

## Exercise 1

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

int main() {
    // Create a map of key locations on the keyboard
    vector<string> keyboard = {
        "qwertyuiop",
        "asdfghjkl",
        "zxcvbnm"
    };

    // Read the input characters from the user
    string input;
    cout << "Enter characters: ";
    getline(cin, input);

    if (input.empty()) {
        std::cout << "Input is empty." << std::endl;
        return 0;
    } else if (!std::all_of(input.begin(), input.end(), [](char c)
{ return std::isalpha(c); }))) {
        std::cout << "Input contains non-alphabetic characters." <<
std::endl;
        return 0;
    }

    // Create a vector of key locations for each input character
    vector<vector<pair<int, int>>> locations(input.size());
    for (int i = 0; i < input.size(); i++) {
        for (int j = 0; j < keyboard.size(); j++) {
            for (int k = 0; k < keyboard[j].size(); k++) {
                if (keyboard[j][k] == tolower(input[i])) {
                    locations[i].push_back(make_pair(j, k));
                }
            }
        }
    }
}
```

```

    // Create a matrix to represent the keyboard and mark the
    locations of the keys that can print the input characters
    //    vector<vector<char>> matrix(keyboard.size(),
    vector<char>(keyboard[0].size(), '.'));

    vector<vector<char>> matrix = {{'.', '.', '.', '.', '.', '.',
    '.', '.', '.', '.'},
                                {' ', '.', '.', '.', '.', '.',
    '.', '.', '.'},
                                {' ', ' ', '.', '.', '.', '.',
    '.', '.'}};

    for (int i = 0; i < locations.size(); i++) {
        for (auto loc : locations[i]) {
            if (loc.first == 0) {
            }
            else if (loc.first == 1) {
                loc.second = loc.second + 1;
            }
            else {
                loc.second = loc.second + 2;
            }
            matrix[loc.first][loc.second] = 'o';
        }
    }

    // Display the keyboard matrix
    for (int i = 0; i < matrix.size(); i++) {
        for (int j = 0; j < matrix[i].size(); j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
        for (int k = 0; k < i; k++) {
            cout << " ";
        }
    }

    return 0;
}

```

```
\\wsl$\Ubuntu\home\jackgloves\L2_workspace\ex1.exe
Enter characters: smu
. . . . . 0 . . .
. 0 . . . . .
. . . . . 0

Process finished with exit code 0
```

## Exercise 2

```
#include <iostream>
#include <cmath>
#include <stdexcept>

bool isPrime(int m) {
    if (m <= 1) {
        return false; // 1 and below are not prime numbers
    }

    for (int i = 2; i <= sqrt(m); i++) {
        if (m % i == 0) {
            return false; // m is divisible by i, hence it's not a
prime number
        }
    }

    return true; // m is not divisible by any number from 2 to
sqrt(m), hence it's a prime number
}

void goldbach(int m) {
    if (m <= 2 || m % 2 != 0) {
        throw std::invalid_argument("Input must be a positive even
integer.");
    }

    bool found = false;

    for (int i = 2; i <= m/2; i++) {
        if (isPrime(i) && isPrime(m-i)) {
            std::cout << m << " = " << i << " + " << m-i << std::endl;
            found = true;
        }
    }
}
```

```

    }

    if (!found) {
        std::cout << "Goldbach's conjecture is false for " << m << "."
<< std::endl;
    }
}

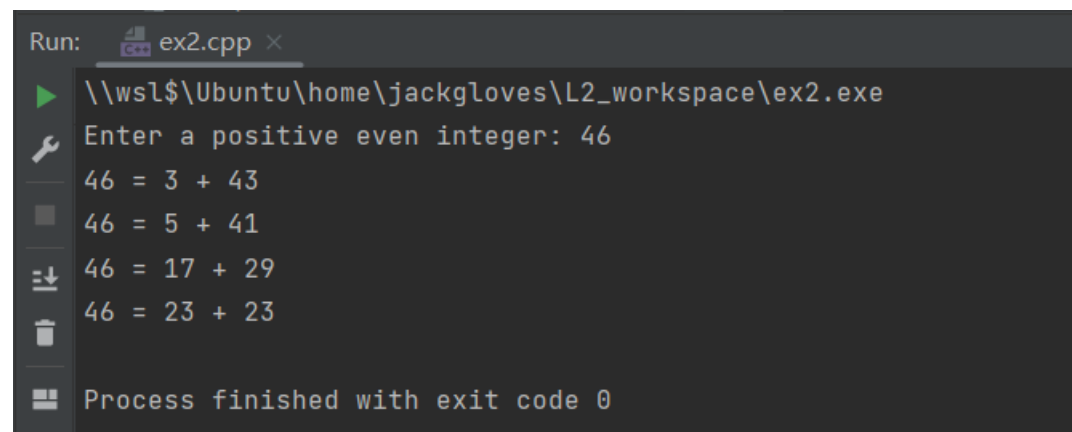
int main() {
    int m;

