Modul Praktikum

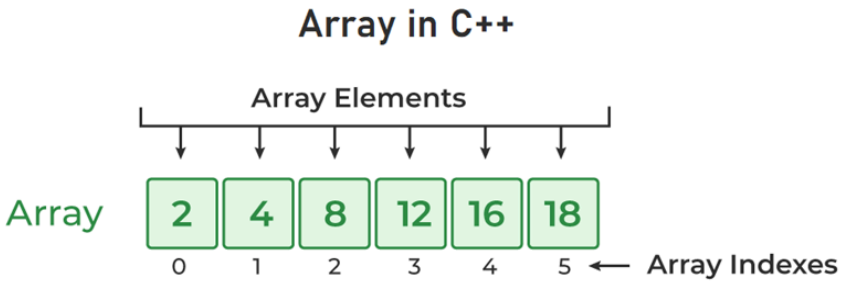
SDAA

2

ARRAY, STRUCT, DAN POINTER

ARRAY

Array adalah kumpulan elemen dengan tipe data yang sama yang disimpan dalam lokasi memori yang berdekatan. Dengan menggunakan array, kita dapat menyimpan beberapa nilai dalam satu variabel, sehingga memudahkan pengelolaan dan manipulasi data terkait. Setiap elemen dalam array diakses menggunakan indeks, dengan elemen pertama berada pada indeks 0.



Gambar 1. Array

Dalam array ini pula, terdapat sebuah karakteristik penting yang harus kita ketahui, yaitu sebagai berikut:

- Ukuran Tetap
  Setelah ukuran array ditetapkan, ukuran tersebut tidak dapat diubah lagi. Ukuran array ditentukan pada saat kode dikompilasi, yang berarti jumlah elemen dalam array harus ditentukan pada saat deklarasi array.
- Bertipe Data Sama
  Semua elemen yang berada pada array harus memiliki tipe yang sama, contohnya, semua bertipe integer, float, atau char.
- Lokasi Memori Dekat
  Elemen array tersimpan pada lokasi memori yang berurutan, sehingga untuk mengakses array sangatlah mudah.

DEKLARASI DAN INISIALISASI PADA ARRAY

Dengan sebuah tipe data *T*, maka *T*[*ukuran*] dapat dibaca sebagai "array bertipe *T* dengan elemen sebanyak *ukuran*." Elemen ini memiliki indeks yang dimulai dari *0* hingga *ukuran-1*. Contohnya adalah sebagai berikut:

```
tipe_data  nama_array[ukuran];
```

Deklarasi Array

```
// sebuah array berisi 3 float, v[0], v[1], v[2]
float v[3];

// sebuah array berisi 32 pointer ke char a[0]...a[31]
char* a[32];

// sebuah cara deklarasi yang salah
void f(int i) {
    int v1[i];
}
```

Inisialisasi Array

```
// inisialisasi dengan menggunakan nilai
int numbers[5] = {1, 2, 3, 4, 5};

// tanpa menggunakan ukuran
int numbers[] = {1, 2, 3, 4, 5};

/*
inisialisasi array tanpa nilai
lengkap, sehingga ekuivalen
int numbers[5] = {1, 2, 0, 0, 0};
*/
int numbers[5] = {1, 2};

// inisialisasi dengan nilai 0
int numbers[5] = {0};

/*
inisialisasi yang berisikan terlalu
banyak nilai akan menghasilkan error
*/
int numbers[5] = {1, 2, 3, 4, 5, 6};
```

