COMP104 Notes

Note1

In this code:

assignment_3_Q1.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main() {
    int i = 100;
    int j1 = 2 * i / 3 % 4; // 2
    float j2 = 2 * i / 3 % 4; // 2
    int j3 = 2.4 * i / 3 % 4;
    float j4 = 2.4 * i / 3 % 4;
    return 0;
}
```

Watch our for the casting problems when evaluating an expression, for example, in this code in line 8, and line 8 the compiler gives this issue:

```
expression must have integral or unscoped enum type C/C++ (2140)
(double)(2.399999999999999911)
View Problem (Alt+F8)  Quick Fix... (Ctrl+.)
```

This is because we can't multiply 2.4 "which is a auto detected as **double**" to be multiplied by **i** which is in this case an **"integer"** so the right solution for this program in line 8:

assignment_3_Q1.cpp > main()

```cpp
#include <iostream>
using namespace std;

int main() {
    int i = 100;
    int j1 = 2 * i / 3 % 4; // 2
    float j2 = 2 * i / 3 % 4; // 2
    int j3 = static_cast<int>(2.4 * i / 3) % 4;
    /*
    float j4 = 2.4 * i / 3 % 4;
    */
    return 0;
}
```

Because the value will be copied to **j3** which is an **integer** data type. **NOTE: In this case the modulus operator is valid.**

And the solution for the program in line 9:

assignment_3_Q1.cpp > main()

```cpp
#include <iostream>
using namespace std;

int main() {
    int i = 100;
    int j1 = 2 * i / 3 % 4; // 2
    float j2 = 2 * i / 3 % 4; // 2
    int j3 = static_cast<int>(2.4 * i / 3) % 4;
    float j4 = static_cast<float> (2.4 * i / 3) / 4;
    return 0;
}
```

Note:

1. **we omitted the modulus operator, because In C++, the modulus operator % cannot be used directly with floating-point values (float or double). The modulus operator is designed for integer arithmetic, and its use with floating-point values is not allowed in C++.**
2. **The static cast type is float because the copy will be to j4 which is floating-point data type.**

Note 2

What is the difference between compile-error, run-time error, logic error?

1. **Compile-Time Errors:** Compile-time errors, also known as **syntax errors**, **occur during the compilation phase of the program** when the code violates the rules of the programming language. These errors prevent the program from being successfully compiled, ***meaning the program cannot be translated into machine code and, therefore, cannot be executed.*** Common reasons for compile-time errors include syntax mistakes, missing semicolons, undeclared variables, etc.

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    // Missing semicolon on the next line
    std::cout << "This line will cause a compile-time error" << std::endl
    return 0;
}
```

In this example, the missing semicolon after the first std::cout statement will cause a compile- time error. The compiler will point out the line where the error occurred and its type.

2. **Run-Time Errors:** Run-time errors **occur during the execution of the program** and are caused by unexpected conditions or situations that arise while the program is running. ***These errors are not detected during the compilation phase, so the program compiles successfully.*** However, when the problematic code is executed, the program may crash or behave unexpectedly due to issues such as division by zero, accessing invalid memory, etc.

```cpp
#include <iostream>

int main() {
    int num1 = 10;
    int num2 = 0;
    int result = num1 / num2; // Division by zero
    std::cout << "Result: " << result << std::endl;
    return 0;
}
```

In this example, attempting to divide num1 by num2, which is 0, will cause a run-time error due to division by zero.

3. **Logic Errors:** Logic errors, also known as semantic errors, ***occur when the program runs successfully and produces output without any error messages,*** but the output is not what was expected or intended. These errors happen when there is a flaw or mistake in the algorithm or logic used in the program. ***The program may execute without any issues, but the results will be incorrect.***

```cpp
#include <iostream>

int main() {
    int radius = 5;
    float pi = 3.14;
    float area = radius * radius * pi; // Incorrect calculation of area
    std::cout << "Area: " << area << std::endl;
    return 0;
}
```

In this example, the logic error occurs when the value of pi is approximated as 3.14, but this is not precise enough for accurate calculations. The area of a circle calculated using this value of pi will be incorrect.

