid and invalid boundary data. It will also test that the output produced by the program is accurate and what is expected.

3.4.1. Test Plan

Test #	Test	Purpose	Test data	Expected result
2.1	Enter <u>valid data</u> for Circle Radius, Triangle Base and Triangle Height	Ensure the acceptance of valid data	2.A	Pass, prompt user to enter new value
2.2	Enter <u>invalid data</u> for Circle Radius , and <u>valid data</u> for Triangle Base and Triangle Height	Ensure system can handle invalid data	2.B	Pass, prompt user to enter new value
2.3	Enter <u>invalid data</u> for Triangle Height , and <u>valid data</u> for Circle Radius and Triangle Base	Ensure system can handle invalid data	2.C	Pass, prompt user to enter new value
2.4	Enter <u>erroneous data</u> for Triangle Base , and <u>valid data</u> for Circle Radius and Triangle Height	Ensure system does not accept erroneous data	2.D	Fail
2.5	Enter <u>valid boundary data</u> for Circle Radius	Ensure the acceptance of valid data	2.E	Pass
2.6	Enter <u>valid boundary data</u> for Triangle Height	Ensure the acceptance of valid data	2.F	Pass
2.7	Enter <u>valid boundary data</u> for Triangle Base	Ensure the acceptance of valid data	2.G	Pass

2.8	Enter <u>invalid boundary</u> data for Circle Radius	Ensure system can handle invalid data	2.H	Pass, prompt user to enter new value
2.9	Enter <u>invalid boundary</u> data for Triangle Height	Ensure system can handle invalid data	2.I	Pass, prompt user to enter new value
2.10	Enter <u>invalid boundary</u> data for Triangle Base	Ensure system can handle invalid data	2.J	Pass, prompt user to enter new value
2.11	Enter a triangle height greater than the diameter of the circle	Ensure system does not allow for this and prompts the user to input valid data	2.K	Pass, prompt user to enter new value
2.12	Ensure the correct areas are calculated	Ensure the system produces accurate output	2.L	Pass, output is accurate and is calculated correctly
2.13	Enter <u>erroneous data</u> for Triangle Height , and <u>valid data</u> for Circle Radius and Triangle Base	Ensure system does not accept erroneous data	2.M	Fail
2.14	Enter <u>erroneous data</u> for Circle Radius , and <u>valid data</u> for Triangle Base and Triangle Height	Ensure system does not accept erroneous data	2.N	Fail

3.4.2. Test Data

Test Data 2.A for Test 2.1

Field	Value	Type	Type of Data
Circle Radius	20	Float	Valid
Triangle Height	20	Float	Valid
Triangle Base	20	Float	Valid

Test Data 2.B for Test 2.2

Field	Value	Type	Type of Data
Circle Radius	-45	Float	Invalid
Triangle Height	20	Float	Valid
Triangle Base	20	Float	Valid

Test Data 2.C for Test 2.3

Field	Value	Type	Type of Data
Circle Radius	20	Float	Valid
Triangle Height	9999999999999999	Float	Invalid
Triangle Base	20	Float	Valid

Test Data 2.D for Test 2.4

Field	Value	Type	Type of Data
Circle Radius	25	Float	Valid
Triangle Height	30	Float	Valid
Triangle Base	A	Float	Erroneous

Test Data 2.E for Test 2.5

Field	Value	Type	Type of Data
Circle Radius	999999	Float	Valid Boundary
Triangle Height	30	Float	Valid
Triangle Base	12	Float	Valid

Test Data 2.F for Test 2.6

Field	Value	Type	Type of Data
Circle Radius	327	Float	Valid
Triangle Height	0.1	Float	Valid Boundary
Triangle Base	356	Float	Valid

Test Data 2.G for Test 2.7

Field	Value	Type	Type of Data
Circle Radius	327	Float	Valid
Triangle Height	351	Float	Valid
Triangle Base	0.01	Float	Valid Boundary

Test Data 2.H for Test 2.8

Field	Value	Type	Type of Data
Circle Radius	0	Float	Invalid Boundary
Triangle Height	56	Float	Valid
Triangle Base	356	Float	Valid

Test Data 2.I for Test 2.9

Field	Value	Type	Type of Data
Circle Radius	67	Float	Valid
Triangle Height	-1	Float	Invalid Boundary
Triangle Base	356	Float	Valid

Test Data 2.J for Test 2.10

Field	Value	Type	Type of Data
Circle Radius	67	Float	Valid
Triangle Height	89	Float	Valid
Triangle Base	100000	Float	Invalid Boundary

Test Data 2.K for Test 2.11

Field	Value	Type	Type of Data
Circle Radius	20	Float	Valid
Triangle Height	42	Float	Invalid (compared to radius)

Test Data 2.L for Test 2.12

Field	Value	Type	Type of Data
Circle Radius	85	Float	Valid
Triangle Height	65	Float	Valid
Triangle Base	52	Float	Valid

Test Data 2.M for Test 2.13

Field	Value	Type	Type of Data
Circle Radius	85	Float	Valid
Triangle Height	Pete	String	Erroneous
Triangle Base	52	Float	Valid

Test Data 2.N for Test 2.14

Field	Value	Type	Type of Data
Circle Radius	\$	String	Erroneous
Triangle Height	65	Float	Valid
Triangle Base	52	Float	Valid

3.4.3. Test Runs and Results

Test 2.1**Evidence:**

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
  <-----r----->  
20  
2. Enter the inner triangle base:  
  <-----b----->  
20  
3. Enter the inner triangle height:  
  <-----^h----->  
20  
- Triangle base is 20.000.  
- Triangle height is 20.000.  
- Therefore the area of the triangle is 200.000.  
- Circle radius is 20.000.  
- Therefore the area of the circle is 1256.637.  
-----  
The area of the section is 1056.637.  
-----  
Program ended with exit code: 0
```

*Figure 10. Input and Output / Result for Test 2.1***Result:**

Pass: The valid data was accepted as expected, and the correct values were outputted.

Test 2.2

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
-45  
*Invalid circle radius, must be greater than 0 and less than 999999, try again.*  
1. Enter the outer circle radius:  
<          r----->
```

Figure 11. Input and Output / Result for Test 2.2

Result:

Pass: The program caught the invalid circle radius (out of permitted range) and prompted the user to input valid data as expected.