ARRAY MULTIDIMENSI

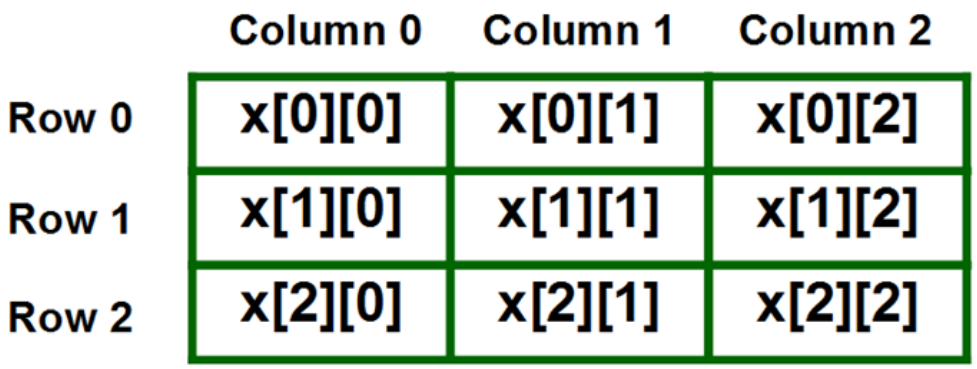
Array yang dideklarasikan dengan lebih dari satu dimensi disebut array multidimensi. Array multidimensi yang paling banyak digunakan adalah array 2D dan array 3D. Array ini umumnya direpresentasikan dalam bentuk baris dan kolom.

```
tipe_data    nama_array[ukuran_1][ukuran_2]…[ukuran_n];
```

- Array 2 Dimensi

  Array dua dimensi adalah pengelompokan elemen yang disusun dalam baris dan kolom. Setiap elemen diakses menggunakan dua indeks: satu untuk baris dan satu untuk kolom, yang membuatnya mudah divisualisasikan sebagai tabel.

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



Gambar 2. Array 2 Dimensi

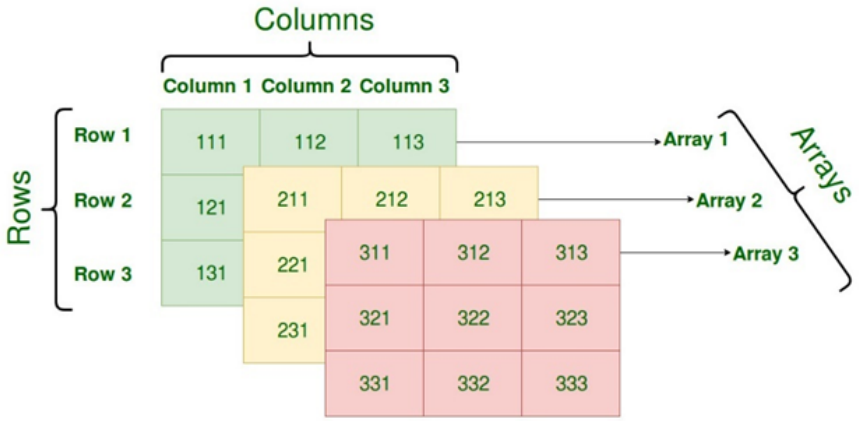
```
#include <iostream>
using namespace std;

int main() {
    int arr[4][4];

    for (int i = 0; i < 4; i++) {
        for (int i = 0; i < 4; j++) {
            arr[i][j] = i + j;
        }
    }

    for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                cout << arr[i][j] << " ";
            }
            cout << endl;
        }
    }

    return 0;
}
```

Pada kode di atas, kita telah mendeklarasikan array 2D dengan 4 baris dan 4 kolom, setelah itu kita menginisialisasi array dengan nilai (i+j) pada setiap iterasi loop. Kemudian kita mencetak array 2D tersebut menggunakan nested loop.

- Array 3 Dimensi

  Array tiga dimensi berisikan kumpulan berbagai susunan dua dimensi yang ditumpuk satu di atas yang lain, sehingga dapat dipresentasikannya sebagai Tiga indeks yaitu indeks baris, indeks kolom, dan indeks tingkat yang digunakan untuk mengidentifikasi setiap elemen dalam susunan tiga dimensi secara unik.