Very important Note 3

What is the result of using negative values with the modulus operator?

In mathematics and most programming languages, including C++, the modulus operator (often denoted by `%`) can be used with negative values. The result of the modulus operation on negative values follows certain rules.

In C++, the modulus operation for negative values follows the sign of the dividend (the number being divided). The remainder will always have the same sign as the dividend.

Here are some examples to illustrate the behavior:

“Left always wins”

1. **Positive dividend, positive divisor**: **“+, +”**

```cpp
int result = 10 % 3; // result = 1
```

In this case, the remainder is 1.

2. **Negative dividend, positive divisor: “-, +”**

```cpp
int result = -10 % 3; // result = -1
```

The dividend is -10, and the remainder has the same sign as the dividend (-1).

3. **Positive dividend, negative divisor: “+, -”**

```
int result = 10 % -3; // result = 1
```

The dividend is 10, and the remainder has the same sign as the dividend (1).

4. **Negative dividend, negative divisor: “-, -”**

```
int result = -10 % -3; // result = -1
```

In this case, the dividend is -10, and the remainder has the same sign as the dividend (-1).

So, the modulus operator in C++ handles negative values in a way that ensures the remainder takes the sign of the dividend. This behavior is consistent with the mathematical definition of the modulus operation.

Note 4:

You can use the static cast to convert the integer to the type of the defined enumeration type, e.g:

assignment_4_Q3.cpp > ...

```cpp
#include <iostream>
#include <string>
using namespace std;

enum GameOption {
    Rock,
    Paper,
    Scissors
};

string determineWinner(int game1, int game2) {
    game1 = static_cast<GameOption> (game1);
    game2 = static_cast<GameOption> (game2);
    if (game1 == game2)
        return "Draw!";
    switch (game1) {
    case GameOption::Rock:
        if (game2 == GameOption::Scissors)
            return "Player1 wins";
        else
            return "Player2 wins";
        break;
    case GameOption::Scissors:
        if (game2 == GameOption::Paper)
            return "Player1 wins";
        else
            return "Player2 wins";
        break;
    case GameOption::Paper:
        if (game2 == GameOption::Rock)
            return "Player1 wins";
        else
            return "Player2 wins";
        break;
    default:
        return "invalid input";
    }
}

int main() {
    int input1, input2;
    cout << "Rock: 0, Paper: 1, Scissors: 2" << endl;
    cout << "Enter the game Sequence (for exmaple 0 1 as Rock and Paper): ";
    cin >> input1 >> input2;
    if ((input1 < 0 || input1 > 2) || (input2 < 0 || input2 > 2)) {
        cout << "invalid input" << endl;
        return -1;
    }
    string result = determineWinner(input1, input2);
    cout << result << endl;
    return 0;
}
```

For example, In this code, especially in line 12, 13 you will find that we casted the **int** type to the **Game Option** enumerated type.

Note 5:

How to determine the number of digits of the user input ?

```cpp
#include <iostream>
using namespace std;

int main() {
    int userInput;
    cout << "Enter an integer: ";
    cin >> userInput;

    // Handle the case when the user enters a negative number
    if (userInput < 0) {
        userInput = -userInput;
    }

    int numDigits = 1; // Minimum number of digits is 1 for single-digit nu

    while (userInput >= 10) {
        userInput /= 10;
        numDigits++;
    }

    cout << "Number of digits: " << numDigits << endl;

    return 0;
}
```

NOTE:

1. **If the user input is negative value that will affect the division, then if the number is smaller than zero we multiply it by “negative” to get the position value.**
2. **the number of the ‘numDigits‘ in the code must be 1 that determines at least a 1 digit as input to the function.**

Note 6:

In this code:

midterm-solutions > exam1 > Question3-a.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main() {
    int x = 4;
    x += 2 + ++x;
    cout << x << endl;
    return 0;
}
```

What is the value of x?

May be your answer would be that the value of x is 11,

NOTE: evaluate the expression, simplify it until it reach the end simplification expression.

X += 2 + ++x -> x = x + (2 + ++x)

1. **Parentheses will be evaluated first**
2. **The value of x will be incremented from 4 to 5**
3. **The value of x will be add to 2, so 2 + 5 = 7**
4. **X in memory is valued by 5**
5. **The latest expression is 5 + 7 = 12.**

Note 7:

In this code:

midterm-solutions > exam2 > Question3-c.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main() {
    int x = 5;
    x = x++ % 5 || x - 5;
    if (x)
        cout << "True" << endl;
    else
        cout << "False" << endl;
    return 0;
}
```

What is the Boolean will be assigned to x?

May be you say False, but it's actually True, because the first condition "x++ % 5" is 5 % 5 = 0, but in the second condition "6 - 5 = 1", which evaluates the whole condition to "True".

Note 8:

All the switch statement conditions

1. Consider the following snippet of the code.

```
switch (x*2){
    default: cout <<"default,";
    case 4: cout <<"4,";
    case 2: cout <<"2,"; break;
    case 10: cout <<"10,";
}
```

What is the output when

(a) $x = 0$

○ *"no thing will be printed"* ○ default,4,2, ○ default,4,2,10, ○ 4,2, ○ 10, ○ 10,default,

(b) $x = 2$

○ *"no thing will be printed"* ○ default,4,2, ○ default,4,2,10, ○ 4,2, ○ 10, ○ 10,default,

1.

Very Important Note:

Steps for solving the switch statement correctly:

1. Execute the condition and calculate the condition expression
2. If you met the condition value in any case except default:
    a. If it has break statement git out of the switch statement
    b. If it doesn't have a break statement, execute all the conditions under it **including the default condition,** until you met a break statement or until all the conditions is finished
3. If you didn't met the condition
    a. If the default is the last statement only default will be executed
    b. If default isn't the last statement het the default statement and execute everything under it until you met a break.

Final Exams Notes:

final 2020 Notes:

Question1a:

Write a C++ Program to compute the following f(n) for a give n:

$$f(n) = \begin{cases} \sum_{i=1}^{n} \cos(i) & \text{if } n \text{ is even} \\ \sum_{i=1}^{n} \sin(i) & \text{if } n \text{ is odd} \end{cases}$$

Note:

1. In the loop you encounter to solve the problem you have to make I starts from 1.

```
if (n % 2 == 0)
  for (int i = 1; i <= n; i++)

else
  for (int i = 1; i <= n; i++)
```

2. To get the correct result in the terminal you have to define pi = 3.141592654, and multiply it by i then divide by 180, why???

The values π (pi) and 180 are used in the conversion factor to convert between degrees and radians.

The conversion factor is used to transform angles measured in degrees to their equivalent in radians. Here's the rationale behind it:

- There are 360 degrees in a full circle.
- There are 2π radians in a full circle.

So, to convert from degrees to radians, you divide the angle in degrees by 360 and then multiply by 2π:

$$1 \text{ deg} = \frac{2\pi}{360}$$

$$1\, rad = \frac{360}{2\pi}$$

C++ use the rad measurement not the degree like the calculator, to get the same result as the calculator you multiply the rad number by $\frac{angle \cdot 2\pi}{360} = \frac{angle \cdot \pi}{180} = \text{deg}$

So the code for the compute function is:

```cpp
// compute function f(n)
double computeFunction(int n) {
  double pi = 3.14159265358979323846;
  double result = 0.0;

  if (n % 2 == 0)
    for (int i = 1; i <= n; i++)
      result += cos(i * pi / 180.0);
  else
    for (int i = 1; i <= n; i++)
      result += sin(i * pi / 180.0);
  return result;
}
```

Question2a:

How to alternate between "+", and "-", in multiplication inside a loop?

