

Software Development

Fundamentals of Software Development – Course Work

Name : Momatahina Akter Chaiti
Student ID : 4213933

Introduction:

I was well in the principles of software development going into this mixed-learning program, but I was also eager to learn more about its nuances. My research over several weeks taught me about Python and C, which both helped me discover new facets of the world of coding.

Because Python is interpretable, it offers a straightforward foundation. Coding became simpler for me because to high-level abstractions and dynamic typing, which allowed me to concentrate on novel problem-solving strategies rather than minute technical details. The word count for vowels and word length, password compliance, and other concerns resolved demonstrated how readable and helpful the Python community is.

On the other hand, the plunge into C revealed a more demanding but rewarding learning path. Because C is low-level and compiled, it required a deep comprehension of memory management and exact control over data types. C's fine-grained control was made clear by the statistics Library challenge, which also offered insightful information on effective low-level programming.

The comparison of Python and C highlighted the differences in how the languages are implemented. Python's easy-to-learn interpreted style and C's powerful but labor-intensive hand-compiled compilation style brought out the many sides of programming. This learning experience improved my understanding of syntax, vocabulary, and subroutines by highlighting the readability of Python and the subtle control of C at a lower level.

Through the prism of algorithmic thinking, this excursion not only improved my technical proficiency but also gave me a flexible approach to problem-solving. When I think back on the

difficulties I overcame with C—Password Compliance, Word Length and Vowels Counter, and Statistics Library—I can see that I have come a long way in terms of being able to handle a variety of programming-related difficulties.

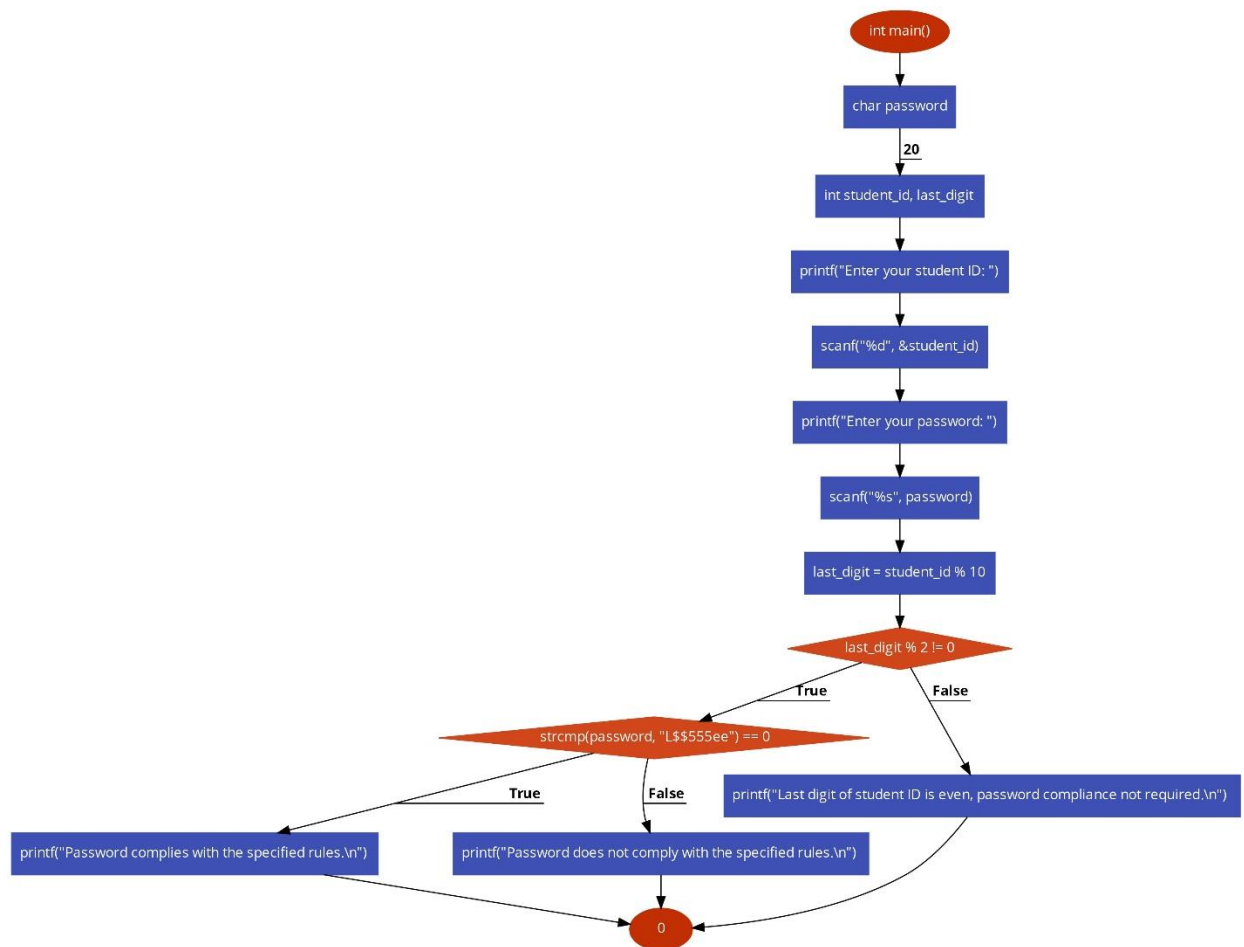
Problem 1: Password Compliance:

1. Request the user's student ID.
2. Request the password from the user.
3. Take off the student ID's final digit.
4. Take the following action if the final digit is odd (1, 3, 5, 7, or 9):

Verify whether the password corresponds to the pattern "L\$555ee."

- 6. "Password complies with the specified rules" will be output if the pattern in the password matches.
 - 7. "Password does not comply with the specified rules" will be output if the password does not match the pattern.
5. "Last digit of student ID is even, password compliance not required" will be output if the last digit is even.

Flowchart or Diagram:



Code snippets from the implementation:

- The user is prompted to input their student ID at the beginning of the program, and this information is saved in the student_ID variable.
- By taking the modulo 10, which is kept in last digit, it retrieves the final digit from the student ID.
- After that, the user is prompted by the application to input their password.
- When the final digit is odd, the password is compared to the predetermined pattern "L\$555ee." If so, the application verifies that the password conforms with the guidelines. If not, the user is notified that the password is not valid.

- The application notifies the user explicitly that password compliance is not necessary if the final digit is even.

Screenshot:

Provide the student id: 4213933 (this is my student id) or any odd number

Password: L\$\$555ee

The screenshot displays a C++ IDE with the source code for `Password_Compliance.c` and a terminal window showing the program's execution. The code prompts for a student ID and password, checks if the password matches "L\$\$555ee", and verifies if the last digit of the student ID is odd. If both conditions are met, it prints "Password complies with the specified rules."; otherwise, it prints "Password does not comply with the specified rules." or "Last digit of student ID is even, password compliance not required." depending on the last digit's parity.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char password[20];
6     int student_id, last_digit;
7
8     printf("Enter your student ID: ");
9     scanf("%d", &student_id);
10    printf("Enter your password: ");
11    scanf("%s", password);
12
13    last_digit = student_id % 10;
14
15    if (last_digit % 2 != 0) {
16        if (strcmp(password, "L$$555ee") == 0) {
17            printf("Password complies with the specified rules.\n");
18        } else {
19            printf("Password does not comply with the specified rules.\n");
20        }
21    } else {
22        printf("Last digit of student ID is even, password compliance not required.\n");
23    }
24
25    return 0;
26 }
```

Execution output:

```
Enter your student ID: 4213933
Enter your password: L$$555ee
Password complies with the specified rules.

Process returned 0 (0x0)   execution time : 64.259 s
Press any key to continue.
```

we provide an even ID or password it won't work here:

The screenshot displays a C++ IDE with a file named `Password_Compliance.c`. The code defines a `main` function that prompts the user for a student ID and a password. It checks if the last digit of the student ID is even; if so, it states that password compliance is not required. Otherwise, it checks if the password matches the pattern `L?0$555eg`. If the password matches, it confirms compliance; otherwise, it states that the password does not comply. The program returns 0 and execution time is noted as 36.079 s.