Test 2.3

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
20  
2. Enter the inner triangle base:  
<-----b----->  
999999999999999  
*Invalid triangle base, must be greater than 0 and less than 999999, try again.*  
2. Enter the inner triangle base:  
<-----b----->
```

Figure 12. Input and Output / Result for Test 2.3

Result:

Pass: The program caught the invalid triangle base (out of permitted range) and prompted the user to input valid data as expected.

Test 2.4

Evidence:

[illegible]

Figure 13. Input and Output / Result for Test 2.4

Result:

Fail: The program did not continue, as it was expecting a float value, and therefore would not accept the erroneous data 'A'. This caused it to enter an infinite loop and crash.

Test 2.5

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
999999  
2. Enter the inner triangle base:  
<-----b----->  
30  
3. Enter the inner triangle height:  
<          ^h          >  
12  
- Triangle base is 30.000.  
- Triangle height is 12.000.  
- Therefore the area of the triangle is 180.000.  
- Circle radius is 999999.000.  
- Therefore the area of the circle is 3141586386944.000.  
-----  
The area of the section is 3141586386944.000.  
-----  
Program ended with exit code: 0|
```

Figure 14. Input and Output / Result for Test 2.5

Result:

Pass: The program allowed the valid boundary circle radius as expected and produced the correct outputs.

Test 2.6

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
327  
2. Enter the inner triangle base:  
<-----b----->  
0.1  
3. Enter the inner triangle height:  
<          ^h          >  
356  
- Triangle base is 0.100.  
- Triangle height is 356.000.  
- Therefore the area of the triangle is 17.800.  
- Circle radius is 327.000.  
- Therefore the area of the circle is 335927.375.  
-----  
The area of the section is 335909.562.  
-----  
Program ended with exit code: 0|
```

Figure 15. Input and Output / Result for Test 2.6

Result:

Pass: The program allowed the valid boundary triangle base as expected and produced the correct outputs.

Test 2.7

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
327  
2. Enter the inner triangle base:  
<-----b----->  
351  
3. Enter the inner triangle height:  
<          ^h          >  
0.01  
- Triangle base is 351.000.  
- Triangle height is 0.010.  
- Therefore the area of the triangle is 1.755.  
- Circle radius is 327.000.  
- Therefore the area of the circle is 335927.375.  
  
-----  
The area of the section is 335925.625.  
-----  
Program ended with exit code: 0|
```

Figure 16. Input and Output / Result for Test 2.7

Result:

Pass: The program allowed the valid boundary triangle height as expected and produced the correct outputs.

Test 2.8

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
0  
*Invalid circle radius, must be greater than 0 and less than 999999, try again.*  
1. Enter the outer circle radius:  
<          r----->
```

Figure 17. Input and Output / Result for Test 2.8

Result:

Pass: The program caught the invalid boundary circle radius (out of permitted range) and prompted the user to input valid data as expected.

Test 2.9

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
67  
2. Enter the inner triangle base:  
<-----b----->  
-1  
*Invalid triangle base, must be greater than 0 and less than 999999, try again.*  
2. Enter the inner triangle base:  
<-----b----->
```

Figure 18. Input and Output / Result for Test 2.9

Result:

Pass: The program caught the invalid boundary triangle base (out of permitted range) and prompted the user to input valid data as expected.

Test 2.10

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
67  
2. Enter the inner triangle base:  
<-----b----->  
89  
3. Enter the inner triangle height:  
<          ^h          >  
100000  
Invalid: Inner triangle height must be less than circle diameter. Try again.  
  
3. Enter the inner triangle height:  
<          ^h          >
```

Figure 19. Input and Output / Result for Test 2.10

Result:

Pass: The program caught the invalid boundary triangle height (out of permitted range) and prompted the user to input valid data as expected.

Test 2.11

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
20  
2. Enter the inner triangle base:  
<-----b----->  
42  
Invalid: Inner triangle base must be less than circle diameter. Try again.  
  
2. Enter the inner triangle base:  
<-----b----->
```

Figure 20. Input and Output / Result for Test 2.11

Result:

Pass: The program caught the invalid triangle base, which must be less than the circle diameter, and prompted the user to input valid data as expected.

Test 2.12

Evidence:

```
-----  
Calculate area of section (exercise 2)  
-----  
  
1. Enter the outer circle radius:  
<          r----->  
85  
2. Enter the inner triangle base:  
<-----b----->  
65  
3. Enter the inner triangle height:  
<          ^h          >  
52  
- Triangle base is 65.000.  
- Triangle height is 52.000.  
- Therefore the area of the triangle is 1690.000.  
- Circle radius is 85.000.  
- Therefore the area of the circle is 22698.008.  
-----  
The area of the section is 21008.008.  
-----  
Program ended with exit code: 0
```

Figure 21. Input and Output / Result for Test 2.12

Result:

Circle radius: 85

Triangle height: 65

Triangle base: 52

$$\begin{aligned}\text{Circle area} &= \text{PI} * (\text{radius}^2) \\ &= 3.14159265359 * (85^2) \\ &= 3.14159265359 * (7225) \\ &= 22698.008\end{aligned}$$

$$\begin{aligned}\text{Triangle area} &= (b * h) / 2 \\ &= (65 * 52) / 2 \\ &= (3380) / 2 \\ &= 1690\end{aligned}$$

$$\begin{aligned}\text{Section area} &= \text{Circle area} - \text{Triangle area} \\ &= 22698.008 - 1690\end{aligned}$$

= 21008.008

Pass: The program produces an accurate result for the section and outputs the result correctly.