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



Gambar 3. Array 3 Dimensi

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

int main() {
    int arr[3][3][3];

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                arr[i][j][k] = i + j + k;
            }
        }
    }

    for (int i = 0; i < 3; i++) {
        cout << "layer ke" << i << endl;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                cout << arr[i][j][k] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
```

```
    return 0;
}
```

Pada kode di atas, kita telah mendeklarasikan array tiga dimensi, kemudian menginisialisasinya menggunakan tiga *nested-for loops*. Setelah itu, kita mencetak semua elemen pada array tiga dimensi lagi menggunakan tiga *nested-for loops*.

STRUCT

Struct adalah pengelompokan variabel-variabel yang bernaung dalam satu nama yang sama. Berbeda dengan array yang berisi sekumpulan variabel-variabel yang bertipe sama dalam satu nama. Maka suatu struct dapat terdiri atas variabel-variabel yang berbeda tipenya dalam satu nama struct. Struct biasa dipakai untuk mengelompokkan beberapa informasi yang berkaitan menjadi sebuah kesatuan. Variabel-variabel yang membentuk suatu struct, selanjutnya disebut sebagai elemen dari struct atau **field**. Dengan demikian dimungkinkan suatu struct dapat berisi elemen-elemen data berbeda tipe seperti char, int, float, double, dan lain-lain. Dalam program yang sederhana, jika kita menggunakan sedikit variable tentu tidak jadi masalah. Akan tetapi jika kita akan membuat sebuah program yang lebih kompleks, dengan berbagai macam nama dan tipe variabel dalam pendeklarasianya. Dengan struct, kita bisa mengelompokkan berbagai nama dan tipe variabel tersebut sesuai dengan kelompoknya. Hal ini tentunya bisa berguna untuk memudahkan dalam mengelompokkan sebuah variabel. Contohnya adalah sebagai berikut:

```
struct alamat {
    char* nama;               // “John Doe”
    long int nomor_rumah;     // 33
    string nama_jalan;        // “Awang Long”
    srring kota;              // “Samarinda Ulu”
    long kode_pos;            // 75123
};
```

Struct pada tabel tersebut mendefinisikan sebuah tipe baru bernama `alamat`, yang terdiri dari item-item yang dibutuhkan untuk mengirim surat kepada seseorang. Perhatikan pula tanda titik koma di akhir. Lokasi tersebut adalah salah satu dari sedikit tempat di C++ di mana tanda titik koma diperlukan setelah kurung kurawal, sehingga orang sering lupa menambahkannya.

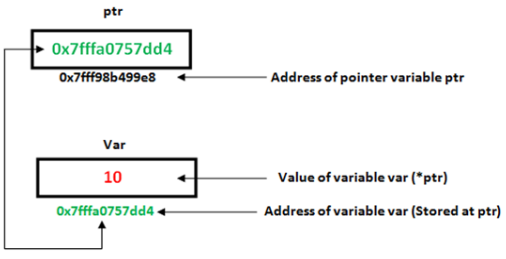
Variabel bertipe `alamat` dapat dideklarasikan seperti variabel lainnya, dan anggota individunya dapat diakses menggunakan operator titik (`.`). Sebagai contoh:

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

struct alamat{
    string nama;
    long int nomor_rumah;
    string nama_jalan;
    string kota;
    long kode_pos;
};

int main() {
    alamat jd;
    jd.nama = "John Doe";
    jd.nomor_rumah = 33;

    cout << jd.nama << endl;
    cout << jd.nomor_rumah << endl;

    return 0;
}
```

Gambar 4. Struct

POINTER

Pointer adalah sebuah variabel atau objek yang menunjuk ke variabel atau objek lainnya, dalam hal ini pointer bisa dikatakan sebagai sebuah alias dari variabel lainnya. Pointer hanyalah sebuah variabel yang menyimpan alamat memori, memori tersebut berasal dari variabel, objek, dan lainnya.



Gambar 5. Pointer

Dengan menggunakan pointer maka sangat memungkinkan bagi kita untuk menunjuk suatu memori, mendapatkan isi memori dan mengubah isi memori yang ditunjuk.