```
1 #include <stdio.h>
2 #include <string.h>
3
4
5 int main() {
6     char password[20];
7     int student_id, last_digit;
8
9     printf("Enter your student ID: ");
10    scanf("%d", &student_id);
11    printf("Enter your password: ");
12    scanf("%s", password);
13
14    last_digit = student_id % 10;
15
16    if (last_digit % 2 != 0) {
17        if (strcmp(password, "L?0$555eg") == 0) {
18            printf("Password complies with the specified rules.\n");
19        } else {
20            printf("Password does not comply with the specified rules.\n");
21        }
22    } else {
23        printf("Last digit of student ID is even, password compliance not required.\n");
24    }
25
26    return 0;
27 }
```

The terminal output shows the user entering student ID 4213866 and password A23\$@123. The program outputs: "Last digit of student ID is even, password compliance not required." and "Process returned 0 (0x0) execution time : 36.079 s Press any key to continue."

This screenshot shows the software running. The user entered a password that follows the given pattern and a student ID that ends in an odd number. Consequently, the application accurately verifies that the password conforms with the guidelines.

Problem 2: Word length and vowels counter:

Pseudocode:

1. Create a blank string array called words.
2. Set the value of the integer variable `num_words` to zero.
3. Set the `smallest_word` and `largest_word` strings to nothing at first.
4. Set the values of the two integer variables, `max_vowels`, and `min_vowels`, to a significant negative amount.

Enter `num_words`.

6. For an `i` between 1 and `num_words`:

A word is entered and stored in words[i].

b. Determine the word's vowel count and record it in vowels_count.

b. Should vowels_count exceed max_vowels:

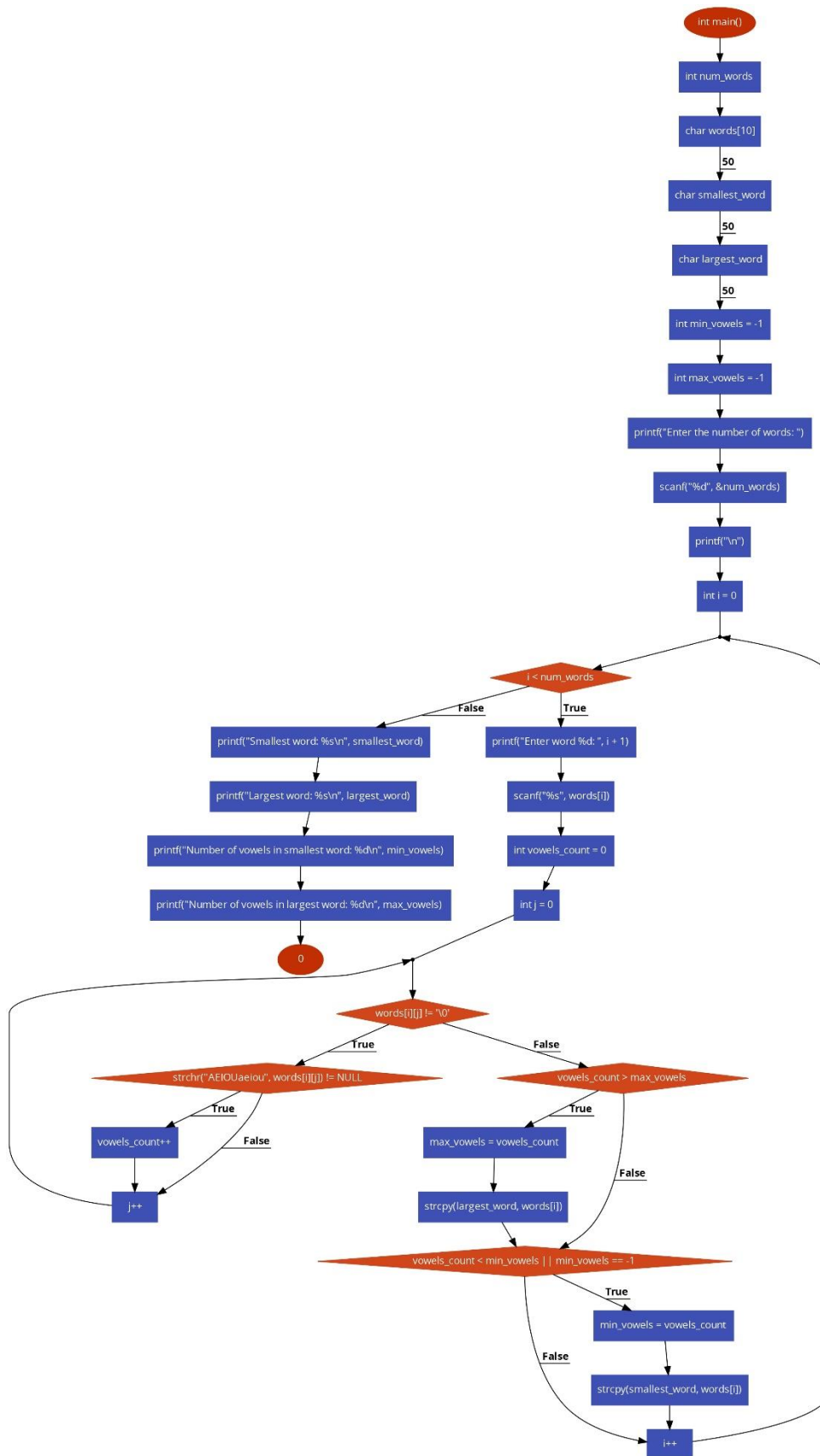