    std::cout << "Enter a positive even integer: ";
    std::cin >> m;

    try {
        goldbach(m);
    } catch (const std::invalid_argument& e) {
        std::cerr << "Invalid input: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}

```



```

Run: ex2.cpp x
\\wsl$\\Ubuntu\\home\\jackgloves\\L2_workspace\\ex2.exe
Enter a positive even integer: 46
46 = 3 + 43
46 = 5 + 41
46 = 17 + 29
46 = 23 + 23
Process finished with exit code 0

```

### Exercise 3

```

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

// Define the standard
const double PI = 3.141592653589793238463;

// Approach a: Leibniz formula

```

```

double Pi1(int n) {
    if (n == 0) {
        return 4;
    } else {
        double sign = (n % 2 == 0) ? 1 : -1;
        return sign * (4.0 / (2*n + 1)) + Pi1(n-1);
    }
}

// Approach b: Nilakantha series
double Pi2(int nTerm) {
    int n = 1;
    double pi = 3;

    while (true) {
        double term = 4.0 / (2*n * (2*n+1) * (2*n+2));
        if (n % 2 == 0) {
            pi -= term;
        } else {
            pi += term;
        }
        if (n > nTerm-1) {
            break;
        }
        n++;
    }

    return pi;
}

// Approach c: Machin-like formula
double arctan(double x, int nTerm) {
    double temp = 0;
    int n = 0;

    while (true) {
        double term = (1.0 / (2.0 * n + 1.0)) * pow(x, (2.0 * n + 1.0));
        if (n % 2 == 0) {
            temp += term;
        } else {
            temp -= term;
        }
        if (n > nTerm - 1) {

```

```

        break;
    }
    n++;
}
return temp;
}

double Pi3(int nTerm) {
    double pi = 4.0 * (4.0 * arctan((1.0/5.0), nTerm) -
arctan((1.0/239.0), nTerm));
    return pi;
}

//main function here
int main() {
    int n;
    cout << "Enter the number of terms in the series: ";
    cin >> n;

    double pi1 = Pi1(n);
    double error1 = fabs(PI - pi1);
    cout << "Approximation of pi using Leibniz formula and using " <<
n << " terms = " << pi1 << endl;
    cout << "The error of using Leibniz formula and using " << n << "
terms = " << error1 << endl;

    double pi2 = Pi2(n);
    double error2 = fabs(PI - pi2);
    cout << "Approximation of pi using Nilakantha series and using "
<< n << " terms = " << pi2 << endl;
    cout << "The error of using Nilakantha series and using " << n <<
" terms = " << error2 << endl;

    double pi3 = Pi3(n);
    double error3 = fabs(PI - pi3);
    cout << "Approximation of pi using Machin-like formula and using
" << n << " terms = " << pi3 << endl;
    cout << "The error of using Machin-like formula and using " << n
<< " terms = " << error3 << endl;

    return 0;
}
//

```

Run: ex3.cpp x

▶ \\wsl\$\Ubuntu\home\jackgloves\L2\_workspace\ex3.exe

🔧 Enter the number of terms in the series: 999

Approximation of pi using Leibniz formula and using 999 terms = 3.14059

■ The error of using Leibniz formula and using 999 terms = 0.001

⇅ Approximation of pi using Nilakantha series and using 999 terms = 3.14159

🗑 The error of using Nilakantha series and using 999 terms = 1.11957e-06

Approximation of pi using Machin-like formula and using 999 terms = 3.14159

📊 The error of using Machin-like formula and using 999 terms = 8.88178e-16