Test 2.13

Evidence:

```
2. Enter the inner triangle base:
<-----b----->
*Invalid triangle base, must be greater than 0 and less than 999999, try again.*
2. Enter the inner triangle base:
<-----b----->
*Invalid triangle base, must be greater than 0 and less than 999999, try again.*
2. Enter the inner triangle base:
<-----b----->
*Invalid triangle base, must be greater than 0 and less than 999999, try again.*
2. Enter the inner triangle base:
<-----b----->
*Invalid triangle base, must be greater than 0 and less than 999999, try again.*
2. Enter the inner triangle base:
<-----b----->
*Invalid triangle base, must be greater than 0 and less than 999999, try again.*
2. Enter the inner triangle base:
<-----b----->
```

Figure 22. Input and Output / Result for Test 2.13

Result:

Fail: The program did not continue, as it was expecting a float value, and therefore would not accept the erroneous data 'Pete'. This caused it to enter an infinite loop and crash.

Test 2.14

Evidence:

```
$
*Invalid circle radius, must be greater than 0 and less than 999999, try again.*
1. Enter the outer circle radius:
<      r----->
*Invalid circle radius, must be greater than 0 and less than 999999, try again.*
1. Enter the outer circle radius:
<      r----->
*Invalid circle radius, must be greater than 0 and less than 999999, try again.*
1. Enter the outer circle radius:
<      r----->
*Invalid circle radius, must be greater than 0 and less than 999999, try again.*
1. Enter the outer circle radius:
<      r----->
*Invalid circle radius, must be greater than 0 and less than 999999, try again.*
1. Enter the outer circle radius:
<      r----->
*Invalid circle radius, must be greater than 0 and less than 999999, try again.*
1. Enter the outer circle radius:
```

Figure 23. Input and Output / Result for Test 2.14

Result:

Fail: The program did not continue, as it was expecting a float value, and therefore would not accept the erroneous data '\$'. This caused it to enter an infinite loop and crash.

3.5. Limitations of the Solution

The solution does not currently validate against the input of data of conflicting types, such as the input of a char (character) where a float or integer is expected. This currently causes the program to crash and enter into an infinite loop, as evidenced in test 2.4, 2.13 and 2.14.

Future improvements to the program would be to implement input validation to check the data type of the input, and handle it accordingly, by catching the error and prompting the user to enter valid data. This would improve the user experience as the program would continue its execution and would simply ask for valid input data.

4. Exercise 3

Write a program that allows the user to enter the price of 5 products into an array. Once all the prices are entered calculate the total price, and then calculate the total price plus 20% VAT.

4.1. Problem

The program must allow a user to input five real number values and store them in a 1D array of length 5. It must then keep a running sub-total of the values minus VAT, and once totalled, calculate the 20% VAT on top of the sub-total to calculate the final total.

The program should implement a for-loop to populate the array with values, and to keep a running sub-total. A constant should be used to store VAT (0.2), as it should not be changed during execution.

The VAT can be calculated using the following equation:

$$\text{Total (inc vat)} = \text{sub-total} + (\text{sub-total} * \text{VAT})$$

Variables must be used in order to store the values entered by the user, therefore it must be decided what data type each input should be. To do this, sample data has been used, as shown below.

Table 10. Sample data for input, exercise 3

Item	Sample Data
Product price	45.67, 527.99, 76.00, 1.23, 891.01, 3.99

Using the sample data, it was decided that the following data types should be used.

Table 11. Data types chosen for exercise 3

Item	Data Type	Data Type in C
Product price	Float	Float

User input should be sensible, therefore it should be validated. If found to be invalid, the program should ask the user for valid input. This process will occur until valid data has been input.

Table 12. Validation rules for user input, exercise 3

Field	Data Type	Validation Rule
Product price	Float	Range check: > 0 and <= 10000

Table 13. Valid, invalid and erroneous data for exercise 3

Field	Valid	Invalid	Erroneous
Product price	20.45, 567.89	-1, -999, 0, 10001	A, %, \$ etc

4.2. Analysis of Solution

4.2.1. Problem Break Down

- a. Declare constant $VAT = 0.2$
- b. Declare and initialise four variables: loop incrementor $i = 0$, $array_prices[5] = \{0.00, 0.00, 0.00, 0.00, 0.00\}$, $total = 0.00$, $sub-total = 0.00$
- c. For-loop while $i < 5$ (loop 5 times)
 - a. Ask user to input value input value i
 - b. Store input i into $array[i]$
 - i. Validate user input while $price < 0$ or > 10000
 1. If invalid, ask user for another value
 - ii. If valid:
 - c. Set $sub-total = sub_total + array[i]$
 - d. Print sub-total
 - e. Set $total = sub_total + (sub_total * VAT)$
 - f. Print total
- d. Print sub-total
- e. Print total

4.2.2. Pseudocode

Constant VAT = 0.2

```

function is_range_float(low, high, input)
    if(input <= low or input > high)
        return 0
    }
    return 1
end

procedure calculate_total()
    int i = 0
    real array_prices[5] = {0.00, 0.00, 0.00, 0.00, 0.00}
    real total = 0.00
    real sub_total = 0.00

    for (i = 0, i < 5, i = i + 1)

        print("Please enter the price of product {}, format 0.00." + (i+1))
        input(array_prices[i])

        while(is_range_float(0,10000,array_prices[i])==0)
            print("Product must be greater than 0 and less than or equal to
10,000. Try again")
            input(array_prices[i])

        end while

        sub_total = sub_total + array_prices[i];
        total = sub_total + (sub_total * VAT);

        print("Running sub-total minus VAT = {}" + sub_total)
        print("Running sub-total plus VAT = {}" + total)

    end for

    print("Sub-total minus VAT = {}" + sub_total)
    print("Final Total plus VAT = {}" + total)
end

function main()
    calculate_total()
end

