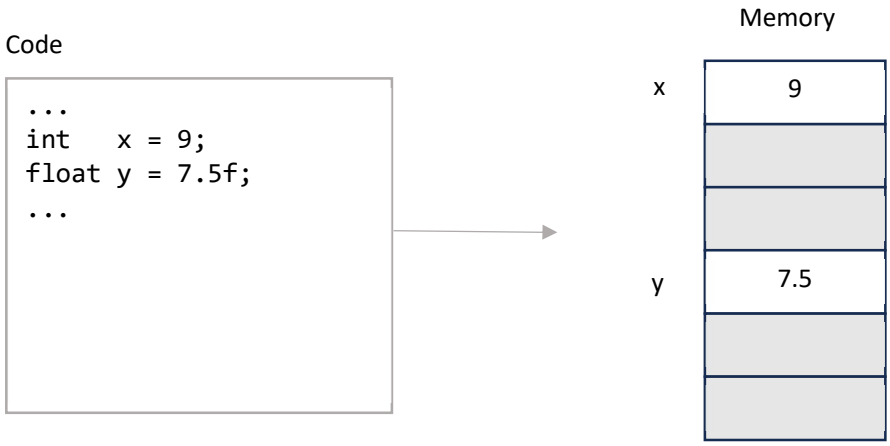
08 Array

Ringkasan Materi

Pengertian

Pada materi terdahulu (02 Literal Data) sudah pernah dibahas bahwa variabel digunakan untuk menyimpan data selama program berjalan. Saat variabel dideklarasikan yang terjadi adalah pengalokasian sebuah 'petak' di memori untuk menyimpan data dengan nama tertentu, yaitu nama variabel. Ketika terjadi pengisian nilai pada variabel tersebut, maka nilai akan tersimpan pada 'petak' yang sudah dialokasikan sebelumnya. Ilustrasinya seperti pada Gambar 1 berikut ini.



Gambar 25 Alokasi variabel dalam memori

Gambar 25 di atas mengilustrasikan alokasi memori ketika ada deklarasi variabel **x** dengan nilai **9** dan variabel **y** dengan nilai **7.5**. Demikian seterusnya ketika ada pendeklarasian variabel baru maka akan terjadi hal yang sama.

Kasus:

Diminta untuk membuat program yang menyimpan 40 data nilai mata kuliah Algoritma dan Pemrograman. Dari data nilai yang tersimpan tersebut nantinya akan digunakan untuk perangkingan, perhitungan rata – rata, dan lain – lain.

Berdasar materi yang sudah pernah dipelajari, maka data nilai – nilai tersebut akan disimpan satu persatu dalam setiap variabel dengan cara mendeklarasikan 40 variabel, misal dengan namai **nilai_1**, **nilai_2**, **nilai_3** dan seterusnya. Untuk menghitung rata – rata misalnya, maka harus dilakukan perhitungan:

```
rerata = (nilai_1 + nilai_2 + nilai_3 + ... + nilai_40) / 40.0;
```

yang tentunya akan sangat merepotkan.

Untuk itulah diperlukan mekanisme penyimpanan yang lebih baik untuk menangani data dalam jumlah yang banyak.

Salah satu struktur data yang dapat digunakan untuk menangani data dengan jumlah yang banyak adalah **array** atau larik.

Cara kerjanya hampir sama dengan pendeklarasian variabel biasa. Bedanya, pada *array* tiap data yang tersimpan pada 'petak' memori yang berbeda – beda akan diorganisir oleh sebuah nama.

Unuk kasus nilai di atas, dapat menggunakan *array* dengan sintaks sebagai berikut:

```
double nilai[40];
```

Nama *array* adalan **nilai** dengan ukuran **40**. Deklarasi di atas akan memerintahkan program untuk memesan sebanyak **40** 'petak' memori dengan nama **nilai** bertipe **double**.

Catatan

> **String** pada dasarnya dalah sebuah *array* bertipe char.String dalam bahasa C berupa *array char* yang diakhiri dengan karakter NULL. Sedangkan dalam bahasa C++ menggunakan mekanisme yang lebih kompleks, di mana di dalamnya terdapat beberapa fungsi untuk manaipulasi.

