Logo, company name

Description automatically generated

CS2043

Assignment 1

September 28th, 2023

UNB Fredericton

**Class** **Code**: CS2043

**Document**: Assignment 1

**Student** **Name**: Will Ross

**Student** **Number**: #3734692

**Due** **Date**: September 29th, 2023

**Exercise 1**

**a)**

|  |  |
| --- | --- |
| **Type of activity** | **Time** |
| Understanding the problem | 5 mins |
| Identifying the structure | 15 mins |
| Coding | 25 mins |
| Testing (data set, compiling, error issues) | 15 mins |
| Documentation | 25 mins |
| **Total time of project** | **1 hour and 25 minutes** |

**b)**

A screenshot of a computer program

Description automatically generated

**c)**

Source Code

import java.util.Scanner;

public class Assignment1{

public static void main(String[] args){

int input[];

Scanner intScanner = new Scanner(System.in);

System.out.println("How many numbers would you like to take the average of?");

input = new int[intScanner.nextInt()];

System.out.println("Enter a number then press enter and repeat");

for(int i = 0; i < input.length; i++){

input[i] = intScanner.nextInt();

}

intScanner.close();

System.out.println("The average of the numbers you entered is: " + inputAverage(input));

}

public static double inputAverage(int input[]){

double toReturn = 0;

int total = 0;

for(int i = 0; i < input.length; i++){

total += input[i];

}

toReturn = (((double) total)/ input.length);

return toReturn;

}

}

**d)**

|  |  |  |
| --- | --- | --- |
| **Type of activity** | **Estimated Time** | **Actual Time** |
| Understanding the problem | 5 mins | 5 mins |
| Identifying the structure | 15 mins | 10 mins |
| Coding | 25 mins | 20 mins |
| Testing (data set, compiling, error issues) | 15 mins | 15 mins |
| Documentation | 25 mins | 20 mins |
| **Total time of project** | **1 hour and 25 minutes** | **1 hour and 10 minutes** |

**e)**

A screenshot of a computer screen

Description automatically generated

Cyclomatic complexity = E – N + 2p,

where E = number of edges of the graph,

N = number of nodes of the graph, and

p = number of connected components (usually p = 1)

E = 12

N = 11

P = 1

Cyclomatic complexity = 12 – 11 + 2(1)

Cyclomatic complexity = 3

**Exercise 2**

**a)**

|  |  |
| --- | --- |
| **Type of activity** | **Time** |
| Understanding the problem | 5 mins |
| Identifying the structure | 10 mins |
| Coding | 15 mins |
| Testing (data set, compiling, error issues) | 15 mins |
| Documentation | 25 mins |
| **Total time of project** | **1 hour and 20 minutes** |

**b)**

A screen shot of a graph

Description automatically generated

**C)**

Source Code

import java.util.Scanner;

public class Assignment1{

public static void main(String[] args){

int input[];

Scanner intScanner = new Scanner(System.in);

System.out.println("How many numbers would you like to take the average of?");

input = new int[intScanner.nextInt()];

System.out.println("Enter a number then press enter and repeat");

for(int i = 0; i < input.length; i++){

input[i] = intScanner.nextInt();

}

intScanner.close();

System.out.println("The average of the numbers you entered is: " + inputAverage(input));

System.out.println("The greatest number that was submitted was: " + getGreatestNumber(input));

}

public static double inputAverage(int input[]){

double toReturn = 0;

int total = 0;

for(int i = 0; i < input.length; i++){

total += input[i];

}

toReturn = (((double) total)/ input.length);

return toReturn;

}

public static int getGreatestNumber(int input[]){

int current = input[0];

for(int i = 1; i < input.length; i++){

if(current < input[i]){

current = input[i];

}

}

return current;

}

}

**d)**

|  |  |  |
| --- | --- | --- |
| **Type of activity** | **Estimated Time** | **Actual Time** |
| Understanding the problem | 5 mins | 2 mins |
| Identifying the structure | 10 mins | 5 mins |
| Coding | 15 mins | 15 mins |
| Testing (data set, compiling, error issues) | 15 mins | 15 mins |
| Documentation | 15 mins | 15 mins |
| **Total time of project** | **1 hour and 20 minutes** | **52 minutes** |