```

4.3. Annotated C Code Listing

```
#include <stdio.h>
//declare constant VAT (UK VAT at 20% or 0.2)
#define VAT 0.20

//declare prototype function calculate_total, get_float_input and is_range_float
void calculate_total(void);
float is_range_float(float high, float low, float input);
//is range function is used to validate user input by comparing the input to a low and high
value
//if the input value is less than or equal to low, or greater than high, return false
//otherwise, if the input value is within the permitted range, return true
float is_range_float(float low, float high, float input){
    if(input <= low || input > high){
        return 0;
    }
    return 1;
}

//calculate total procedure used to gather user input on product prices,
//store the prices in an array (array length 5 for 5 inputs),
//calculate the sub-total of the prices minus vat
//calculate the sub-total of the prices plus vat
//and then calculate the final totals (sub-total minus vat and total plus vat)
void calculate_total(){
    //declare and initialise variables
    int i = 0; //declare loop incrementor i
    //declare and initialise float array of length 5
    float array_prices[5] = {0.00, 0.00, 0.00, 0.00, 0.00};
    //declare and initialise total and sub-total variables
    float total = 0.00;
    float sub_total = 0.00;

    //for-loop used to iterate through array
    //condition i < 5 will iterate using loop incrementor i while i is less than 5 (0,1,2,3,4
    // = 5 iterations)
    //i++ means that i increments by one on each iteration (shorthand for i = i + 1)
    for (i = 0; i < 5; i++){

        //print prompt for user to enter the price of the nth product
        //i+1 value is used as we (humans) count from 1 upwards, but array indexes begin at
        0,

        //this means that if i = 0, we are on the 1st product (i (0) + 1 = 1)
        printf("- - - - - \n");
        printf("Please enter the price of product %d (in format 0.00):\n", i+1);
        //assign user input to array_prices[i] (where i is the index)
        //call get_float_input function to gather user input
```

```

scanf("%f",&array_prices[i]);

//---validate user input for sensible data ---
//call is_range_float function with parameters 0 for low, 10000 for high and
array_prices[i] for input
//if the function returns false, the value is not within the permitted range,
therefore ask for new input
while(is_range_float(0,10000,array_prices[i])==0){
    //display error message to user
    printf("Product prices must be greater than £0 and less than or equal to £10,000.
Please try again.\n");
    //prompt user for new input
    printf("Please enter the price of product %d (in format 0.00):\n", i+1);
    scanf("%f",&array_prices[i]);

}
//set variable sub total to be sub_total plus the value of array_index[i]
sub_total = sub_total + array_prices[i];
//set variable total to be the sub_total plus 20% VAT using constant VAT
total = sub_total + (sub_total * VAT);
//print the running sub total (minus vat) on each iteration
printf(" - Running sub-total minus vat = £%.2f...\n",sub_total);
//print the running total on each iteration
printf(" - Running sub-total plus vat = £%.2f...\n\n",total);

}

printf("\n-----\n\n");
printf("Sub-total minus VAT is: £%.2f.\n",sub_total);
printf("Final total inc VAT is: £%.2f.\n",total);
printf("\n-----\n\n");
}

int main(int argc, const char * argv[]) {
    calculate_total();//call calculate total function
    return 0;
}

```

4.4. Testing

A test plan has been devised to test the system against three main types of data: valid, invalid, and erroneous, plus both valid and invalid boundary data. The tests will also test that the calculations performed by the program produce accurate output, and outputs are displayed correctly to the user.

4.4.1. Test Plan

Test #	Test	Purpose	Test data	Expected result
3.1	Enter <u>valid data</u> for Product	Ensure the acceptance of valid data	3.A	Pass
3.2	Enter <u>erroneous data</u> for Product	Ensure system does not allow erroneous data	3.B	Fail
3.3	Enter <u>valid boundary data</u> for Product	Ensure the acceptance of valid data	3.C	Pass
3.4	Enter <u>invalid boundary data</u> for Product	Ensure system can handle invalid data	3.D	Pass
3.5	Enter <u>invalid extreme data</u> for Product	Ensure system can handle invalid data	3.E	Pass
3.6	Enter <u>invalid extreme data</u> for Product	Ensure system can handle invalid data	3.F	Pass
3.7	Ensure system calculates correct sub-total and total in VAT	Ensure system can produce accurate outputs	3.G	Subtotal = 19.95 Total = 23.94
3.8	Ensure system calculates correct sub-total and total in VAT	Ensure system can produce accurate outputs	3.H	Subtotal = 5 Total = 6
3.9	Ensure system calculates correct sub-total and total in VAT	Ensure system can produce accurate outputs	3.I	Subtotal = 38885 Total = 46662

4.4.2. Test Data

Test Data 3.A for Test 3.1

Field	Value	Type	Type of Data
Product	3.99	Float	Valid

Test Data 3.B for Test 3.2

Field	Value	Type	Type of Data
Product	A	Char	Erroneous

Test Data 3.C for Test 3.3

Field	Value	Type	Type of Data
Product	10000	Float	Valid Boundary

Test Data 3.D for Test 3.4

Field	Value	Type	Type of Data
Product	0	Float	Invalid Boundary

Test Data 3.E for Test 3.5

Field	Value	Type	Type of Data
Product	-9999	Float	Invalid Extreme

Test Data 3.F for Test 3.6

Field	Value	Type	Type of Data
Product	9999999999999999	Float	Invalid Extreme

Test Data 3.G for Test 3.7

Field	Values	Type	Type of Data
Product[0-4]	1.99, 2.99, 3.99, 4.99, 5.99	Float	Valid

Test Data 3.H for Test 3.8

Field	Values	Type	Type of Data
Product[0-4]	1, 1, 1, 1, 1	Float	Valid

Test Data 3.I for Test 3.9

Field	Values	Type	Type of Data
Product[0-4]	9999, 8888, 7777, 6666, 5555	Float	Valid

4.4.3. Test Runs and Results

Test 3.1

Evidence:

```

- - - - -
Please enter the price of product 1 (in format 0.00):
3.99
- Running sub-total minus vat = £3.99...
- Running sub-total plus vat = £4.79...
- - - - -
Please enter the price of product 2 (in format 0.00):

```

Figure 24. Input and Output / Result for Test 3.1

Result:

Pass: The program allows the valid input of 3.99 and does not prompt the user to input a new value.