Operasi

Array bisa dianalogikan sebuah loker yang memiliki beberapa ruang penyimpanan. Tiap ruang penyimpan memiliki nomor urut untuk membedakan satu sama lain. Tiap data yang tersimpan dalam *array* juga akan memiliki nomor urut untuk identifikasi. Gambar 26 berikut ini menunjukkan ilustrasi sebuah *array* dengan nama **myArray** berukuran **6.**

myArray

indeks	0	1	2	3	4	5
elemen						

Gambar 26 Ilustrasi array

Keterangan:

- Elemen: isi dari *array* di mana nilai tersimpan.
- Indeks: nomor urut tiap elemen yang dimulai dari **0**.
  Indeks terakhir dari *array* berukuran **n** adalah **n – 1.**

Pengaksesan *array* (contoh: **myArray**)dapat dilakukan dengan sintaks **myArray[***indeks***]**. Pengaksesan ini bisa dilakukan untuk membaca atau menulis. Contoh penggunaannya adalah sebagai berikut:

```
int myArray[6];
// memasukkan elemen array pada indeks 3
myArray[3] = 9;
// membaca elemen array pada indeks 3
cout << "isi array pada indeks 3 adalah " << myArray[3];
```

Elemen *array* dapat diinsialisasi saat deklarasi, contohnya adalah sebagai berikut:

```
int myArray[6] = { 5, 12, 45, 9, 8, 7 };
```

Pengaksesan juga dapat dilakukan menggunakan perulangan di mana *counter* (langkah) perulangan dapat dimanfaatkan untuk nomor indeks. Contohnya adalah seperti berikut ini:

```
for(int = 0; i < 6; ++i)
{
    cout << “elemen ke-“ << i << “ : “ << myArray[i] << endl;
}
```

Coba potongan kode di atas menggunakan perulangan **while** dan **for.**

Array Multidimensi

Pada dasarnya *array* multidimensi adalah *array* yang di dalamnya berisi *array*. Sebagai contoh adalah ruang kelas di Universitas Amikom Yogyakarta yang memiliki aturan penomoran nomor gedung, nomor lantai, dan nomor ruang; seperti Ruang 5.3.4 yang berarti ruang kelas di gedung 5, lantai 3, ruang 4. Ruang kelas ini dapat dimodelkan menjadi sebuah array berdimensi 3, yaitu:

- Dimensi ke-1: gedung
- Dimensi ke-2: lantai
- Dimensi ke-3: ruang

Yang dapat ditulis dalam kode sebagai berikut:

```
ruangKelas[5][3][4];
```

Contoh lain adalah array 2 dimensi yang biasa digunakan untuk merepresentasikan sebuah tabel yang terdiri dari baris (sebagai dimensi ke-1) dan kolom (sebagai dimensi ke-2). Berikut ini adalah contoh kode pendeklarasian array 2 dimensi yang bertipe integer berukuran 3 x 3:

```
int myArray[3][3] = { { 9, 7, 8 }, { 5, 4, 2 }, {3, 1, 2 } };
// membaca array
for(int i = 0; i < 3; ++i)
{
          for(int j = 0; j < 3; ++j)
          {
                    cout << myArray[i][j] << ‘ ‘;
          }
          cout << endl;
}
```

Batasan

Array dapat menangani data dalam jumlah banyak dengan waktu yang sangat cepat. Namun, ada batasan yaitu berkaitan dengan ukuran karena harus ditentukan saat deklarasi. Sebagai contoh *array* **myArray** di atas berukuran 6 sehingga selama digunakan tidak bisa menyimpan data yang ke-7, 8 atau seterusnya.

Catatan

1. *Array* cocok digunakan untuk data yang jumlahnya sudah dapat dipastikan (tidak ada penambahan).
2. Untuk data yang jumlahnya tidak tentu dapat menggunakan struktur data **vector** di header **<vector>**. Di bahasa pemrograman yang lain ada yang menyebut ArrayList, List, atau yang lain.

