



# Array & Function

## Array

Array / Larik adalah cara untuk menyimpan elemen **tipe data (int, float, ...)** yang sama. Array dapat di-deklarasi dengan menggunakan ‘Kurung Siku’ (`[]`) dan ukuran dari array tersebut bersifat tetap (Tidak dapat diubah / di tetapkan ketika mendeklarasi array)

```
#include <iostream>

int main() {
    // Deklarasi array beserta isinya
    int my_array[5] = {1, 2, 3, 4, 5};

    // Output dari array
    for (int i = 0; i < 5; i++) {
        std::cout << my_array[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Disini kita membuat sebuah array bertipe data integer bernama `my_array`, mengisi array tersebut dengan nilai 1 sampai 5. Kemudian kita gunakan perulangan `for` untuk mengiterasi elemen dari array tersebut lalu di output lewat `cout`.

**Note:**

Bisa dilihat array sendiri dimulai dari index ke-0, jadi lebih baik jika mulai dari 0.

*Apakah yang terjadi jika kalian mengakses index variable yang tidak anda isi sebelumnya?*

```
#include <iostream>

int main() {
    /* Program ini akan menyimpan kode ASCII (Binary) dari karakter alfabet
       kedalam bentuk tipe data integer */
    int my_array[5];
    my_array[0] = 2;
    my_array[1] = 3;
    // my_array[2] = 'C'; // Lihat index ke-2 ini tidak di-isi
    my_array[3] = 5;
    my_array[5] = 9; // Index-5 melewati batas dari index yang telah dibuat

    // Output dari array
    for (int i = 0; i < 5; i++)
    {
        printf("Array index ke-%d = %d\n", i, my_array[i]);
    }

    return 0;
}
```

Output:  
Array index ke-0 = 2  
Array index ke-1 = 6  
Array index ke-2 = 0  
Array index ke-3 = 3  
Array index ke-4 = 0

**Penjelasan:**

Lihat pada index ke-2 dan ke-4, nilai index tersebut 0 karena kita tidak mengisi nilai pada index tersebut (*atau disebut dengan NULL*). Hal tentang NULL dan pemberian argumen melalui array akan dibahas pada materi *pointer* agar kepala kalian tidak pusing terlebih dahulu 😊. Jadi pada intinya selalu mulai index array dari 0 dan jangan sampai ada index yang kosong.

## Matrix Array 2-Dimensi

## Array 2 Dimensi

Array 2 dimensi adalah array yang memiliki dua dimensi atau lebih. Array 2 dimensi dapat digunakan untuk merepresentasikan matriks atau tabel. Setiap elemen dalam array 2 dimensi diidentifikasi oleh dua indeks: indeks baris dan indeks kolom.

Untuk mendeklarasikan array 2 dimensi dalam C++, kita dapat menggunakan sintaksis berikut:

```
tipe_data nama_array[baris][kolom];
```

Contoh:

```
int my_array[3][4] = {  
    {1, 2, 3, 4},  
    {5, 6, 7, 8},  
    {9, 10, 11, 12}  
};
```

Dalam contoh di atas, kita mendeklarasikan array 2 dimensi `my_array` dengan 3 baris dan 4 kolom. Kita menginisialisasikan array dengan angka-angka yang disusun dalam bentuk matriks 3x4.

Untuk mengakses elemen-elemen dalam array 2 dimensi, kita dapat menggunakan dua indeks: indeks baris dan indeks kolom. Contoh:

```
int elemen = my_array[1][2];
```

Dalam contoh di atas, kita mengakses elemen di baris ke-1 dan kolom ke-2 dari array `my_array`, yang bernilai 7.

## Sub Program

Sebuah fungsi di dalam C++ adalah sebuah blok kode yang melakukan tugas tertentu dan dapat dipanggil dari bagian lain dari program. Fungsi dapat mengambil parameter input dan mengembalikan nilai output, atau dapat memodifikasi keadaan program melalui efek samping.

Untuk mendefinisikan sebuah fungsi di dalam C++, Anda perlu menentukan namanya, tipe pengembalinya, dan daftar parameter (jika ada), diikuti dengan tubuh fungsi yang diapit oleh kurung kurawal `{}`. Berikut adalah contoh dari sebuah fungsi sederhana yang mengambil dua bilangan bulat sebagai input dan mengembalikan jumlahnya:

```
int tambah(int a, int b) {
    return a + b;
}
```

Di sini, kita mendefinisikan sebuah fungsi yang disebut `tambah` yang mengambil dua bilangan bulat (`a` dan `b`) sebagai parameter input dan mengembalikan jumlahnya sebagai sebuah bilangan bulat. Untuk memanggil fungsi ini dari bagian lain dari program, kita cukup menggunakan namanya dan memberikan parameter input yang dibutuhkan:

```
int hasil = tambah(3, 4);
```

Di sini, kita memanggil fungsi `tambah` dengan nilai 3 dan 4 sebagai parameter input, dan menyimpan hasilnya dalam variabel `hasil`.

#### Note:

Void (Non-value returning) sama seperti regular function namun tanpa harus mengembalikan hasil dari fungsi tersebut, void cocok digunakan untuk fungsi yang melakukan operasi print di dalamnya atau operasi dengan pointer (Pointer bakal dibahas di pertemuan berikutnya)

Salah satu manfaat dari penggunaan *function* ini adalah kita bisa lebih mudah untuk *re-factoring* (Meringkas kode menjadi lebih singkat tanpa mengganggu program) sehingga lebih mudah untuk di deteksi ketika ada kesalahan, Perhatikan kode berikut

```
#include <iostream>

int main() {
    // Program untuk melihat apakah siswa lulus mata kuliah atau tidak
    int grade = 90;
```

```

if (grade >= 60) {
    if (grade >= 90) {
        cout << "Congratulations on your A grade!\n";
    } else if (grade >= 80) {
        cout << "Congratulations on your B grade!\n";
    } else if (grade >= 70) {
        cout << "Congratulations on your C grade!\n";
    }
    return true;
} else {
    cout << "Sorry, you have failed the course.\n";
    return false;
}

return 0;
}

```

Kita bisa membuat fungsi `check_grade` untuk memisahkan if-else statement dari fungsi `main`

```

#include <iostream>

void check_grade(int grade) {

    if (grade >= 60) {
        if (grade >= 90) {
            cout << "Congratulations on your A grade!\n";
        } else if (grade >= 80) {
            cout << "Congratulations on your B grade!\n";
        } else if (grade >= 70) {
            cout << "Congratulations on your C grade!\n";
        }
        return true;
    } else {
        cout << "Sorry, you have failed the course.\n";
        return false;
    }

}

int main() {
    // Program untuk melihat apakah siswa lulus mata kuliah atau tidak
    int grade = 90;

    if (check_grade(grade)) {
        cout << "Congrats on passed in this class\n";
    }

    return 0;
}

```

Kita bisa membuatnya lebih sederhana lagi agar lebih mudah untuk dibaca

```
#include <iostream>

void check_grade(int grade) {

    // Use inversion to switch an early return (no else statement)
    if (grade < 60) {
        cout << "Sorry, you have failed the course.\n";
        return false;
    }

    if (grade >= 90) {
        cout << "Congratulations on your A grade!\n";
    } else if (grade >= 80) {
        cout << "Congratulations on your B grade!\n";
    } else if (grade >= 70) {
        cout << "Congratulations on your C grade!\n";
    }
    return true;
}

int main() {
    // Program untuk melihat apakah siswa lulus mata kuliah atau tidak
    int grade = 90;

    if (check_grade(grade)) {
        cout << "Congrats on passed in this class\n";
    }
}
```

Tujuan dari cara ini adalah agar *code* kita tidak terlalu indentasi ke-dalam sehingga memudahkan untuk memonitoring *code*.

"If you need more than 3 levels of indentation, you're screwed anyway,  
and should fix your program."  
– Linus Torvalds