Consider this question:

Write the body of the main of a C++ Program that reads an integer n, then gets the sum of the series:

$$sum = 2^2 - 4^2 + 6^2 - 8^2 + \ldots$$

First approach:

To define a multiplier and multiply it by –1 in each iteration

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main() {
  int n = 0, sum = 0, multiplier = 1;
  cin >> n;

  for (int i = 2; i <= n; i+=2) {
    sum += pow(i, 2) * multiplier;
    multiplier *= -1;    // switch the sign for the next term
  }

  cout << sum << endl;

  return 0;
}
```

Second approach:

You can use `? :` statement to get 1, or –1 each iteration

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main() {
  int n = 0, sum = 0;
  cin >> n;

  for (int i = 2; i <= n; i++)
    sum += pow(i, 2) * (i % 4 == 0 ? -1 : 1);

  cout << sum << endl;

  return 0;
}
```

In this code the multiplication is alternated between –1 and 1

Third approach:

If the loop starts from 0 and its control variable is incremented by 1, then you can use the exponential expression:
$-1^{(n+1)}$*this starts with –1, and keep alternating*

Question1b:

Very Important Question:

How to find if the given number is prime number or not?

Consider this question:

Write a C++ Program to count the number of primers between two given numbers x and y. For example, the number of primes between 4 and 18 is 5 "5, 7, 11, 13, 17", Your code must keep the run using do while loop!

Don't give must attention to the while loop, our focus here is to understand the function that determines if the current number is prime or not!

```cpp
// check if the digit is prime or not
bool isPrime(int n) {
  // 0 and 1 are not prime numbers.
  if (n <= 1)
    return false;

  // Check if the digit is divisible by any number from 2 to sqrt(n)
  for (int i = 2; i <= sqrt(n); i++)
    if (digit % i == 0)
      return false;

  return true;
}
```

- *In this function "isPrime", we check if the number is smaller than 2, because prime numbers starts from 2 up to infinity.*
- *Then we check the divisibility of the numbers starting from 2 up to the square root of n.*

Note: *Why we check up to the square root of n?*

Because if a number has a divisor larger than its square root, it must also have a corresponding divisor smaller than its square root.

$$6 \cdot 6 = 36$$

What if we negated the 6 to be 4

$$y \cdot 4 = 36$$

Then y must be greater than 6, in this case "9"

If there any number divisible by the number except itself or 1 then its not a prime.

The main function:

```cpp
#include <iostream>
#include "myFunctions.h"
using namespace std;

int main() {
  int i = 10;
  if (isPrime(i))
    cout << i << " is a prime number." << endl;
  else
    cout << i << " is not a prime number." << endl;
  return 0;
}
```

The solution of Question2-b is:
prime function:

```
// check if the digit is prime or not
bool isPrime(int n) {
  // 0 and 1 are not prime numbers.
  if (n <= 1)
    return false;

  // Check if the digit is divisible by any number from 2 to sqrt(n)
  for (int i = 2; i <= sqrt(n); i++)
    if (digit % i == 0)
      return false;

  return true;
}
```

Main function:

```
#include <iostream>
#include "myFunctions.h"
using namespace std;

int main() {
  int x = 0, y = 0, counter = 0;
  cout << "x = ";
  cin >> x;
  cout << "y = ";
  cin >> y;

  // count the prime number starting from x to y
  int i = x;
  do {
    if (isPrime(i++))
      counter++;
  } while (i <= y);

  cout << "The number of prime numbers between "
       << x << " and " << y << " is " << counter << endl;
  return 0;
}
```

Important Note:

We use the variable i instead of x directly because we want to print the initial value of x in the output stream.

final 2021 Notes:

Question 2b:

Correct this code:

double x = 1, y = 0;

do;[

y += x % 3; x++;

] while {1/x > 0.01],

Very Important Note to be realized here

You can't perform modulus operations on the data type "double" you have to explicitly transform that double data type variable to "int" then perform the desired operations.