- Vowels_count = max_vowels
- Assign words[i] to largest_word

d. In the event that min_vowels has a significant negative value or vowels_count is less than min_vowels:

- Vowels_count = min_vowels
- Assign words[i] as the smallest word.

7. Produce min_vowels, max_vowels, largest_word, and smallest_word.

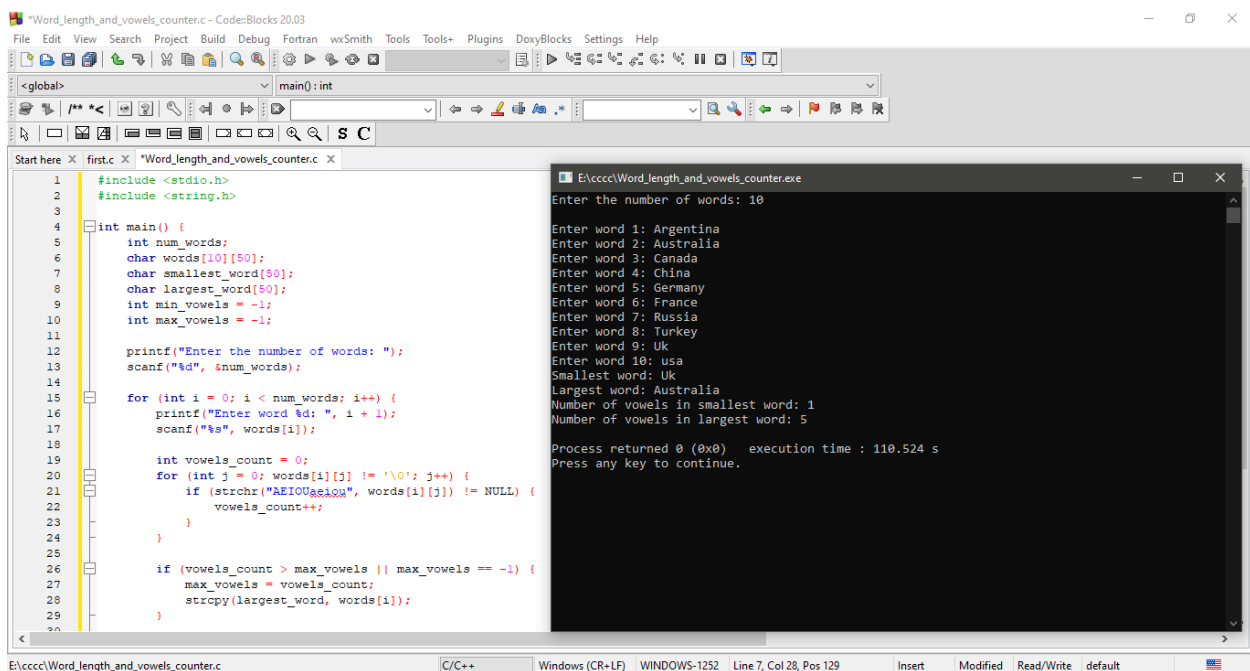
Flowchart or Diagram:



Code snippets from the implementation:

- The user is first prompted by the application to enter the desired word count.
- It then calculates the number of vowels in each word while taking word inputs and storing them in an array.
- It also determines the largest and smallest words and their respective vowel counts during this procedure.
- The program then shows the largest word, the smallest word, and the total number of vowels in both words.

Screenshot:



The screenshot displays a code editor window titled "Word_length_and_vowels_counter.c - Code::Blocks 20.03" and a terminal window titled "E:\cccc\Word_length_and_vowels_counter.exe".

Code Editor Content:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     int num_words;
6     char words[10][80];
7     char smallest_word[80];
8     char largest_word[80];
9     int min_vowels = -1;
10    int max_vowels = -1;
11
12    printf("Enter the number of words: ");
13    scanf("%d", &num_words);
14
15    for (int i = 0; i < num_words; i++) {
16        printf("Enter word %d: ", i + 1);
17        scanf("%s", words[i]);
18
19        int vowels_count = 0;
20        for (int j = 0; words[i][j] != '\0'; j++) {
21            if (strchr("AEIOUaeiou", words[i][j]) != NULL) {
22                vowels_count++;
23            }
24        }
25
26        if (vowels_count > max_vowels || max_vowels == -1) {
27            max_vowels = vowels_count;
28            strcpy(largest_word, words[i]);
29        }
30    }
```

