Tutorial 3

Praktikum Pemrograman Berbasis Objek

Asisten IF2210 2023/2024

Gimana Praktikum 2 nya?

Gampang? Susah? Pusing nge-*match output* yang linenya banyak? :D

Kesalahan Umum Praktikum 2

- Mengakses method dengan cara menggunakan identifier secara manual. Sebenarnya masih mengeluarkan *output* yang benar, cuman bukan *best practice*
- Kurang teliti dalam penamaan file input

Generic

- Generic adalah sebuah *template* yang bisa digunakan dengan *constraint* yang bisa diubah dalam *compile time* untuk tipe data sesuai yang kita tentukan
- Generic bisa digunakan untuk function dan juga class
- Tujuan dari Generic adalah:
  - DRY (*Don't Repeat Yourself*)
  - Abstraksi

Generic Function

- **Generic Function** adalah fungsi yang **dapat dibuat dari template**, sehingga dapat membuat suatu **algoritma generik** yang dapat bekerja untuk **tipe data / nilai apapun sesuai konstrain**.

Contoh Fungsi: maxElmt (Normal)

```
int maxElmt(int* arr, int N)
// mengembalikan elemen terbesar pada array
// Array arr memiliki elemen sebanyak N
// Diasumsikan N > 0
{
    int max_result = arr[0];
    for (int i = 1; i < N; i++) {
        if (arr[i] > max_result) {
            max_result = arr[i];
        }
    }
    return max_result;
}
```

Contoh Fungsi: maxElmt (Generic)

```cpp
template<class T>
T maxElmt(T* arr, int N)
// mengembalikan elemen terbesar pada array
// Array arr memiliki elemen sebanyak N
// Diasumsikan N > 0
{
    T max_result = arr[0];
    for (int i = 1; i < N; i++) {
        if (arr[i] > max) {
            max_result = arr[i];
        }
    }
    return max_result;
}
```

```cpp
int main() {
    int intArray[] = {1, 3, 5, 7, 9};
    int maxInt = maxElmt(intArray, 5);

    double doubleArray[] = {1.1, 2.2, 3.3, 4.4, 5.5};
    double maxDouble = maxElmt(doubleArray, 5);

    std::cout << "Maximum integer: " << maxInt << std::endl;
    std::cout << "Maximum double: " << maxDouble << std::endl;

    return 0;
}
```

```
Maximum integer: 9
Maximum double: 5.5
```

Generic Class

- **Generic Class** adalah kelas yang dapat didefinisikan dari **sebuah template**, sehingga **tidak perlu** membuat ulang implementasi kelas yang sama untuk **tipe data / nilai yang berbeda**.

Generic Class Example: Vector2

- Vector ini adalah representasi dari vektor yang sudah dipelajari di fisika maupun matematika.
- Catatan: bedakan dengan STL vector milik C++ yang pada dasarnya merupakan array dinamis

Vector2 (Normal)

```cpp
class Vector2 {
private:
    int* elements;

public:
    Vector2() {
        this->elements = new int[2];
        this->elements[0] = 0;
        this->elements[1] = 0;
    }

    Vector2(const Vector2& other) {
        this->elements = new int[2];
        this->elements[0] = other.elements[0];
        this->elements[1] = other.elements[1];
    }

    ~Vector2() {
        delete[] this->elements;
    }
```

```cpp
int& operator[](int idx) {
    return this->elements[idx];      Yonas Adiel, 2 years
}

Vector2 operator+(const Vector2& other) {
    Vector2 result;
    result.elements[0] = elements[0] + other.elements[0];
    result.elements[1] = elements[1] + other.elements[1];
    return result;
}

Vector2 operator-(const Vector2& other) {
    Vector2 result;
    result.elements[0] = elements[0] - other.elements[0];
    result.elements[1] = elements[1] - other.elements[1];
    return result;
}
```