The correction is:

```
#include <iostream>
using namespace std;

int main() {
  double x = 1, y = 0;
  do {
    y += static_cast<int>(x) % 3;
    x ++;
  } while (1/x > 0.01);
  return 0;
}
```

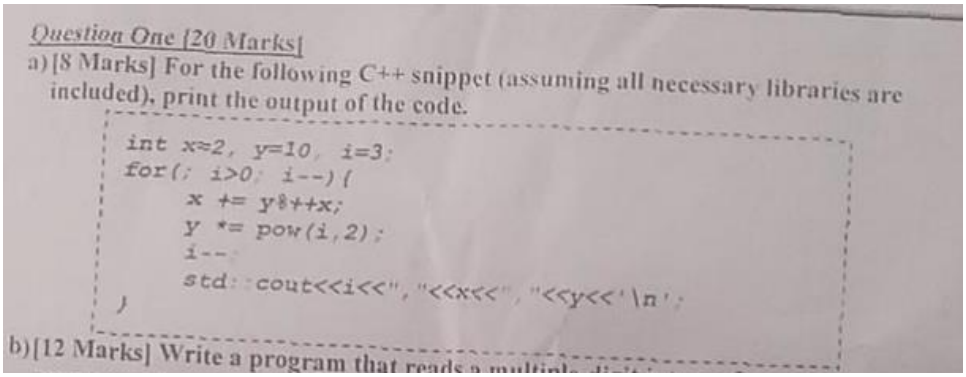
Question 3b:

Very good question:

Write a C++ Program to read two running times (as input) T1 = h1:m1:s1, and T2 = h2:m2:s2, (hours: minutes : seconds) and then finding the difference between them. For example, the difference of the two running times T1 = 90:1:6 and T2 = 92:1:8 is 2:2:2 I.e, 2 hours : 2 minutes : 2 seconds. Your code must keep the run using do while loop.

final 2023 Notes:

Question 1a:



Question One [20 Marks]

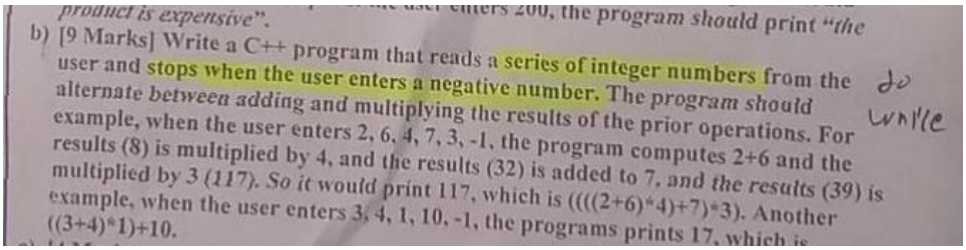
a) [8 Marks] For the following C++ snippet (assuming all necessary libraries are included), print the output of the code.

```
int x=2, y=10, i=3;
for(; i>0; i--){
    x += y%++x;
    y *= pow(i,2);
    i--;
    std::cout<<i<<","<<x<<","<<y<<'\n';
}
```

b)[12 Marks] Write a program that reads a multiple

Note: i is incremented twice.