OPERATOR DALAM POINTER

Dalam C++, operator yang bekerja dengan pointer sangat penting untuk memanipulasi alamat memori dan nilai yang disimpan di dalamnya. Ada dua jenis operator utama yang sering digunakan dalam konteks pointer, yaitu Address-Of Operator dan Dereference Operator. Berikut adalah penjelasan lebih rinci tentang masing-masing operator:

- Address-Of Operator
  Address-of Operator (&) adalah operator yang memungkinkan kita untuk mendapatkan/melihat alamat memori yang dimiliki oleh variabel tersebut. Cara menggunakannya adalah dengan meletakan tanda & di depan identitas saat pemanggilan variabel. Hal itu akan membuat compiler memberikan alamat memori bukan isi/nilai dari memori tersebut. Contoh penggunaan Address-Of Operator sebagai berikut.

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

int main() {
    // Mendeklarasikan variabel a dengan nilai 42
    int a = 42;

    // Mendeklarasikan pointer ptr yang dapat menyimpan alamat tipe int
    int *ptr;

    // Menyimpan alamat dari variabel a ke pointer ptr
    ptr = &a;

    // Menampilkan alamat memori dari variabel a
    cout << "Alamat dari variabel a: " << &a << endl;
    cout << "Alamat yang disimpan dalam pointer ptr: " << ptr << endl;

    return 0;
}

```

Gambar 6. Address-Of Operator

- Dereference Operator
  Jika pada address-of operator memungkinkan kita untuk melihat alamat dari variabel yang telah kita buat, maka untuk dereference operator ini berlaku kebalikannya di mana operator ini memungkinkan kita untuk melihat nilai yang tersimpan pada alamat memori. Contohnya adalah sebagai berikut.

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

int main() {
    // Mendeklarasikan variabel a dengan nilai 42
    int a = 42;

    // Menyimpan alamat dari variabel a ke pointer ptr
    int *ptr = &a;

    // Mengakses dan menampilkan nilai dari variabel a melalui pointer ptr
    cout << "Nilai dari variabel a melalui pointer: " << *ptr << endl;

    // Mengubah nilai dari variabel a melalui pointer ptr
    // Menugaskan nilai baru ke variabel a melalui dereference pointer ptr
    *ptr = 100;

    // Menampilkan nilai a setelah diubah
    cout << "Nilai baru dari variabel a: " << a << endl;

    return 0;
}

```

Gambar 7. Dereference Operator

MEMBUAT SEBUAH POINTER

Pada umumnya pointer sebenarnya adalah sebuah variabel, peraturan yang dimiliki variabel juga berlaku pada pointer, jadi tidak jauh beda dengan variabel. pointer hanya mendapatkan beberapa perbedaan yaitu penambahan dua operator yang akan membuat variabel menjadi variabel pointer. Bentuk penulisan sebuah pointer adalah sebagai berikut.

```
Tipe_data *identitas;
// atau
Tipe_data *identitas = &variabel;
```

```
int *pInt;
double *pDouble = &myVar;
```

PENDALAMAN STRING

String adalah rangkaian karakter yang diapit oleh tanda petik dua, seperti pada kalimat “Ini adalah string”. Sebenarnya, sebuah string memiliki satu karakter tambahan yang tidak terlihat, yaitu karakter penutup '\0' dengan nilai 0, yang menandakan akhir dari string tersebut. Contohnya adalah sebagai berikut.

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

int main() {
    bool a = sizeof("Bohr") == 4;
    bool b = sizeof("Bohr") == 5;

    cout << a << endl; // False
    cout << b << endl; // True

    return 0;
}
```

Gambar 8. String

Tipe dari string sendiri adalah "array yang berisikan elemen berupa karakter konstan", jadi "Bohr" merupakan ekuivalen dari tipe "const char[5]."

Sebuah string dapat ditugaskan kepada char*. Hal ini diperbolehkan karena pada mulanya tipe string merupakan sebuah char*. Sehingga kode yang telah dituliskan pada bahasa C atau C++ tetap dapat digunakan. Namun, terdapat error seperti berikut apabila kita ingin mengubah string tersebut melalui sebuah pointer.