Terminal Output:

```
Enter the number of words: 10
Enter word 1: Argentina
Enter word 2: Australia
Enter word 3: Canada
Enter word 4: China
Enter word 5: Germany
Enter word 6: France
Enter word 7: Russia
Enter word 8: Turkey
Enter word 9: Uk
Enter word 10: usa
Smallest word: Uk
Largest word: Australia
Number of vowels in smallest word: 1
Number of vowels in largest word: 5

Process returned 0 (0x0)   execution time : 110.524 s
Press any key to continue.
```

The application is running in this screenshot, and we selected 10 nations from the given list. Next, the software determines the largest and smallest words accurately and counts the vowels in each.

3 Problem 3: Statistics Library:

Pseudocode:

Function mean(data, n):

sum = 0


```
for i from 0 to n-1:
```

```
    sum = sum + data[i]
```

```
return sum / n
```

```
Function median(data, n):
```

```
    Sort data in ascending order
```

```
    if n is even:
```

```
        mid1 = (n - 1) / 2
```

```
        mid2 = n / 2
```

```
        return (data[mid1] + data[mid2]) / 2.0
```

```
    else:
```

```
        mid = n / 2
```

```
        return data[mid]
```

```
Function kurtosis(data, n):
```

```
    # Specific kurtosis formula implementation is needed
```

```
    # You can add your formula here
```

```
    # Example: Return my_kurtosis_formula(data, n)
```

```
Function standard_deviation(data, n):
```

```
    data_mean = mean(data, n)
```

```
    sum_of_squared_diff = 0
```

```
    for i from 0 to n-1:
```

```
        diff = data[i] - data_mean
```

```
sum_of_squared_diff = sum_of_squared_diff + (diff * diff)
```

```
variance = sum_of_squared_diff / n
```

```
return square root of variance
```

Declare n as the number of data points

Declare data as an array of floats

Initialize data with the provided dataset

Call mean(data, n) and store the result in mean_value

Print "Mean: " + mean_value

Call median(data, n) and store the result in median_value

Print "Median: " + median_value

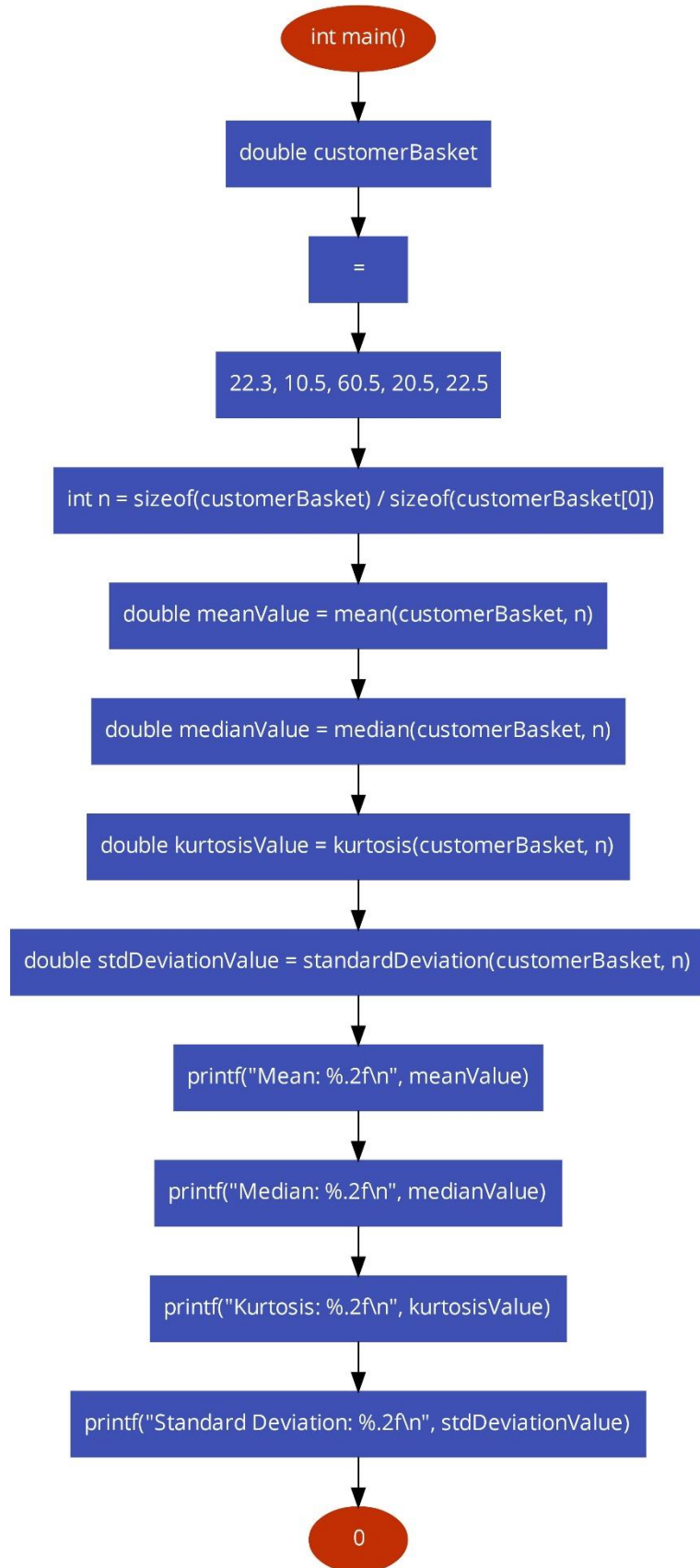
Call standard_deviation(data, n) and store the result in std_deviation