Question 2b:



product is expensive".

b) [9 Marks] Write a C++ program that reads a series of integer numbers from the user and stops when the user enters a negative number. The program should alternate between adding and multiplying the results of the prior operations. For example, when the user enters 2, 6, 4, 7, 3, -1, the program computes 2+6 and the results (8) is multiplied by 4, and the results (32) is added to 7, and the results (39) is multiplied by 3 (117). So it would print 117, which is ((((2+6)*4)+7)*3). Another example, when the user enters 3, 4, 1, 10, -1, the programs prints 17, which is ((3+4)*1)+10.

do while

```cpp
#include <iostream>
using namespace std;

int main() {
  int ival = 0, total = 1, alternator = 0;

  do {
    cin >> ival;
    if (ival < 0)
      break;
    if (alternator % 2 == 0)
      total *= ival;
    else
      total += ival;
    alternator++;
  } while (true);

  cout << "total = " << total << endl;
  return 0;
}
```

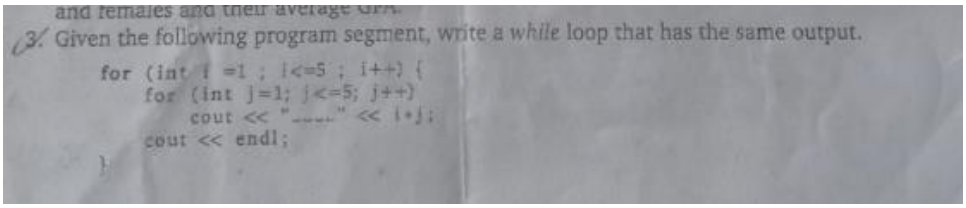
Note: You have to start with multiplication after the alternator or result will not be correct.

Question 3a:

How to create the function "factorial"?

```
int factorial(int val) {
  if (val > 1)
    return factorial(val - 1) * val;
  return 1;
}
```

Note:



When you transfer this code you have to initialize j = 1 under the first loop:

cpp — Copy code

```cpp
int i = 1;
while (i <= 5) {
    int j = 1;
    while (j <= 5) {
        cout << "..." << i * j;
        j++;
    }
    cout << endl;
    i++;
}
```

What is the sample correlation coefficient and how to write it in c++?

The sample correlation coefficient, often denoted as "r", measures the strength and direction of the linear relationship between two sets of data. It ranges between -1 and 1, where -1 indicates a perfect negative correlation, 1 indicates a perfect positive correlation, and 0 indicates no correlation.

To calculate the sample correlation coefficient between two sets of data (let's call them **x** and **y**), you can use the following formula:

$$r = \Sigma((xi - \bar{x}) * (yi - \bar{y})) / \sqrt{(\Sigma(xi - \bar{x})^2 * \Sigma(yi - \bar{y})^2)}$$

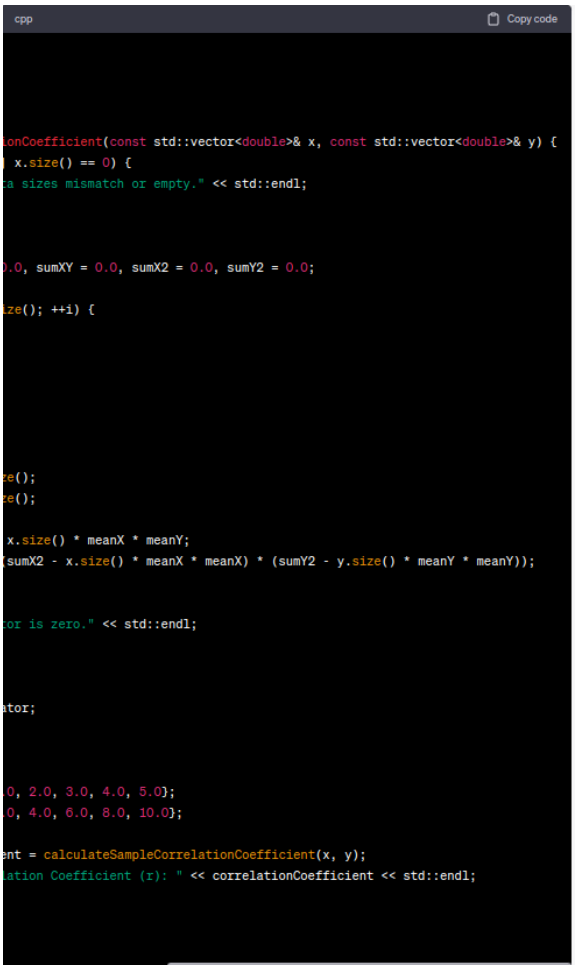
Where:

- **xi** and **yi** are individual data points from the sets **x** and **y**.
- $\bar{x}$ and $\bar{y}$ are the means of sets **x** and **y**, respectively.