**e)**

A black screen with white text

Description automatically generated

Cyclomatic complexity = E – N + 2p,

where E = number of edges of the graph,

N = number of nodes of the graph, and

p = number of connected components (usually p = 1)

E = 21

N = 18

P = 1

Cyclomatic complexity = 21 – 18 + 2(1)

Cyclomatic complexity = 5

**Exercise 3**

**a)**

|  |  |
| --- | --- |
| **Type of activity** | **Time** |
| Understanding the problem | 0 mins |
| Identifying the structure | 5 mins |
| Coding | 10 mins |
| Testing (data set, compiling, error issues) | 10 mins |
| Documentation | 25 mins |
| **Total time of project** | **50 minutes** |

**b)**

A screenshot of a computer

Description automatically generated

**c)**

Source Code

import java.util.Scanner;

public class Assignment1{

public static void main(String[] args){

int input[];

Scanner intScanner = new Scanner(System.in);

System.out.println("How many numbers would you like to take the average of?");

input = new int[intScanner.nextInt()];

System.out.println("Enter a number then press enter and repeat");

for(int i = 0; i < input.length; i++){

input[i] = intScanner.nextInt();

}

intScanner.close();

System.out.println("The average of the numbers you entered is: " + inputAverage(input));

System.out.println("The greatest number that was submitted was: " + getGreatestNumber(input));

System.out.println("The smallest number that was submitted was: " + getSmallestNumber(input));

}

public static double inputAverage(int input[]){

double toReturn = 0;

int total = 0;

for(int i = 0; i < input.length; i++){

total += input[i];

}

toReturn = (((double) total)/ input.length);

return toReturn;

}

public static int getGreatestNumber(int input[]){

int current = input[0];

for(int i = 1; i < input.length; i++){

if(current < input[i]){

current = input[i];

}

}

return current;

}

public static int getSmallestNumber(int input[]){

int current = input[0];

for(int i = 1; i < input.length; i++){

if(current > input[i]){

current = input[i];

}

}

return current;

}

}

**d)**

|  |  |  |
| --- | --- | --- |
| **Type of activity** | **Estimated Time** | **Actual Time** |
| Understanding the problem | 0 mins | 0 mins |
| Identifying the structure | 5 mins | 0 mins |
| Coding | 10 mins | 4 mins |
| Testing (data set, compiling, error issues) | 10 mins | 10 mins |
| Documentation | 25 mins | 15 mins |
| **Total time of project** | **50 minutes** | **29 minutes** |

**e)**

**A diagram of a computer

Description automatically generated with medium confidence**

Cyclomatic complexity = E – N + 2p,

where E = number of edges of the graph,

N = number of nodes of the graph, and

p = number of connected components (usually p = 1)

E = 30

N = 25

P = 1

Cyclomatic complexity = 30 – 25 + 2(1)

Cyclomatic complexity = 7

**Exercise 4**

**a)** Show on a graph the total effort estimates for Exercises 1, 2, 3.

**b)** Show on a graph the total actual effort values for Exercises 1, 2, 3.

**c)** Show on a graph the count for the total lines of code in each of Exercises 1, 2, 3. Count every line in your source code files (you may use wc command in Linux).

**d)** Show on a graph the count for only the new lines of code in Exercises 1, 2, 3 (in Ex. 1 it’s the entire program, in Ex. 2 it’s the number of new lines of code added, just estimate, then similar for Ex. 3).

**e)** Show on a graph the values calculated using McCabe’s Cyclomatic Complexity for programs in Exercises 1, 2, 3.

**Exercise 5**

When looking at our graphs from exercise 4 we see as the complexity goes up and our actual time effort goes down. Why, probably due to code reusability, being able to copy the code I wrote to get the greatest number and paste the code for the get the smallest number method, with only having to change the if operator from greater the to less than. (Figure 5.1)

*(Figure 5.1)*

Another factor would be the amount of code written. For exercise 1 I wrote much more code compared to exercise 3, but the code was much simpler than exercise 3. For exercise 1 most of the code was setting up my scanner to take in the inputs rather than the actual average number method. As it goes on there is less code, but it gets more complex with less actual written code. (Figure 5.2)

*(Figure 5.2)*