Vector2 (Normal)

```
bool operator<(const Vector2& other) {
    if (elements[0] != other.elements[0]) {
        return elements[0] < other.elements[0];
    }
    return elements[1] < other.elements[1];
}

bool operator>(const Vector2& other) {
    if (elements[0] != other.elements[0]) {
        return elements[0] > other.elements[0];
    }
    return elements[1] > other.elements[1];
}
```

```
friend std::ostream& operator<<(ostream& os, Vector2 vector) {
    os << "<";
    os << vector.elements[0];
    os << ",";
    os << vector.elements[1];
    os << ">";
    return os;
}

friend std::istream& operator>>(istream& is, Vector2& vector) {
    return is >> vector.elements[0] >> vector.elements[1];
}
```

Generic Class Example: Vector2

- Terdapat 2 kekurangan dari vector ini:
  - Elemen vector harus berupa integer
  - Vector hanya memiliki panjang 2
- Kabar baiknya, dua kekurangan ini dapat diselesaikan dengan membuat generic class dari Vector!

Vector2 (Generic)

```cpp
class Vector2 {
private:
    int* elements;

public:
    Vector2() {
        this->elements = new int[2];
        this->elements[0] = 0;
        this->elements[1] = 0;
    }

    Vector2(const Vector2& other) {
        this->elements = new int[2];
        this->elements[0] = other.elements[0];
        this->elements[1] = other.elements[1];
    }

    ~Vector2() {
        delete[] this->elements;
    }
```

Sebelum

```cpp
template<class T, int N>
class Vector {
private:
    T* elements;

public:
    Vector() {
        this->elements = new T[N];
    }

    Vector(const Vector<T, N>& other) {
        this->elements = new T[N];
        for (int i = 0; i < N; i++) {
            this->elements[i] = other.elements[i];
        }
    }

    ~Vector() {
        delete[] this->elements;
    }
```

Sesudah

Vector2 (Generic)

```
int& operator[](int idx) {
    return this->elements[idx];        Yonas Adiel, 2 years
}

Vector2 operator+(const Vector2& other) {
    Vector2 result;
    result.elements[0] = elements[0] + other.elements[0];
    result.elements[1] = elements[1] + other.elements[1];
    return result;
}

Vector2 operator-(const Vector2& other) {
    Vector2 result;
    result.elements[0] = elements[0] - other.elements[0];
    result.elements[1] = elements[1] - other.elements[1];
    return result;
}
```

Sebelum

```
T& operator[](int idx) {
    return this->elements[idx];
}

Vector<T, N> operator+(const Vector<T, N>& other) {
    Vector<T, N> result;
    for (int i = 0; i < N; i++) {
        result.elements[i] = elements[i] + other.element
    }
    return result;
}

Vector<T, N> operator-(const Vector<T, N>& other) {
    Vector<T, N> result;
    for (int i = 0; i < N; i++) {
        result.elements[i] = elements[i] - other.element
    }
    return result;
}
```

Sesudah

Vector2 (Generic)

```cpp
bool operator<(const Vector2& other) {
    if (elements[0] != other.elements[0]) {
        return elements[0] < other.elements[0];
    }
    return elements[1] < other.elements[1];
}

bool operator>(const Vector2& other) {
    if (elements[0] != other.elements[0]) {
        return elements[0] > other.elements[0];
    }
    return elements[1] > other.elements[1];
}
```

Sebelum

```cpp
bool operator<(const Vector<T, N>& other) {
    for (int i = 0; i < N; i++) {
        if (this->elements[i] != other.elements[i]) {
            return this->elements[i] < other.elements[i];
        }
    }
    return false; // vector sama
}

bool operator>(const Vector<T, N>& other) {
    for (int i = 0; i < N; i++) {
        if (this->elements[i] != other.elements[i]) {
            return this->elements[i] > other.elements[i];
        }
    }
    return false; // vector sama
}
```

Sesudah

Vector2 (Generic)

```
friend std::ostream& operator<<(ostream& os, Vector2 vector) {
    os << "<";
    os << vector.elements[0];
    os << ",";
    os << vector.elements[1];
    os << ">";
    return os;
}

friend std::istream& operator>>(istream& is, Vector2& vector) {
    return is >> vector.elements[0] >> vector.elements[1];
}
```