Here's how you can calculate the sample correlation coefficient in C++:

cpp Copy code

```cpp
#include <iostream>
#include <cmath>
#include <vector>

double calculateSampleCorrelationCoefficient(const std::vector<double>& x, c
    if (x.size() != y.size() || x.size() == 0) {
        std::cerr << "Input data sizes mismatch or empty." << std::endl;
        return 0.0;
    }

    double sumX = 0.0, sumY = 0.0, sumXY = 0.0, sumX2 = 0.0, sumY2 = 0.0;

    for (size_t i = 0; i < x.size(); ++i) {
        sumX += x[i];
        sumY += y[i];
        sumXY += x[i] * y[i];
        sumX2 += x[i] * x[i];
        sumY2 += y[i] * y[i];
    }

    double meanX = sumX / x.size();
    double meanY = sumY / y.size();

    double numerator = sumXY - x.size() * meanX * meanY;
    double denominator = sqrt((sumX2 - x.size() * meanX * meanX) * (sumY2 -

    if (denominator == 0.0) {
        std::cerr << "Denominator is zero." << std::endl;
        return 0.0;
    }

    return numerator / denominator;
}

int main() {
    std::vector<double> x = {1.0, 2.0, 3.0, 4.0, 5.0};
    std::vector<double> y = {2.0, 4.0, 6.0, 8.0, 10.0};

    double correlationCoefficient = calculateSampleCorrelationCoefficient(x,
    std::cout << "Sample Correlation Coefficient (r): " << correlationCoeffi

    return 0;
}
```



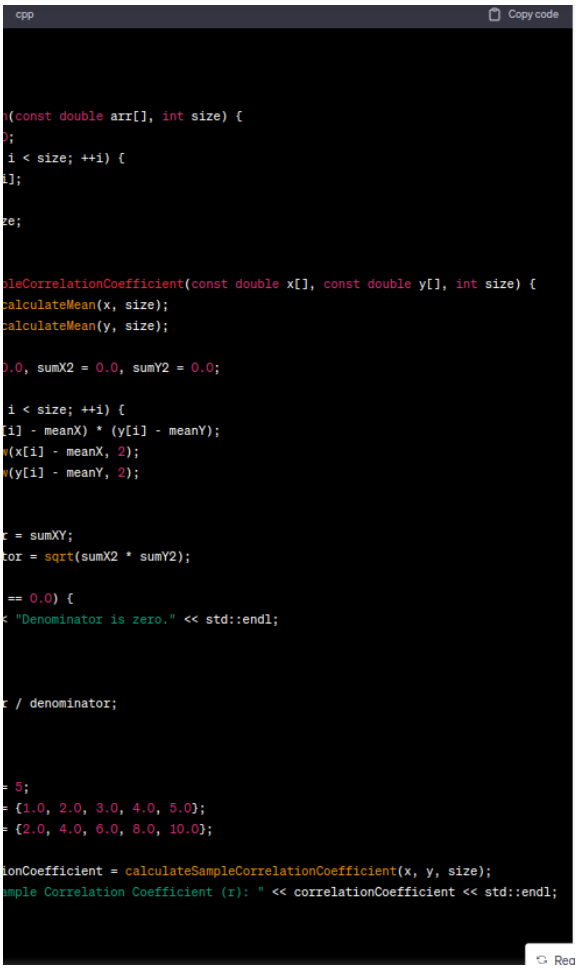
Replace the **x** and **y** vectors with your own data sets, and the code will calculate and output

the sample correlation coefficient.