```
void f() {
    char* p = "Plato";

    /*
    error: penugasan kepada nilai konstan;
    hasilnya adalah undefined
    */
    p[4] = 'e';
}
```

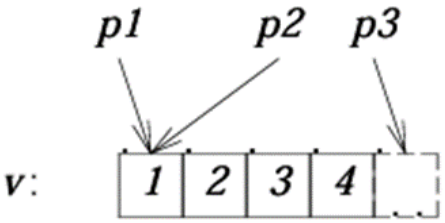
Error seperti ini umumnya tidak dapat terdeteksi saat program sedang dijalankan, dan aturan implementasinya bervariasi. Menggunakan string yang bersifat konstan tidak hanya lebih jelas, tetapi juga memungkinkan pengoptimalan yang signifikan dalam penyimpanan dan akses string tersebut. Jika kita membutuhkan string yang dapat dimodifikasi, kita harus menyalin karakter tersebut ke dalam array.

```
void f() {
   char p[] = "Zeno";
   p[0] = 'R';
}
```

POINTER PADA ARRAY

Dalam C++, pointer dan array saling terkait erat. Nama array dapat digunakan sebagai pointer ke elemen awalnya. Misalnya:

```
int v[] = {1, 2, 3, 4};
int* p1 = v;       // pointer ke elemen paling awal pada array (implisit)
int* p2 = &v[0];   // pointer ke elemen paling awal pada array
int* p3 = &v[4];   // pointer ke satu nilai setelah elemen akhir
```



Gambar 9. Ilustrasi Pointer pada Array

Ketika membuat sebuah array yang bertipe integer, maka proses alokasi pada memori kita akan berurutan dengan jarak 4 byte sebanyak panjangnya sebuah array. Jadi ketika ingin mengakses array menggunakan pointer kita cukup menyimpan alamat dari data pertama saja, kemudian ketika ingin mengakses data selanjutnya cukup dengan melakukan operasi increment.

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

int main() {
    int a[5] = {1, 2, 3, 4, 5};
    int *p = a;  // Pointer p menunjuk ke elemen pertama array a

    // Menggunakan batas akhir dari array, yaitu di luar elemen terakhir
    while (p != &a[5]) {
        cout << "Nilai: " << *p << ", Alamat: " << p << endl;
        p++;  // Maju ke elemen berikutnya
    }

    return 0;
}

```

Gambar 10. Pointer di Array

FUNGSI DENGAN PARAMETER POINTER

Meskipun penggunaan pointer nampaknya kurang bermanfaat dalam contoh kode sebelumnya, karena kita dapat dengan mudah mengubah nilai variabel langsung dengan menetapkan nilai baru ke variabel yang ingin diubah. Hal ini akan berubah jika kita ingin bermain menggunakan fungsi maupun prosedur. Lihat contoh *source code* pada gambar di bawah. Dan *output* dari kode tersebut dapat dilihat pada gambar berikutnya.

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

int ubahNilai(int x, int y) {
    return x = y;
}

int main() {
    int a = 10;
    ubahNilai(a, 20);
    cout << a << endl;
    return 0;
}
```

Gambar 11. Usaha Mengubah Nilai Melalui Fungsi

Source code tersebut gagal mengubah nilai dari variabel "a" yang dapat dilihat melalui gambar berikut.