Sebelum

```
friend ostream& operator<<(ostream& os, const Vector<T, N>& vector) {
    os << "<";
    for (int i = 0; i < N; i++) {
        os << vector.elements[i];
        if (i != N - 1) {
            os << ",";
        }
    }
    os << ">";
    return os;
}

friend istream& operator>>(istream& is, Vector<T, N>& vector) {
    for (int i = 0; i < N; i++) {
        is >> vector.elements[i];
    }
    return is;
}
```

Sesudah

Vector2 (Generic)

```cpp
#include "Vector.hpp"
#include <iostream>

int main() {
    Vector<int, 4> v1, v2;
    cout << "Masukkkan vektor 4 elemen: ";
    cin >> v1;

    v2[0] = -1;
    v2[1] = -2;
    v2[2] = -3;
    v2[3] = -4;
    cout << v1 << " + " << v2 << " = " << v1 + v2 << endl;
    cout << v1 << " - " << v2 << " = " << v1 - v2 << endl;
}
```

```
Masukkkan vektor 4 elemen: 9 5 2 3
<9,5,2,3> + <-1,-2,-3,-4> = <8,3,-1,-1>
<9,5,2,3> - <-1,-2,-3,-4> = <10,7,5,7>
```

Exception

- Exception: Sebuah *event* yang mendisrupsi *flow* program normal
- Melambangkan behavior yang tidak diharapkan
- Dengan adanya exception, kita dapat menangani behavior yang tidak diharapkan tersebut sesuai kehendak kita.

Contoh: Vector (Dynamic Array)

- Kita membuat Vector sebagai *utility class*
- Jika ada error dalam pengunaan Vector, seperti apa penanganannya?
  - Print pesan? Langsung exit program? Recovery?
- Jawabannya: Tergantung program
  - Vector cukup “melempar” sinyal error berupa Exception
  - Sinyal itu ditangkap oleh program. Lalu program menangani sesuai kebutuhannya.

Exception: Sintaks

- Pada kode yang melempar:

```
throw <suatu objek>;
```

- Pada kode yang menangani:

```
try {
    // kode yang berpotensi melempar exception
} catch (<suatu objek> e) {
    // penanganan
}
```

Contoh: Index Out-of-Bound

Bagaimana jika pada akses indeks di Vector, indeksnya *out of bound*?

Contoh: Throw

```
T& operator[](int idx) {
    if (idx < 0 || N <= idx) {
        throw "Invalid index";
    }
    return this->elements[idx];
}
```

```
int main() {
    Vector<int, 4> v;

    v[5] = 7;

    cout << "Baris ini tidak dieksekusi" << endl;

    return 0;
}
```

```
terminate called after throwing an instance of 'char const*'
Aborted (core dumped)
```

Exception berupa constant array of char.

Exception menyebabkan program berjalan tidak sempurna: langsung exit dengan kode bukan 0.

Contoh: Try-Catch

```
T& operator[](int idx) {
    if (idx < 0 || N <= idx) {
        throw "Invalid index";
    }
    return this->elements[idx];
}
```

```
Error: Invalid index
Baris ini dieksekusi
```

```
int main() {
    Vector<int, 4> v;

    try {
        v[5] = 7;
        cout << "Baris ini tidak dieksekusi" << endl;
    } catch (const char* err) {
        cout << "Error: " << err << endl;
    }

    cout << "Baris ini dieksekusi" << endl;

    return 0;
}
```

Exception yang dilempar (*throw*), ditangkap (*catch*) oleh program utama.

Program juga berhenti sempurna (*exit code*: 0)

Contoh: Class Instance

Exception yang di-*throw* dapat berupa *class instance*...

```
class VectorIndexOutOfBoundException {
private:
    int idxAccessed;
    int numOfElements;
public:
    VectorIndexOutOfBoundException(int idxAccessed, int numOfElements) {
        this->idxAccessed = idxAccessed;
        this->numOfElements = numOfElements;
    }
    void printMessage() {
        cout << "Error: you are trying to access index " << idxAccessed;
        cout << " but the vector only have " << numOfElements;
        cout << " elements." << endl;
    }
};
```

```
T& operator[](int idx) {
    if (idx < 0 || N <= idx) {
        VectorIndexOutOfBoundException e(idx, N);
        throw e;
    }
    return this->elements[idx];
}
```