cpp Copy code

```cpp
#include <iostream>
#include <cmath>

double calculateMean(const double arr[], int size) {
    double sum = 0.0;
    for (int i = 0; i < size; ++i) {
        sum += arr[i];
    }
    return sum / size;
}

double calculateSampleCorrelationCoefficient(const double x[], const double
    double meanX = calculateMean(x, size);
    double meanY = calculateMean(y, size);

    double sumXY = 0.0, sumX2 = 0.0, sumY2 = 0.0;

    for (int i = 0; i < size; ++i) {
        sumXY += (x[i] - meanX) * (y[i] - meanY);
        sumX2 += pow(x[i] - meanX, 2);
        sumY2 += pow(y[i] - meanY, 2);
    }

    double numerator = sumXY;
    double denominator = sqrt(sumX2 * sumY2);

    if (denominator == 0.0) {
        std::cerr << "Denominator is zero." << std::endl;
        return 0.0;
    }

    return numerator / denominator;
}

int main() {
    const int size = 5;
    double x[size] = {1.0, 2.0, 3.0, 4.0, 5.0};
    double y[size] = {2.0, 4.0, 6.0, 8.0, 10.0};

    double correlationCoefficient = calculateSampleCorrelationCoefficient(x,
    std::cout << "Sample Correlation Coefficient (r): " << correlationCoeffi

    return 0;
}
```

Reg


cpp
Copy code
(const double arr[], int size) {
;
i < size; ++i) {
i];
ze;
leCorrelationCoefficient(const double x[], const double y[], int size) {
alculateMean(x, size);
alculateMean(y, size);
.0, sumX2 = 0.0, sumY2 = 0.0;
i < size; ++i) {
[i] - meanX) * (y[i] - meanY);
(x[i] - meanX, 2);
(y[i] - meanY, 2);
= sumXY;
or = sqrt(sumX2 * sumY2);
== 0.0) {
"Denominator is zero." << std::endl;
/ denominator;
= 5;
= {1.0, 2.0, 3.0, 4.0, 5.0};
= {2.0, 4.0, 6.0, 8.0, 10.0};
ionCoefficient = calculateSampleCorrelationCoefficient(x, y, size);
mple Correlation Coefficient (r): " << correlationCoefficient << std::endl;
Rea

How to insert elements in an array given a specified position:

cpp Copy code

```cpp
#include <iostream>

const int MAX_SIZE = 100;

void insertElement(int arr[], int& size, int position, int element) {
    if (size >= MAX_SIZE) {
        std::cerr << "Array is already at maximum capacity." << std::endl;
        return;
    }

    if (position < 0 || position > size) {
        std::cerr << "Invalid position." << std::endl;
        return;
    }

    // Shift elements to the right to make space
    for (int i = size; i > position; --i) {
        arr[i] = arr[i - 1];
    }

    // Insert the new element
    arr[position] = element;
    size++;
}

int main() {
    int arr[MAX_SIZE] = {1, 2, 3, 5, 6};
    int size = 5;
    int newPosition = 3;  // Index where you want to insert the new element
    int newElement = 4;   // Element to be inserted

    insertElement(arr, size, newPosition, newElement);

    std::cout << "Array after insertion: ";
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

How delete element form an array given a position:

cpp Copy code

```cpp
#include <iostream>

const int MAX_SIZE = 100;

void deleteElement(int arr[], int& size, int position) {
    if (position < 0 || position >= size) {
        std::cerr << "Invalid position." << std::endl;
        return;
    }

    // Shift elements to the left to fill the gap
    for (int i = position; i < size - 1; ++i) {
        arr[i] = arr[i + 1];
    }

    size--;
}

int main() {
    int arr[MAX_SIZE] = {1, 2, 3, 4, 5};
    int size = 5;
    int positionToDelete = 2;  // Index of the element to delete

    deleteElement(arr, size, positionToDelete);

    std::cout << "Array after deletion: ";
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```