Faith Olopade

CSU11021 Final Examination Description

Student Number: 21364066

Faith Olopade
12-14-2021

Part 1: Statistics

- To find the average I found the total sum of the elements in my sequence and then divide by the number of elements in my set.
- To find the sum I used a while loop. This loop ensured that the count of elements in my sequence was not greater than the size of the sequence, this was needed to make sure that I would not have an infinite loop and my average would be correct. In this loop I also incremented by sequence by 4 rather than by 1 since we are dealing with word sized values and not byte size values.
- To find the average I then divided the sum of elements by the number of elements in the sequence. I needed the remainder to help me determine whether I needed to round up or to round down
- My next task was to round to the nearest whole number. To do this I used an if statement to check that the remainder was greater than my size divided by two which gave me my midpoint. This told me if I needed to round down or to round up. For example, if my sum was 55 and I divided by 10 I would get 5.5 and in my average register I would have 5 due to UDIV only storing integer values then if the remainder was equal or greater than this remainder, I would know that I have gone past my midpoint and need to add 1 to round up. If it was less than then I would know that I am below my midpoint and no change is required.
- To find the mode I analysed each element in my sequence and see which repeats the most. I used an outer while loop that would make sure I did not go past the last element in my sequence and then I needed an inner while loop to count from the current element to the end of my sequence and see how many times it repeats.
- Before I started the first loop, I needed to go back to my first element as after completing the average my sequence is on the last element. I also kept a register which would be replaced if after my compare instruction with my count of repeats of the current element was greater than the previous elements number of repeats, hence giving the mode.

<u>Average Pseudo Code:</u>

```
@ average = 0

@ sizeSequence = word[AdrSequence]

@ AdrSequence += 4

@ countOfElements = 0

@ sumElements = 0


@ while(count < sizeSequence) {

        @ element = word[AdrSequence]

        @ countOfElements += 1

        @ sumElements += element
```

```
        @ AdrSequence += 4
@ }

@ tempAverage = 0
@ tempAverage = sumOfElements / sizeSequence
@ quotient = sizeSequence * tempAverage
@ remainder = sumOfElements - quotient
@ divisor = 2
@ sizeSequence / divisor

@ if(remainder > sizeSequence / divisor) {
        @ tempAverage += 1
@ }
@ average = tempAverage
```

Mode Psuedo Code:

```
@ multiplier = 4
@ resetSequence = sizeSequnce * multiplier
@ resetRegister
@ AdrSequence -= resetSequence
@ countHighestRepeat = 1
@ countOfElements = 1
@ mode = 0
@ while(countOfElements < sizeSequence) {
        @ countOfElements += 1
        @ element = word[AdrSequence]
        @ repeatsOfElement = 1
        @ AdrSequence += 4
        @ countOfRemainingElements = 1

        @ while(countOfRemainingElements <= sizeSequence) {
```

```
                @ countOfRemainingElements += 1
                @ temp = word[AdrSequence]
                @ repeatsOfElement +=1
                @ if(element = temp) {
                        @ repeatsOfElement +=1
                @ }
        @ }

        @ if(repeatsOfElement > countHighestRepeat) {
                @ countHighestRepeat = repeatsOfElement
                @ mode = element
        @ }
@ }
```

Part 2: Calculator

- To isolate the ascii expression I needed to create the numbers to do this I created a while loop that will stop when it met a non-digit character and depending on what the non-digit operation was I either multiplied, subtracted or added.
- In terms of subtraction, I first created the number as a positive number and then I multiplied the number by -1 to make it negative.
- I made a while loop which made sure if the character that was being passed was not null my loop would be executed.

```
@ resultOfOperation = 0
@ boolean subtract = false
@ if (char = '-') {
        @ subtract = true
        @ AdrExpression += 1
        @ char = byte[AdrExpression]
@ }
@ currentValue = 0
@ multiplier
@ while(char >= '0' & char <= '9') {
        @ char -= '0'
```

```
        @ currentValue *= 10
        @ currentValue += char
        @ AdrExpression += 1
        @ char = byte[AdrExpresion]
@ }
@ resetRegister
@ previousValue = currentValue
@ muliplier = -1
@ if(subtract = true) {
        @ subtraction = false
        @ currentValue *= -1
@ }
@ resultOfOperation += previousValue
@ functionSelect = 0
@ while(char != NULL) {
        @ if(char = '*') {
                @ functionSelect = 1
        @ }
        @ else {
                @ functionSelect = 2
        @ if (char = '-') {
                @ subtraction = true
                @ AdrExpression += 1
                @ char = byte[AdrExpression]
        @ }
        @ if(subtraction = false & char = '-') {
                @ subtraction = true
                @ AdrExpression += 1
                @ char = byte [AdrExpression]
        @ }
        @ while(char >= '0' & char <= '9') {
```

```
                @ char -= '0'
                @ currentValue *= 10
                @ currentValue += char
                @ AdrExpression += 1
                @ char = byte[AdrExpresion]
        @ }
        @ if(subtraction = true) {
                @ subtraction = false
                @ currentValue *= -1
        @ }
        @ if(functionSelect = 1) {
                @ resultOfOperation -= previousValue
                @ mutiplication = previousValue * currentValue
                @ resultOfOperation += multiplication
        @ }
        @ else {
                @ resultOfOperation += currentValue
        @ }
        @ previousValue = currentValue
@ }
```