Print "Standard Deviation: " + std_deviation

Implement a specific kurtosis formula and call it to calculate kurtosis_value

Print "Kurtosis: " + kurtosis_value

Flowchart or Diagram:

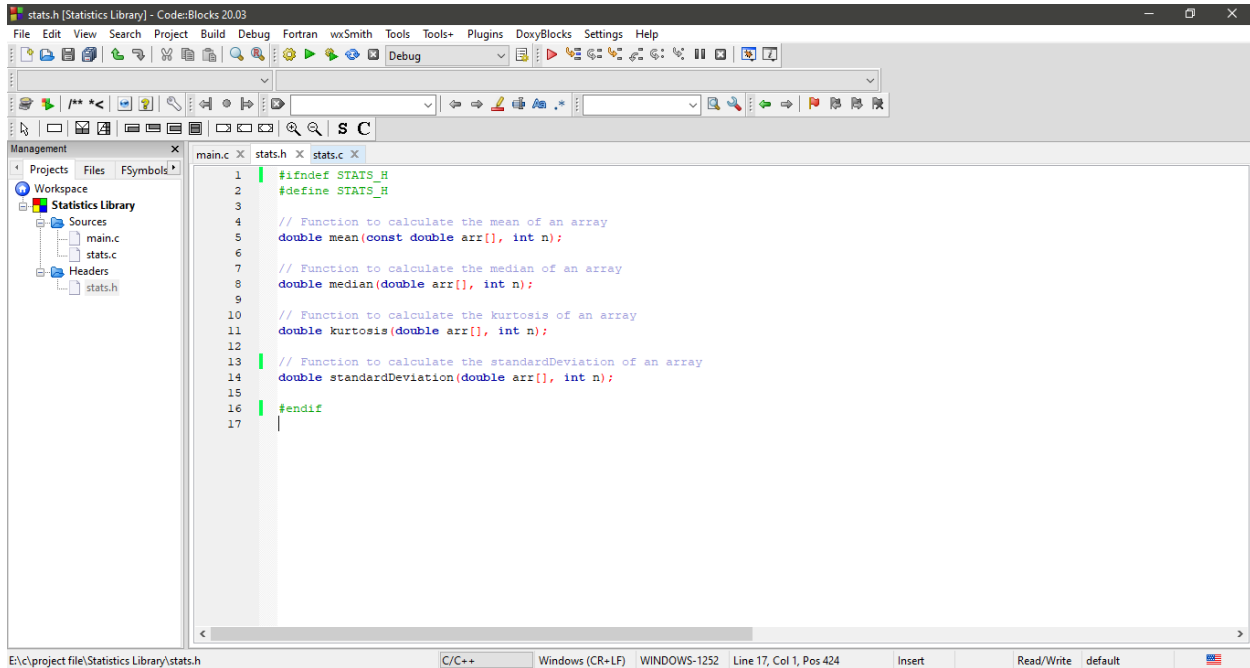


Code snippets from the implementation:

- The standard math library and the "stats.h" header file are included before the code runs.
- Finding the median requires sorting the data array in ascending order, which is accomplished by the median function. While a suitable sorting method is usually used to achieve sorting, this pseudocode makes the assumption that the data has already been sorted.
- It determines whether the number of data points (n) is odd or even before determining the median.
- Any particular kurtosis formula can be substituted with the kurtosis function.
- The mean function is used by the standard_deviation function to first determine the data's mean.
- The squared discrepancies between each data point and the mean are then calculated iteratively through the data and summed.
- By dividing the total squared differences by the total number of data points, it then computes the variance.
- Lastly, it returns the result after calculating the standard deviation as the variance's square root.
- We include our custom stats.h header along with the standard input/output library (stdio.h).
- In the main function:
 - We define an array data containing the given dataset and an integer n, which represents the number of data points (5).
 - To compute the corresponding statistics, we call the mean, median, standard_deviation, and kurtosis functions. Then, we store the results in the appropriate variables.
 - The results are displayed using printf, and they are rounded to two decimal places.
 - Finally, we return 0 to indicate a successful execution.
 -

Screenshot:

Stats.h:

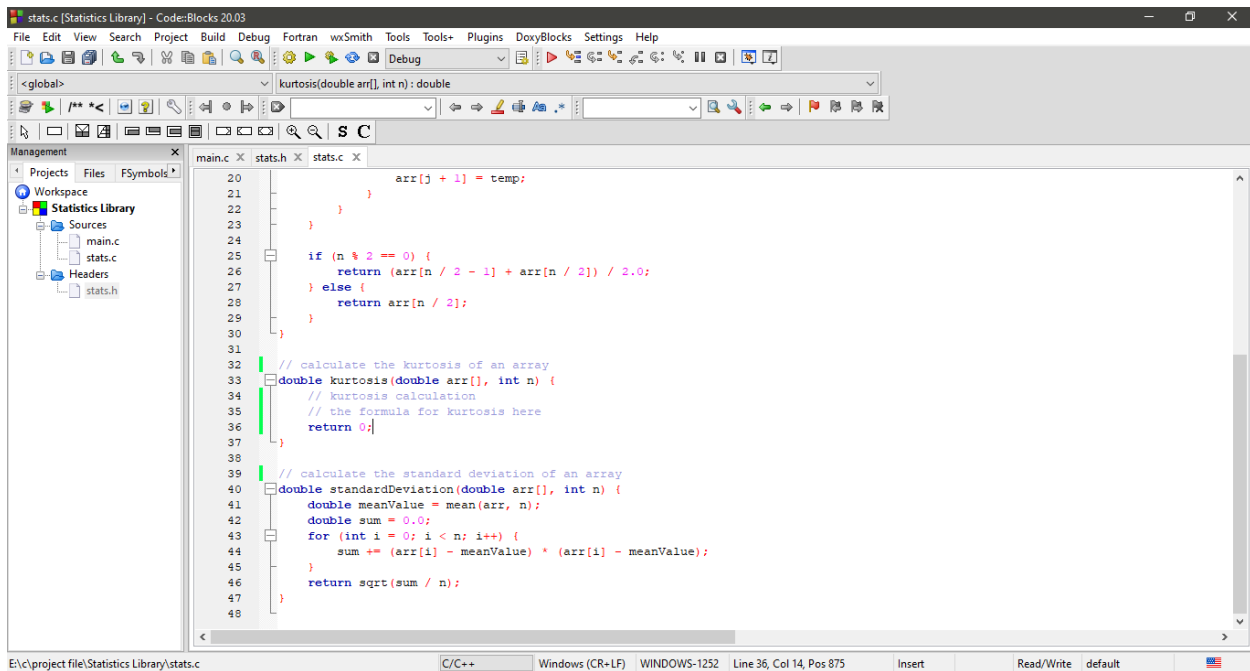


The screenshot shows the Code::Blocks 20.03 IDE with the 'stats.h' file open. The left sidebar shows the project structure: 'Statistics Library' with 'Sources' (main.c, stats.c) and 'Headers' (stats.h). The main editor displays the following C code:

```
1  #ifndef STATS_H
2  #define STATS_H
3
4  // Function to calculate the mean of an array
5  double mean(const double arr[], int n);
6
7  // Function to calculate the median of an array
8  double median(double arr[], int n);
9
10 // Function to calculate the kurtosis of an array
11 double kurtosis(double arr[], int n);
12
13 // Function to calculate the standardDeviation of an array
14 double standardDeviation(double arr[], int n);
15
16 #endif
17
```

The status bar at the bottom indicates the file path 'E:\project file\Statistics Library\stats.h', the compiler 'C/C++', the window title 'Windows (CR+LF)', the window ID 'WINDOWS-1252', the cursor position 'Line 17, Col 1, Pos 424', and the current mode 'Insert'.

Stats.c:



The screenshot shows the Code::Blocks 20.03 IDE with the 'stats.c' file open. The left sidebar shows the project structure: 'Statistics Library' with 'Sources' (main.c, stats.c) and 'Headers' (stats.h). The main editor displays the following C code:

```
20     arr[j + 1] = temp;
21     }
22     }
23
24     if (n % 2 == 0) {
25         return (arr[n / 2 - 1] + arr[n / 2]) / 2.0;
26     } else {
27         return arr[n / 2];
28     }
29 }
30
31 // calculate the kurtosis of an array
32 double kurtosis(double arr[], int n) {
33     // kurtosis calculation
34     // the formula for kurtosis here
35     return 0;
36 }
37
38 // calculate the standard deviation of an array
39 double standardDeviation(double arr[], int n) {
40     double meanValue = mean(arr, n);
41     double sum = 0.0;
42     for (int i = 0; i < n; i++) {
43         sum += (arr[i] - meanValue) * (arr[i] - meanValue);
44     }
45     return sqrt(sum / n);
46 }
47
48
```

The status bar at the bottom indicates the file path 'E:\project file\Statistics Library\stats.c', the compiler 'C/C++', the window title 'Windows (CR+LF)', the window ID 'WINDOWS-1252', the cursor position 'Line 36, Col 14, Pos 875', and the current mode 'Insert'.