Contoh: Class Instance

...maka yang di-*catch* juga harus bertipe *class* tersebut.

```
int main() {
    Vector<int, 4> v;

    try {
        v[5] = 7;
        cout << "Baris ini tidak dieksekusi" << endl;
    } catch (VectorIndexOutOfBoundException err) {
        err.printMessage();
    }

    cout << "Baris ini dieksekusi" << endl;

    return 0;
}
```

```
Error: you are trying to access index 5 but the vector only have 4 elements.
Baris ini dieksekusi
```

Exception: Tipe Data

- Kita harus menuliskan tipe data exception yang akan ditangkap
- Artinya, kita juga perlu menangkap banyak exception jika yang dilempar memiliki tipe berbeda-beda

...or should we?

```
try {
    // doing something dangerous
} catch (Exception1 e) {
    // do something
} catch (Exception2 e) {
    // do something
} catch (Exception3 e) {
    // do something
}
```

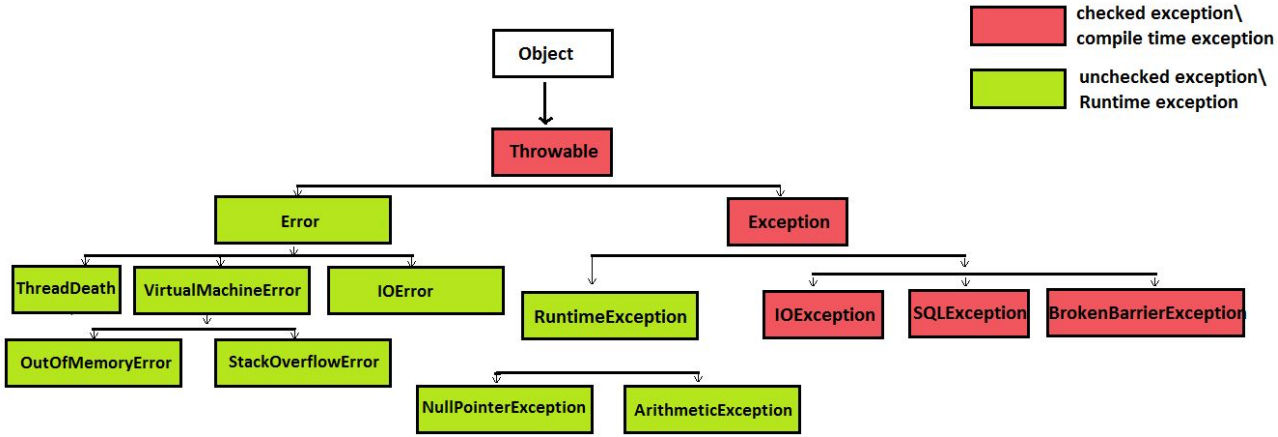
Exception: Polymorphism

Alternatif: Polymorphism

- Idenya, kita dapat membuat sebuah *parent class* **Exception** yang memiliki banyak ***child class***, sesuai kebutuhan.
- Pada *parent class*, kita membuat member seperti **printMessage**, **getMessage**, dll. sesuai yang dibutuhkan oleh kode yang menangani exception.

Exception: Polymorphism

- Contoh kode tidak diberikan, dapat dicoba sendiri di rumah
- Hint: Lihat implementasi oleh C++ itu sendiri



STL

- C++ menyediakan sejumlah fungsi dan modul dasar bawaan yang disebut **standard template library**
- Di antaranya: Algorithm (sort, search), Container (list, vector), dan Iterator.
- Penggunaan cukup dengan include header
  Misal: `#include <vector>` atau `#include <algorithm>`

Contoh Container

- vector: array dinamis
- stack
- queue
- deque: stack sekaligus queue
- list: linked list
- priority_queue
- set
- map
- pair / tuple