```
PS D:\Source Code\College\modul-apl> cd "d:\Source Code\College\modul-apl\" ;
if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
10
```

Gambar 12. Output Gambar 11

Kita dapat melihat bahwa nilai variabel tidak berubah meskipun telah dilakukan operasi pengubahan nilai melalui fungsi. Supaya kita berhasil mengubah nilai variabel tersebut melalui fungsi atau prosedur, perlu melakukan modifikasi pada fungsi dengan menambahkan Operator Dereference (`*`) atau Operator Address-of (`&`) pada parameter fungsi. Contoh penerapan dari modifikasi ini dapat dilihat pada gambar berikut.

- Penggunaan Dereference Operator (*) sebagai Parameter Fungsi
  Berikut merupakan implementasi dari dereference operator sebagai parameter dari sebuah fungsi.

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

int ubahNilai(int *x, int y) {
    return *x = y;
}

int main() {
    int a = 10;
    ubahNilai(&a, 20);
    cout << a << endl;
    return 0;
}
```

Gambar 13. Dereference Operator

- Penggunaan Address-Of Operator (&) sebagai Parameter Fungsi
  Berikut merupakan implementasi dari address-of operator sebagai parameter dari sebuah fungsi.

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

int ubahNilai(int &x, int y) {
    return x = y;
}

int main() {
    int a = 10;
    ubahNilai(a, 20);
    cout << a << endl;
    return 0;
}
```

Gambar 14. Address-Of Operator

Fungsi pada kedua source code di atas berhasil mengubah nilai variabel dengan nilai yang diinginkan. Output dari source code di atas dapat dilihat pada gambar di bawah.

```
PS D:\Source Code\College\modul-apl> cd "d:\Source Code\College\modul-apl\" ;
if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
20
```

Gambar 15. Output Gambar 13 dan 14

POINTER PADA STRUCT

Penggunaan pointer pada struct maupun class juga sangat penting untuk diketahui, karena konsep ini akan sangat berguna untuk memahami bagaimana tiap objek saling terkait.

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

struct Mahasiswa {
    string nama;
    int nim;
};

int main() {
    Mahasiswa mhs1;
    Mahasiswa *mhsPtr = &mhs1;

    mhs1.nama = "Yunjin";
    mhs1.nim = 321;
    cout << mhs1.nama << " " << mhs1.nim << endl;

    mhsPtr->nama = "Chaewon";
    mhsPtr->nim = 123;
    cout << mhs1.nama << " " << mhs1.nim << endl;

    return 0;
}
```

Gambar 16. Pointer di Struct

Berdasarkan gambar di atas, dapat dilihat bahwa terdapat perbedaan cara untuk mengakses atribut pada objek struct. Untuk memberi nilai pada atribut objek digunakan Dot Operator (.), namun hal ini dapat berubah apabila kita menggunakan pointer, untuk memberikan nilai pada atribut objek yang ditunjuk oleh pointer, diperlukan operator yang berbeda, yaitu Arrow Operator (->).

OBJEK YANG SALING TERKAIT

Linked List adalah salah satu struktur data fundamental dalam pemrograman yang terdiri dari kumpulan elemen atau objek yang saling terkait. Berbeda dengan array yang menyimpan data secara berurutan dalam memori, Linked List menyimpan elemen secara dinamis dan setiap elemen, atau yang disebut "node", berisi data serta referensi ke node berikutnya dalam rantai. Struktur ini memungkinkan penyimpanan dan pengelolaan data secara efisien, terutama ketika elemen perlu ditambahkan atau dihapus dengan sering.

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

struct Mahasiswa {
    string nama;
    int nim;
    Mahasiswa *next;
};

int main() {
    Mahasiswa mhs1;
    Mahasiswa mhs2;
    Mahasiswa mhs3;
    Mahasiswa *mhsPtr = &mhs2;

    mhs1.nama = "Yunjin";
    mhs1.nim = 321;
    mhs1.next = mhsPtr;

    mhsPtr->nama = "Chaewon";
    mhsPtr->nim = 123;
    mhsPtr->next = &mhs3;

    mhs3.nama = "Sakura";
    mhs3.nim = 213;
    mhs3.next = nullptr;

    Mahasiswa *head = &mhs1;

    while (head != nullptr) {
        cout << head->nama << " " << head->nim << endl;
        head = head->next;
    }

    return 0;
}
```

Gambar 17. Single Linked List