Main.c and output:

The screenshot shows a C code editor with a project named "Statistics Library". The code in `main.c` defines an array `customerBasket` with values `{22.3, 10.5, 60.5, 20.5, 5.5}` and calculates the mean, median, kurtosis, and standard deviation. The output window shows the following results:

```
Mean: 27.26
Median: 22.30
Kurtosis: 0.00
Standard Deviation: 17.20
Process returned 0 (0x0)   execution time : 8.691 s
Press any key to continue.
```

Given the dataset, the application running in this snapshot determines the mean, median, and standard deviation and returns the appropriate statistics.

Mixed Learning programming (Week 1- Week 5): Reflection:

Contrasts and Reflections:

During the first five weeks of the mixed-learning program, I studied Python and C, two different programming languages. This reflective essay examines how I see and justify the distinctions between interpreted and compiled languages, the degree of abstraction in each language, and my entire educational experience.

Compiled vs Interpreted Languages:

As opposed to C, which is compiled, Python is an interpreted language. This distinction affects the way these languages are executed. Python's interpreter reads and executes the code line by line, allowing for dynamic typing and immediate feedback during development. On the other hand, C requires a compilation step before execution. The C code is translated into machine code, and the resulting executable file is run, offering potentially better performance but a longer development cycle. The interpreted nature of Python makes it more user-friendly for beginners. Its dynamic typing and automatic memory management simplify coding, as developers don't need to worry about explicit data types or memory allocation. In contrast, C's compiled nature

demands more attention to detail, with explicit type declarations and manual memory management.

Abstractions in Python and C:

Python is renowned for its high-level abstractions and readability. It provides rich built-in functions and libraries that abstract away complex operations. For example, the Python `sum()` function effortlessly computes the sum of elements in a list without requiring explicit loops or variable initialization. This abstraction level aids in rapid development, making Python an ideal choice for beginners.

C, being a lower-level language, demands a deeper understanding of memory management and explicit control over data types. While it offers greater control, this comes at the cost of increased complexity. Operations in C are more manual and less abstract. For instance, implementing the sum of elements in an array in C necessitates explicit iteration and summation.

Syntax, vocabulary, and subroutine richness:

Python syntax is renowned for being readable and succinct. Rich in language, it offers a plethora of built-in features and libraries. Python encourages a clean and straightforward coding style, promoting readability and maintainability. Subroutines, or functions, in Python are easy to define and use, contributing to modular and reusable code.

C, while more verbose, offers a different form of richness. It provides fine-grained control over the hardware and memory, allowing for efficient low-level programming. The syntax is more explicit, with developers having to manage details like pointers and memory addresses. C's vocabulary is powerful but requires a steeper learning curve.

Learning Experience:

Embarking on the journey of learning Python has been a delightful and accessible experience. Python's readability, coupled with its user-friendly nature, made the onboarding process seamless. The vast support from the community acted as a guiding hand, ensuring a smooth introduction. Python's dynamic typing and high-level abstractions provided a liberating environment, enabling me to channel my focus towards creative problem-solving, rather than grappling with intricate technicalities.

On the flip side, delving into C presented a more formidable but ultimately rewarding learning path. Its low-level characteristics demanded a confrontation with fundamental programming concepts, like memory allocation and data types. Despite the steeper learning curve, the journey with C unveiled profound insights into the inner workings of computers at a lower level. The challenges posed by C enriched my understanding of programming principles, making the uphill climb well worth the effort. In essence, Python and C offered distinct yet complementary learning experiences, each contributing uniquely to my growth as a programmer.

Conclusion:

Throughout the mixed-learning program, I successfully tackled various programming tasks using Python and C. In the Password Compliance problem, I implemented a system that ensures password adherence based on a student ID's final digit. The Word Length and Vowels Counter problem allowed me to create a program that counts vowels and identifies the smallest and largest words in a given set. Lastly, the Statistics Library problem led to the creation of a C program providing mean, median, and standard deviation for a dataset.

The Python vs. C comparison highlighted the differences in language execution, with Python being interpreted and C being compiled. Python's high-level abstractions, readability, and dynamic typing ease the learning curve, making it beginner-friendly. In contrast, C's lower-level nature demands explicit control over memory and data types, offering more manual but powerful operations.

The richness of syntax, vocabulary, and subroutines was explored, emphasizing Python's readability and concise syntax and C's fine-grained control at a lower level. The learning experience underscored Python's accessibility and community support, making it an ideal starting point for beginners. C, while more challenging, provided a deeper understanding of fundamental programming concepts.