Percobaan

Average

Project ini akan menerima input beberapa data dari *user* kemudian akan dilakukan proses perhitungan rata – rata.

```
double nilai[10];
double total = 0.0;

for (int i = 0; i < 10; i++)
{
          cout << "Masukkan nilai ke-" << (i + 1) << ": ";
          cin >> nilai[i];
          total += nilai[i];
}

double rataRata = total / 10.0;
cout << "Rata-rata nilai: " << rataRata << endl;

system("pause");
```

Searching

Project ini mendemokan algoritma pengurutan secara linier, di mana proses dilakukan dengan cara memeriksa satu persatu elemen di *array* dengan nilai yang dicari.

```
int data[10] = { 34, 7, 23, 32, 5, 62, 32, 7, 78, 12 };
int target;
bool ditemukan = false;
cout << "Data dalam array: ";
for (int i = 0; i < 10; i++)
{
          cout << data[i] << " ";
}
cout << endl;
cout << "nilai dicari: ";
cin >> target;
for (int i = 0; i < 10; i++)
{
          if (data[i] == target)
          {
                    ditemukan = true;
                    cout << "Nilai " << target << " ditemukan pada indeks ke-" << i << endl;
                    break;
          }
}
if (!ditemukan)
{
          cout << "Nilai " << target << " tidak ditemukan dalam array." << endl;
}

system("pause");
```

TicTacToe

Project ini berupa demo *array* 2 dimensi dalam bentuk *game Tic tac toe* sederhana. Papan permainan direpresentasikan sebagai array 2 dimensi yang berukuran 3 x 3. Dalam game ini belum ada mekanisme untuk pemeriksaan pemenang. Potongan kode selengkapnya adalah sebagai berikut ini.

```
// papan permainan terdiri dari 3 baris dan 3 kolom
// direpresentasikan dalam array 2D
// diinisialisasi dengan spasi kosong
char board[3][3] = {
          {' ', ' ', ' '},
          {' ', ' ', ' '},
          {' ', ' ', ' '}
};

// simbol pemain: 'X' dan 'O'
// disimpan dalam array satu dimensi
char players[2] = { 'X', 'O' };

int loop = 0;       // untuk melacak giliran pemain
bool playing = true; // status permainan

// loop utama permainan
while (playing)
{
          system("cls");
          cout << " Tic Tac Toe" << endl << endl;
          cout << " |---|---|---|" << endl;
          // menampilkan papan permainan
          for(int rows = 0; rows < 3; rows++)
          {
                    for(int cols = 0; cols < 3; cols++)
                    {
                              cout << " | " << board[rows][cols];
                    }
                    cout << " |" << endl;
                    cout << " |---|---|---|" << endl;
          }

          cout << endl;
          int row, col;
          // menentukan pemain saat ini berdasarkan giliran
          // jika loop genap, pemain 0 (X), jika ganjil, pemain 1 (O)
          int currentPlayer = loop % 2;
          // meminta input baris dan kolom dari pemain saat ini
          cout << " Player " << players[currentPlayer] << ": ";
          cin >> row >> col;

          // memperbarui papan permainan dengan simbol pemain saat ini
          board[row][col] = players[currentPlayer];
          loop++;
}
```

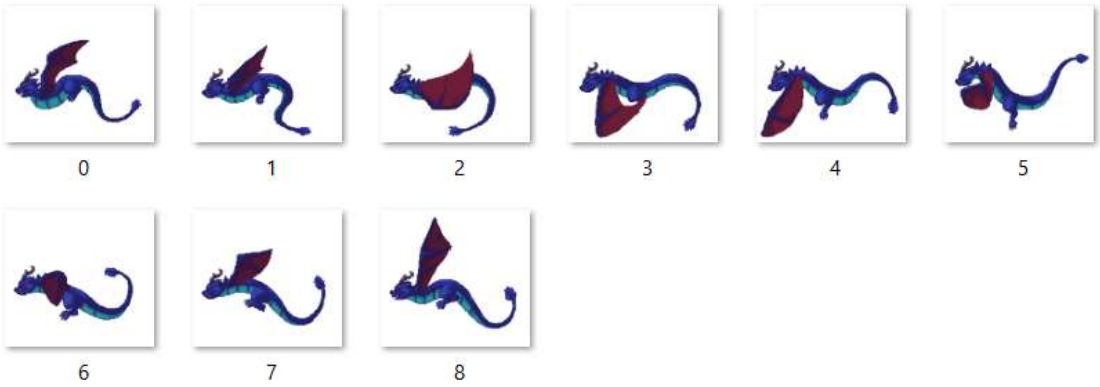
Tambahkan logika untuk penentuan pemenang, boleh menggunakan bantuan AI.