Contoh: Vector, Map, Queue

```
int main() {
    vector<int> v;  // seperti array, tapi ukuran dinamis
    v.push_back(4); // v = 4
    v.push_back(2); // v = 4 2
    v.pop_back();   // v = 4

    map<string, int> m;
    m["abc"] = 1;
    m["def"] = 2;
    cout << m["abc"] << endl; // writes 1

    queue<int> q;
    q.push(4); // q = 4
    q.push(2); // q = 4 2
    q.pop();   // q = 2, returns 4

    return 0;
}
```

Contoh Algorithm

```
int main() {
    ...

    sort(a, a + n); // mengurutkan array a berukuran n
    sort(v.begin(), v.end()); // mengurutkan vector v

    find(v.begin(), v.end(), 3); // menemukan nilai 3 di vector v

    binary_search(v.begin(), v.end(), 3); // memeriksa keberadaan nilai 3 di vector terurut

    return 0;
}
```

STL Lainnya

https://en.cppreference.com/w/

C++ reference

C++11, C++14, C++17, C++20, C++23, C++26 | Compiler support C++11, C++14, C++17, C++20, C++23, C++26

Language
- Keywords — Preprocessor
- ASCII chart
- Basic concepts
  - Comments
  - Names (lookup)
  - Types (fundamental types)
  - The main function
- Expressions
  - Value categories
  - Evaluation order
  - Operators (precedence)
  - Conversions — Literals
- Statements
  - if — switch
  - for — range-for (C++11)
  - while — do-while
- Declarations — Initialization
- Functions — Overloading
- Classes (unions)
- Templates — Exceptions
- Freestanding implementations

Standard library (headers)

Named requirements

Feature test macros (C++20)

Language support library
- Program utilities
- source_location (C++20)
- Coroutine support (C++20)
- Three-way comparison (C++20)
- Type support
- numeric_limits — type_info
- initializer_list (C++11)

Concepts library (C++20)

Diagnostics library
- exception — System error
- basic_stacktrace (C++23)

Memory management library
- unique_ptr (C++11)
- shared_ptr (C++11)
- weak_ptr (C++11)
- Memory resources (C++17)
- Allocators — Low level management

Metaprogramming library (C++11)
- Type traits — ratio
- integer_sequence (C++14)

General utilities library
- Function objects — hash (C++11)
- Swap — Type operations (C++11)
- Integer comparison (C++20)
- pair — tuple (C++11)
- optional (C++17)
- expected (C++23)
- variant (C++17) — any (C++17)
- String conversions (C++17)
- Formatting (C++20)
- bitset — Bit manipulation (C++20)
- Debugging support (C++26)

Strings library
- basic_string — char_traits
- basic_string_view (C++17)
- Null-terminated strings:
  - byte — multibyte — wide

Containers library
- vector — deque — array (C++11)
- list — forward_list (C++11)
- map — multimap — set — multiset
- unordered_map (C++11)
- unordered_multimap (C++11)
- unordered_set (C++11)
- unordered_multiset (C++11)
- Container adaptors
- span (C++20) — mdspan (C++23)

Iterators library

Ranges library (C++20)

Algorithms library
- Execution policies (C++17)
- Constrained algorithms (C++20)

Numerics library
- Common math functions
- Mathematical special functions (C++17)
- Mathematical constants (C++20)
- Basic linear algebra algorithms (C++26)
- Numeric algorithms
- Pseudo-random number generation
- Floating-point environment (C++11)
- complex — valarray

Date and time library
- Calendar (C++20) — Time zone (C++20)

Localization library
- locale — Character classification
- text_encoding (C++26)

Input/output library
- Print functions (C++23)
- Stream-based I/O — I/O manipulators
- basic_istream — basic_ostream
- Synchronized output (C++20)
- File systems (C++17)

Regular expressions library (C++11)
- basic_regex — Algorithms
- Default regular expression grammar

Concurrency support library (C++11)
- thread — jthread (C++20)
- atomic — atomic_flag
- atomic_ref (C++20) — memory_order
- Mutual exclusion — Semaphores (C++20)
- Condition variables — Futures
- latch (C++20) — barrier (C++20)
- Safe Reclamation (C++26)

Sekian.

Ditunggu praktikum dan tutorial selanjutnya.