Test 3.2

Evidence:

[illegible]

Figure 25. Input and Output / Result for Test 3.2

Result:

Fail: The program did not continue, as it was expecting a float value, and therefore would not accept 'A'. This caused it to enter an infinite loop and crash.

Test 3.3

Evidence:

```
-----  
Please enter the price of product 1 (in format 0.00):  
10000  
- Running sub-total minus vat = £10000.00...  
- Running sub-total plus vat = £12000.00...  
  
-----  
Please enter the price of product 2 (in format 0.00):
```

Figure 26. Input and Output / Result for Test 3.3

Result:

Pass: The program allows the valid boundary product input of 10000 and does not prompt the user to input a new value.

Test 3.4

Evidence:

```
-----  
Please enter the price of product 1 (in format 0.00):  
0  
Product prices must be greater than £0 and less than or equal to £10,000. Please try again.  
Please enter the price of product 1 (in format 0.00):
```

Figure 27. Input and Output / Result for Test 3.4

Result:

Pass: The program handles the invalid boundary product input of 0 and prompts the user to input a new value within the specified range.

Test 3.5

Evidence:

```
-----  
Please enter the price of product 1 (in format 0.00):  
-9999  
Product prices must be greater than £0 and less than or equal to £10,000. Please try again.  
Please enter the price of product 1 (in format 0.00):
```

Figure 28. Input and Output / Result for Test 3.5

Result:

Pass: The program handles the invalid extreme product input of -9999 and prompts the user to input a new value within the specified range.

Test 3.6

Evidence:

```
-----  
Please enter the price of product 1 (in format 0.00):  
9999999999999999  
Product prices must be greater than £0 and less than or equal to £10,000. Please try again.  
Please enter the price of product 1 (in format 0.00):
```

Figure 29. Input and Output / Result for Test 3.6

Result:

Pass: The program handles the invalid extreme product input of 9999999999999999 and prompts the user to input a new value within the specified range.

Test 3.7

Evidence:

```
-----
Please enter the price of product 1 (in format 0.00):
1.99
- Running sub-total minus vat = £1.99...
- Running sub-total plus vat = £2.39...

-----
Please enter the price of product 2 (in format 0.00):
2.99
- Running sub-total minus vat = £4.98...
- Running sub-total plus vat = £5.98...

-----
Please enter the price of product 3 (in format 0.00):
3.99
- Running sub-total minus vat = £8.97...
- Running sub-total plus vat = £10.76...

-----
Please enter the price of product 4 (in format 0.00):
4.99
- Running sub-total minus vat = £13.96...
- Running sub-total plus vat = £16.75...

-----
Please enter the price of product 5 (in format 0.00):
5.99
- Running sub-total minus vat = £19.95...
- Running sub-total plus vat = £23.94...

-----

Sub-total minus VAT is: £19.95.
Final total inc VAT is: £23.94.

-----

Program ended with exit code: 0|
```

Figure 30. Input and Output / Result for Test 3.7

Result:

Product input values = 1.99, 2.99, 3.99, 4.99, 5.99
= 19.95

VAT @ 20% = $19.95 * 0.2$
= 3.99

Total = $19.95 + 3.99$
= 23.94

Pass: The program correctly maintains an accurate sub-total, and then calculates the VAT and total.

Test 3.8

Evidence:

```
-----
Please enter the price of product 1 (in format 0.00):
1
- Running sub-total minus vat = £1.00...
- Running sub-total plus vat = £1.20...

-----
Please enter the price of product 2 (in format 0.00):
1
- Running sub-total minus vat = £2.00...
- Running sub-total plus vat = £2.40...

-----
Please enter the price of product 3 (in format 0.00):
1
- Running sub-total minus vat = £3.00...
- Running sub-total plus vat = £3.60...

-----
Please enter the price of product 4 (in format 0.00):
1
- Running sub-total minus vat = £4.00...
- Running sub-total plus vat = £4.80...

-----
Please enter the price of product 5 (in format 0.00):
1
- Running sub-total minus vat = £5.00...
- Running sub-total plus vat = £6.00...

-----

Sub-total minus VAT is: £5.00.
Final total inc VAT is: £6.00.

-----

Program ended with exit code: 0|
```

Figure 31. Input and Output / Result for Test 3.8

Result:

Product input values = 1, 1, 1, 1, 1
= 5

VAT @ 20% = $5 * 0.2$
= 1

Total = $5 + 1$
= 6

Pass: The program correctly maintains an accurate sub-total, and then calculates the VAT and total.