Dragon

Project ini adalah demo animasi *frame-by-frame*. Tiap *frame* berupa sebuah gambar. Animasi berlangsung ketika tiap gambar (*sprite*) ditampilkan secara bergantian. *Project* ini menggunakan *library* eksternal yaitu SIGIL (silakan dilihat kembali materi 07 Pengayaan #2). Aset gambar bisa diunduh di

https://reps.yipsip.my.id/ST068-2025/08/dragon.zip kemudian diekstrak ke lokasi masing – masing. Di modul ini aset berada di **D:\exps\res\sprites\dragon**.

Setelah diekstrak folder **dragon** berisi 9 gambar berformat **.png** dengan nama **0.png**, **1.png**, dan seterusnya sampai **8.png**. Gambar 27 berikut ini adalah isi dari folder **dragon.**



Gambar 27 Kumpulan frame animasi dragon

Penamaan file menggunakan nomor urut 0 – 8 untuk mempermudah saat diakses menggunakan indeks *array*.

Sebelum memulai menulis kode, lakukan pengaturan *library* SIGIL seperti sebelumnya. Berikut ini adalah potongan lengkap kodenya.

```
const char imgPath[] = "D:/exps/res/sprites/dragon/";      // Path gambar, sesuakan sendiri
const int totalFrames = 9;                                 // Total frame animasi
int animationFrames[totalFrames];

// Inisialisasi jendela
slWindow(800, 600, "Animasi Naga Terbang", false);

// Memuat semua frame tekstur naga
for(int i = 0; i < totalFrames; i++)
{
          // fungsi to_string mengubah integer ke string
          // contoh: "D:/exps/res/sprites/dragon/" + to_string(0) + ".png"
          // menjadi "D:/exps/res/sprites/dragon/0.png"
          string fileName = imgPath + to_string(i) + ".png";
          // Memuat tekstur dan menyimpannya dalam array
          animationFrames[i] = slLoadTexture(fileName.c_str());
}

int currentFrame = 0;
double elapsedTime = 0;
while(!slShouldClose())
{
          slSetBackColor(1, 1, 1); // Putih

          // Menggambar frame animasi saat ini di tengah jendela
          slSprite(animationFrames[currentFrame], 400, 300, 192, 176);
          // Memperbarui waktu yang telah berlalu
          elapsedTime += slGetDeltaTime();

          // Jika sudah lebih dari 0.1 detik, ganti ke frame berikutnya
          if(elapsedTime >= 0.1)
          {
                    currentFrame++;
                    elapsedTime = 0;
          }
```

```
        // Jika sudah mencapai frame terakhir, kembali ke frame pertama
        if (currentFrame >= totalFrames)
        {
                currentFrame = 0;
        }

        slRender();
}

slClose();
```

Kode di atas melakukan penggambaran *frame* tiap 0.1 detik. Pengaturan waktu ini menggunakan fungsi **slGetDeltaTime()** yang menghitung selisih waktu antara satu perulangan ke perulangan berikutnya. Selisih waktu tersebut diakumulasikan, bila mencapai 0.1 maka akan berpindah ke *frame* berikutnya.

Latihan

1. Buatlah sebuah program untuk menyimpan data inventaris barang. Data yang disimpan untuk tiap barang adalah **idBarang**, **namaBarang**, dan **jumlahBarang.** Input dilakukan melalui *keyboard*.
   Kriteria:
   - Jumlah data maksimal adalah 50 (ukuran array 50)
   - Ada menu yang berisi:
     - Input: untuk input data barang
     - View: untuk melihat keseluruhan data barang
     - Exit: keluar dari program
   - Gunakan struktur perulangan yang sesuai.
2. Buatlah program untuk menampilkan ucapan “Happy New Year 2026” seperti pada gambar berikut ini:

   Ketentuan:
   - Tulisan 2026 menggunakan *map* berupa *array* string, karakter ‘0’ untuk spasi dan karakter ASCII 178 untuk membuat garisnya.
   - Tulisan “Happy New Year” dibuat animasi, ditampilkan huruf demi huruf dengan delay 250 ms.