Modul 3 (lanjutan) Programming

Multifiles C++ Program

Dalam perjalanan Anda sebagai developer, Anda akan sering sekali menemukan implementasi program yang terdiri atas kumpulan-kumpulan subprogram dengan file yang berbeda. Keuntungan dari memecah program menjadi kumpulan-kumpulan subprogram adalah

- Modularity
  Dengan memecah menjadi subprogram kecil, codebase akan menjadi modular dan lebih mudah dipahami karena program yang awalnya besar dapat dipecah sesuai dengan pengelompokkan dari task yang ingin dijalankan oleh program.

- Readability and Maintainability
  Perhatikan bahwa code dipecah sesuai dengan pengelompokkan task-nya sehingga akan memudahkan kita untuk membuat perubahan pada modul tertentu karena perubahan-perubahan kecil yang dilakukan pada modul tersebut, tidak akan merubah keseluruhan codebase secara langsung.

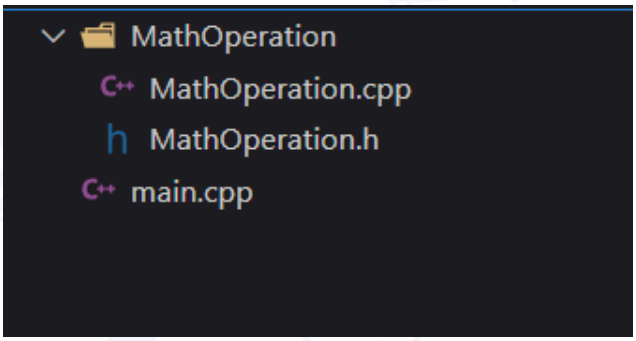
- Collaborative Development
  Dengan pendekatan code yang modular ini, Anda akan dimudahkan ketika bekerja secara paralel dengan rekan Anda karena code yang modular memungkinkan kita untuk secara simultan, mengerjakan modul yang berbeda tanpa mengganggu code rekan Anda.

- Code Organization
  Memecah program menjadi file-file yang lebih kecil kemudian file-file tersebut dikelola berdasarkan fitur atau fungsionalitasnya dapat membantu menciptakan struktur kode yang logis dan terorganisir. Hal ini dapat memudahkan Anda untuk memahami dan ikut berkontribusi pada proyek yang telah ada ataupun yang sedang dikembangkan.

Mari kita bahas mengenai bagaimana cara melakukan multifile program di C++. Contohnya misal saya ingin membuat program sederhana untuk melakukan operasi dasar matematika

Step 1: Membuat Header Filse (‘.h’)

Header file adalah tempat untuk meletakkan deklarasi dari function, class, atau konstanta-konstanta yang diperlukan oleh program. Peletakkan file header file biasanya satu folder dengan pendefinisian function atau class-nya. Misal pada contoh sederhana ini struktur file-nya seperti berikut



Di workspace ada file main.cpp kemudian didalamnya ada folder yang menyimpan file MathOperation.cpp dan MathOpearation.h. Sebenarnya tidak ada aturan eksplisit mengenai bagaimana struktur dari file-filenya harus seperti apa, tetapi sebuah kebiasaan baik jika pembagian filenya didasarkan atas tugas atau fungsionalitas dari subprogramnya. Berikut adalah contoh isi dari header file

```
#ifndef MATH_OPERATIONS_H
#define MATH_OPERATIONS_H

int add(int a, int b);
int subtract(int a, int b);

class Vector
{
private:
    int vectorSize;
    float *placeHolderArr;

public:
    Vector(int vectorSize, float *arr);
    float vectorMagnitude();
    void printVector();
};

#endif
```

Perhatikan macro penjaga tersebut

```
#ifndef MATH_OPERATIONS_H
#define MATH_OPERATIONS_H
...

#endif
```

Macro tersebut merupakan hal penting untuk diikutkan dalam header file karena macro tersebut akan bertindak sebagai penjaga agar tidak ada pendefinisian yang di-execute dua kali. Pada contoh di atas, `#ifndef MATH_OPERATIONS_H`, menggunakan nama `MATH_OPERATIONS_H`. Sebenarnya pemberian nama tersebut bisa sembarang, tetapi konvensinya adalah sesuatu yang related dengan nama file-nya.

Step 2: Implement Source Files (‘.cpp’)

Untuk setiap modul yang dideklarasi pada header file, haruslah ada pendefinisian dari class atau function tersebut. Pendefinisian tersebut diletakkan pada source file ini. Contohnya dibuatlah file bernama MathOperation.cpp yang letaknya satu folder dengan MathOperation.h, berikut adalah isi dari MathOperation.cpp

```
#include "MathOperation.h"
#include <cmath>
#include <iostream>

int add(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}

Vector::Vector(int vectorSize, float *arr)
{
    this->vectorSize = vectorSize;
    this->placeHolderArr = arr;
}
```

```
float Vector::vectorMagnitude()
{
    float tempSum = 0;
    for (int i = 0; i < this->vectorSize; i++) {
        tempSum += placeHolderArr[i] * placeHolderArr[i];
    }
    return sqrt(tempSum);
}

void Vector::printVector()
{
    std::cout << '[';
    for (int i = 0; i < this->vectorSize; i++) {
        std::cout << this->placeHolderArr[i] << ' ';
    }
    std::cout << "]\n";
}
```

Perhatikan bahwa dalam source file tersebut, Anda harus mencantumkan statement `#include "MathOperation.h"`. Jika Anda tidak mencantumkan statement tersebut maka C++ akan bingung di mana letak deklarasinya dan akan memberikan error.

Step 3: Membuat Main Program

Dalam main program, lakukan include pada file header untuk modul-modul yang ingin Anda gunakan. Berikut adalah contoh isi dari main.cpp

```
#include <iostream>
#include "MathOperation/MathOperation.h"

int main()
{
    int result_add = add(5, 3);
    int result_subtract = subtract(8, 2);
    float arr[3] = {1.0, 2.0, 3.0};
    Vector vect1(3, arr);

    std::cout << "Addition: " << result_add << std::endl;
    std::cout << "Subtraction: " << result_subtract << std::endl;
    std::cout << '\n';

    vect1.printVector();
    std::cout << "Magnitude of the vector is " << vect1.vectorMagnitude() << '\n';
    return 0;
}
```

Perhatikan bahwa saat Anda include header file, syntax untuk include, secara default adalah menggunakan alamat relatif. Karena kita meletakkan header file dalam folder MathOperation, maka dalam include di main.cpp statemen include-nya menjadi

```
#include "MathOperation/MathOperation.h"
```

Step 4: Compiling

Terdapat berbagai cara untuk meng-compile code yang telah kita buat. Salah satu yang paling simple-nya adalah dengan menggunakan perintah seperti berikut

```
g++ main.cpp MathOperation/MathOperation.cpp -o my_program
```

Perhatikan bahwa lokasi file dalam perintah untuk compiling pada contoh di atas merupakan alamat relatif dari lokasi folder Anda sekarang. Secara umum berikut adalah sruktur perintah untuk compiling

```
namaCompiler [Lokasi source file (.cpp)] -o [Nama executable]
```

Step 5: Run the Program

Perintah untuk run

```
./my_program
```