Test 3.9

Evidence:

```
-----
Please enter the price of product 1 (in format 0.00):
9999
- Running sub-total minus vat = £9999.00...
- Running sub-total plus vat = £11998.80...

-----
Please enter the price of product 2 (in format 0.00):
8888
- Running sub-total minus vat = £18887.00...
- Running sub-total plus vat = £22664.40...

-----
Please enter the price of product 3 (in format 0.00):
7777
- Running sub-total minus vat = £26664.00...
- Running sub-total plus vat = £31996.80...

-----
Please enter the price of product 4 (in format 0.00):
6666
- Running sub-total minus vat = £33330.00...
- Running sub-total plus vat = £39996.00...

-----
Please enter the price of product 5 (in format 0.00):
5555
- Running sub-total minus vat = £38885.00...
- Running sub-total plus vat = £46662.00...

-----

Sub-total minus VAT is: £38885.00.
Final total inc VAT is: £46662.00.

-----

Program ended with exit code: 0
```

Figure 32. Input and Output / Result for Test 3.9

Result:

Product input values = 9999, 8888, 7777, 6666, 5555
= 38885
VAT @ 20% = 38885 * 0.2
= 7777
Total = 38885 + 7777
= 46662

Pass: The program correctly maintains an accurate sub-total, and then calculates the VAT and total.

4.5. Limitations of the Solution

The solution does not currently validate against the input of data of conflicting types, such as the input of a char (character) where a float or integer is expected. This currently causes the program to crash and enter into an infinite loop, as evidenced in test 3.2.

Future improvements to the program would be to implement input validation to check the data type of the input, and handle it accordingly, by catching the error and prompting the user to enter valid data.

5. Exercise 4

Build a fruit machine game which represents the 3 reels of symbols using numbers ranging from 1 to 5. Allow the user to have as many goes as they like (the numbers are chosen randomly each time).

5.1. Problem

The program must allow the user to input an integer value, which will represent their decision on whether they wish to play the slot machine game (and spin again). If they decide yes, it must generate three random numbers, which will act as slot values, and output them in a logical format to represent a slot machine. It must then calculate the points accumulated for that spin based on the slot values and the following rules.

- Three fives (5, 5, 5): 2000 points
- Two fives (5, 5, n): 100 points
- Three matching numbers (n, n, n): 20 points

The program must keep an updated total of points whilst they decide to spin again. The total points should be outputted after each spin, and also when the user decides to terminate.

The program must validate the user's decision input, spin again, and if not 0 or 1, must ask the user to input either 0 or 1.

Table 14. Validation rules for user input, exercise 4

Field	Data Type	Validation Rule
Spin Again	Integer	1 or 0

Table 15. Valid, invalid and erroneous data for exercise 4

Field	Valid	Invalid	Erroneous
Spin Again	1, 0	3, 4, 5, 6	A, %, \$ etc

5.2. Analysis of Solution

5.2.1. Problem Break Down

- a. Initialise variable spin again as 1 (yes)
- b. Print welcome message
- c. Loop if user wants to spin (initialised as yes)
 - a. Ask user if they want to spin
 - b. If yes: show slot machine with 3 numbers
 - i. Generate 3 random numbers
 - ii. Output slots (graphic)
 - iii. Calculate points based on slot values
 1. If three fives (555)
 - a. Points = 2000
 2. If two fives (55)
 - a. Points = 100
 3. If three matching numbers:
 - a. Points = 20
 - iv. Output points
 - c. If no:
 - i. end
- d. Show total point score
- e. Terminate program

5.2.2. Pseudocode

```

function generate_random()
    int random_number = 0
    random_number = 1 + rand() % 5
    return random_number
end

function spin()
    int one = 0
    int two = 0
    int three = 0
    int points = 0

    one = generate_random()
    two = generate_random()
    three = generate_random()

    if(one == 5 and two == 5 and three == 5)
        points = 2000
    else if((one == 5 and two == 5) or (two == 5 and three == 5) or (one == 5 and
three == 5))
        points = 100
    else if(one == two and two == three and one==three)
        points = 20
    end if
    print("you have won {}" + points)
    return points
end

procedure slot_machine_game()
    int points = 0
    int spin_again = 1
    seed_random(time)
    while(spin_again == 1)
        print("Current points = {}" + points)
        print("Do you want to spin? Press 1 for Yes or 0 for No")
        input(spin_again)

        switch case (spin_again):
            case 1:
                points = points + spin()
            case 0:
                print("Game over.")

            default:
                print("Unknown answer, try again. Press 1 for yes or 0
for no.")
                input(spin_again)
            end switch
        end while
        print("You ended on a total of {} points." + points)
    end

function main()
    slot_machine_game()
end

```

5.3. Annotated C Code Listing

```

//includes are necessary for importing libraries
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
//declare prototype functions and procedures
int spin(void);
void slot_machine_game(void);
int generate_random(void);

//generate_random function is used to generate a random number between 1 and 5
//it returns an integer (the random number)
int generate_random(){
    int random_number = 0;
    //assign random numbers between 1 and 5 to the three variables
    random_number = 1 + rand() % 5; //
    return random_number; //return the random number
}
//function spin is used to 'spin' the slot machine and produce slots
//The slot values are compared to the score logic and points are assigned based on the slots
//The points are returned as an integer to the calling function (where an up-to-date total is kept)
int spin(void){
    //declare and initialise variables to 0
    int one = 0; //one is used to store random number one
    int two = 0; //two to store random number two
    int three = 0; //three to store random number three
    int points = 0; //points is used to store points once calculated

    //assign random numbers between 1 and 5 to the three variables
    //call generate_random function eith
    one = generate_random();
    two = generate_random();
    three = generate_random();

    printf("  -----\\n");
    printf("          Slots\\n");
    printf("  -----\\n");
    //print the random numbers in the 'slot machine'
    printf("  |   |   |   |\\n  |  %d |  %d |  %d |\\n  |   |   |   |
|\\n",one,two,three);
    printf("  -----\\n");

    //calculate number of points by comparing the three numbers
    //example numbers: | 3 | 1 | 2 |
    if(one == 5 && two == 5 && three == 5){
        //Scenario 1: 2000 points
        // - - - - -

```

```

        //Must have three fives for big point win
        //If all three numbers are fives then:
        //set variable points to 2000
        points = 2000;
    }else if((one == 5 && two == 5) || (two == 5 && three == 5) || (one == 5 && three == 5)){
        //Scenario 2: 100 points
        // - - - - -
        //Two fives wins 100 points
        points = 100;
    }else if(one == two && two == three && one==three){
        //Scenario 3: 20 points
        //Any three of the same number, eg 333, 444, 555
        // - - - - -
        //Compare numbers: if one is equal to two eg 3 = 1, and two is equal to three eg 1 =
2, and one is equal to three eg 3 = 2 then:
        //set variable points to be 20
        points = 20;
    }

    //print the points
    printf("\n-----\n");
    if (points > 0) {
        printf("~ ~ You have won %d points! ~ ~\n",points);
    }else{
        printf("~ ~ You lose. %d points! ~ ~\n",points);
    }
    printf("-----\n");
    //return the integer points
    return points;
}

```

```

//slot machine game calls spin() procedure
//it is used to take user input on whether they wish to continue playing
//it is also used to keep a tally of points
void slot_machine_game(){
    //declare and initialise variables
    int points = 0; //points variable used for total running point score
    int spin_again = 1; //spin again used as a decision variable
    int start = 1; //set to 1 if game is on first iteration, used to hide the current points
message

```

```

    //print welcome message
    printf("Welcome to the slot machine game.\n\n");

    //While loop for decision
    //While spin_again == 1 (True / Yes)
    while(spin_again == 1){
        //if not on first iteration
        if(start == 0){

```

```

        //print the current points
        printf("\n    Current points = %d!\n",points);
        printf("- - - - - \n");
    }
    //prompt user asking if they want to spin again
    printf("    Do you want to spin?\n    Press 1 for Yes, 0 for No!\n");
    scanf("%d",&spin_again);

    //if YES (1), they want to spin again
    switch(spin_again){
        case 1 :
            //update points total by calling spin function and using the return value to
add to the total
            points = points + spin();
            break;
        case 0://if NO (0), they do not want to spin again
            printf("    Game over.\n");//print game over
            break;
        default:
            //if the user enters an unknown value, eg '3', print a prompt to enter 1 for
yes, or 0 for no
            printf("Unknown answer, try again. Do you want to spin?\n    Press 1 for Yes,
0 for No!\n");
            scanf("%d",&spin_again);
        }
        start = 0;
    }
    printf("    You ended on a total of %d\n points!\n",points);//print end game message
}
int main(){
    //seed randomness using time as the seed
    //this is used for generating random slot numbers
    //seeding must be done only once during program execution!
    srand(time(NULL));

    slot_machine_game();//call slot machine game procedure
}

```

5.4. Testing

A test plan has been created to test against three main types of data: valid, invalid, and erroneous, plus both valid and invalid boundary data. It will also test that the output produced by the program is accurate and what is expected.

5.4.1. Test Plan

Test #	Test	Purpose	Test data	Expected result
4.1	Test input “1” (yes) allows user to spin again	Ensure the program loops as expected	4.A	Pass
4.2	Test input “0” terminates the program	Ensure the program terminates as expected	4.B	Pass
4.3	Test input “3” is not accepted as a spinAgain response and prompt for another	Ensure the program flow is maintained and the system handles errors correctly	4.C	Prompt user to enter another value (1 or 0)
4.4	Test the program consistently generates 3 random numbers (3 may match, but must differ on each iteration)	Ensure the program generates random numbers as would a slot machine	N/A	Numbers may match but must differ on each iteration
4.5	Test the program calculates points correctly for the scenario “3 matching numbers = 20 points”	Ensure system logic correctly assigns points based on slot values	N/A	Points are allocated correctly
4.6	Test the program calculates points correctly for the scenario “2 matching fives (5) = 100 points”	Ensure system logic correctly assigns points based on slot values	N/A	Points are allocated correctly
4.7	Test the program calculates points correctly for the scenario “3 matching fives (5) = 2000 points”	Ensure system logic correctly assigns points based on slot values	N/A	Points are allocated correctly

4.8	Test the program correctly maintains a running total of points whilst the player continues to spin	Ensure the system correctly calculates the total number of points and maintains an accurate total	N/A	Points are calculated correctly and end on a final accurate total
4.9	Enter <u>erroneous data</u> for Spin Again	Ensure system does not accept erroneous data	4.D	Fail

5.4.2. Test Data

Test Data 4.A for Test 4.1

Field	Value	Type	Type of Data
SpinAgain	1	Integer	Valid

Test Data 4.B for Test 4.2

Field	Value	Type	Type of Data
SpinAgain	0	Integer	Valid

Test Data 4.C for Test 4.3

Field	Value	Type	Type of Data
SpinAgain	3	Integer	Invalid

Test Data 4.C for Test 4.9

Field	Value	Type	Type of Data
SpinAgain	&	Char	Erroneous

5.4.3. Test Runs and Results

Test 4.1**Evidence:**

```
Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
1
-----
Slots
-----
| 3 | 1 | 5 |
|---|
~ ~ You have won 0 points! ~ ~
-----
Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
```

*Figure 33. Input and Output / Result for Test 4.1***Result:**

Pass: When 1 (yes) was input, the spin() method was called to produce the slots and calculate points, and therefore ensures that the correct program flow is maintained.

Test 4.2

Evidence:

```
Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
1
-----
Slots
-----
| 2 | 2 | 2 |
|---|
-----
~ ~ You have won 20 points! ~ ~
-----
Current points = 20!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
0
Game over.
You ended on a total of 20
points!
Program ended with exit code: 0
```

Figure 34. Input and Output / Result for Test 4.2

Result:

Pass as expected: When 0 (no) was input, the message “game over” was printed and the total points were outputted to the user, finally the program terminated.

Test 4.3

Evidence:

```
Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
1
-----
Slots
-----
| 4 | 1 | 3 |
|---|
-----

~ ~ You have won 0 points! ~ ~
-----

Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
3
Unknown answer, try again. Do you want to spin?
Press 1 for Yes, 0 for No!
```

Figure 35. Input and Output / Result for Test 4.3

Result:

Pass as expected: When 3 was input, the message “Unknown answer. Try again. ...” was outputted to the user, and the user was prompted to enter their input again.

Test 4.4

Evidence:

```
Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
1
-----
Slots
-----
| 5 | 2 | 3 |
|---|
-----

~ ~ You have won 0 points! ~ ~
-----

Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
1
-----
Slots
-----
| 2 | 1 | 4 |
|---|
-----

~ ~ You have won 0 points! ~ ~
-----

Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
1
-----
Slots
-----
| 3 | 2 | 1 |
|---|
-----

~ ~ You have won 0 points! ~ ~
-----

Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
```

Figure 36. Input and Output / Result for Test 4.4

Result:

Pass: During each iteration of the game, different slot numbers were produced, therefore the program does generate random numbers as required.

Test 4.5

Evidence:

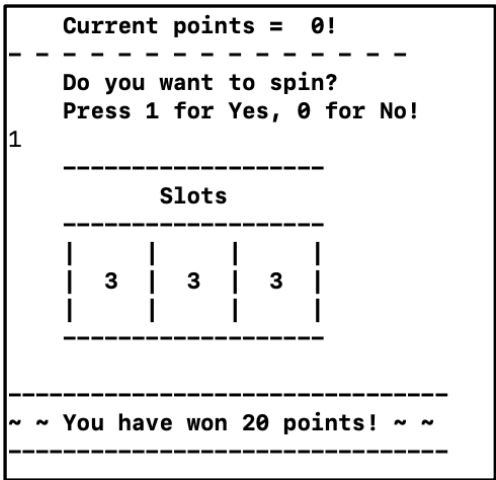


Figure 37. Input and Output / Result for Test 4.5

Result:

Pass: The program correctly allocates 20 points when there are three matching numbers (eg 333). This proves that the system logic correctly assigns points based on slot values.

Test 4.6

Evidence:

1

Do you want to spin?
Press 1 for Yes, 0 for No!

Slots

|

3

|

5

|

5

|

~ ~ You have won 100 points! ~ ~

Figure 38. Input and Output / Result for Test 4.6

Result:

Pass: The program correctly allocates 100 points when there is two matching fives (eg 55[n]).
This proves that the system logic correctly assigns points based on slot values.

Test 4.7

Evidence:

```
Current points = 0!
-----
Do you want to spin?
Press 1 for Yes, 0 for No!
1
-----
Slots
-----
| 5 | 5 | 5 |
|---|
~ ~ You have won 2000 points! ~ ~
-----
Current points = 2000!
-----
```

Figure 39. Input and Output / Result for Test 4.7

Result:

Pass: The program correctly allocates 2000 points when there is three matching fives (555). This proves that the system logic correctly assigns points based on slot values.

Test 4.8

Evidence:

1

Do you want to spin?
Press 1 for Yes, 0 for No!

Slots

3

5

5

~ ~ You have won 100 points! ~ ~

Current points = 100!

1

Do you want to spin?
Press 1 for Yes, 0 for No!

Slots

4

2

5

~ ~ You have won 0 points! ~ ~

Current points = 100!

1

Do you want to spin?
Press 1 for Yes, 0 for No!

Slots

1

1

1

~ ~ You have won 20 points! ~ ~

Current points = 120!

Figure 40. Input and Output / Result for Test 4.8

Result:

Pass: In the evidence above, two scenarios have been triggered:

1. Two fives [355] = (100 points)
2. Three matching numbers [111] = (20 points)

Total: $100 + 20 = 120$

During this, current points is updated and contains the value 120. Therefore, the program correctly calculates the total number of points and maintains an accurate total.

Page 85 of 87

Test 4.9

Evidence:

4	1	2

~ ~ You lose. 0 points! ~ ~		

Current points = 193720!		

Do you want to spin?		
Press 1 for Yes, 0 for No!		

Slots		

3	3	3

~ ~ You have won 20 points! ~ ~		

Current points = 193740!		

Do you want to spin?		
Press 1 for Yes, 0 for No!		

Slots		

3	5	2

Figure 41 Input and Output / Result for Test 4.9

Result:

Fail: The program did not continue, as it was expecting a float value, and therefore would not accept the erroneous data '&'. This caused it to enter an infinite loop and crash.

5.5. Limitations of the Solution

The solution does not currently validate against the input of data of conflicting types, such as the input of a char (character) where a float or integer is expected. This currently causes the program to crash and enter into an infinite loop, as evidenced in test 4.9.

Future improvements to the program would be to implement input validation to check the data type of the input, and handle it accordingly, by catching the error and prompting the user to enter valid data.

6. Reference List

- [1] “Computer Programming Variables.” [Online]. Available: https://www.tutorialspoint.com/computer_programming/computer_programming_variables.htm. [Accessed: 17-Feb-2019].
- [2] “C Constants and Literals.” [Online]. Available: https://www.tutorialspoint.com/cprogramming/c_constants.htm. [Accessed: 17-Feb-2019].
- [3] “Identifier/Variable naming conventions in C language [Rules and Recommendations] - IncludeHelp.” [Online]. Available: <https://www.includehelp.com/c/identifier-variable-naming-conventions.aspx>. [Accessed: 18-Feb-2019].
- [4] “Bubble sort algorithm.” [Online]. Available: <http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/7-Sort/bubble-sort.html>. [Accessed: 18-Feb-2019].
- [5] “TechTerms Definition of an Array.” [Online]. Available: <https://techterms.com/definition/array>. [Accessed: 18-Feb-2019].
- [6] “For-loop flowchart”. [Online]. Available: <https://clanguageprogrammes.blogspot.com/2013/04/syntax-and-flowchart-of-for-loop.html>. [Accessed: 18-Feb-2019].