MINGGU 5 & 6

Praktikum Pemrograman Berbasis Objek
oleh Asisten IF2210 2019/2020

OUTLINE

1. Generic Function
2. Generic Class
3. Exception
4. C++ Standard Template Library

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**.

CONTOH MAX_ELMT

```
int max_elmt(int* arr, int N)
// Mengembalikan element terbesar yang ada di array arr.
// Array arr memiliki elemen sebanyak N.
// Diasumsikan N > 0.
{
    int max_result = arr[0];
    for (int i = 0; i < N; i++) {
        if (max_result < arr[i]) {
            max_result = arr[i];
        }
    }
    return max_result;
}
```

Pada fungsi ini, max_elmt hanya mampu memroses array integer.

Bagaimana jika ingin menerima float, string, atau bahkan kelas lain yang sudah didefinisikan?

CONTOH MAX_ELMT GENERIC

```
template<class T>
T max_elmt(T* arr, int N)
// Mengembalikan element terbesar yang ada di array arr.
// Array arr memiliki elemen sebanyak N.
// Diasumsikan N > 0.
{
    T max_result = arr[0];
    for (int i = 0; i < N; i++) {
        if (max_result < arr[i]) {
            max_result = arr[i];
        }
    }
    return max_result;
}
```

GENERIC

- Tidak perlu mendefinisikan hal yang sama berkali-kali
- Keep your code DRY

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 VECTOR

- Sebagai contoh, silakan akses file Vector2.hpp yang disediakan di olympia.
- 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

DEFINISI VECTOR2

```
class Vector2 {
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;
    }

...
```

```
...

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

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;
}

...
```

DEFINISI VECTOR2

```
...

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

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

Ini kode operator overloading untuk membaca / menuliskan objek dari input / output stream (layar, file, dan lainnya).

CONTOH PROGRAM UTAMA

```
int main() {
    Vector2 v1, v2;

    cout << "Masukkan vektor 2 elemen: ";
    cin >> v1;

    v2[0] = -1;
    v2[1] = -2;

    cout << v1 << " + " << v2 << " = " << v1 + v2 << endl;
    cout << v1 << " - " << v2 << " = " << v1 - v2 << endl;

    return 0;
}
```

GENERIC CLASS VECTOR

- 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!

DEFINISI VECTOR GENERIC

```
class Vector2 {
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

```
template<class T, int N>
class Vector {
public:
    Vector() {
        this->elements = new T[N];
        for (int i = 0; i < N; i++) {
            this->elements[i] = 0;
        }
    }

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

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


...
```

Sesudah

DEFINISI VECTOR GENERIC

```
...

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

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 elements[idx];
}

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

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

...
```

Sesudah

DEFINISI VECTOR GENERIC

```
...

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

```
...

bool operator<(const Vector& 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& 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

DEFINISI VECTOR GENERIC

```
...


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



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

Sebelum

```
...


friend ostream& operator<<(ostream& os, 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

CONTOH PROGRAM UTAMA

```
int main() {
    Vector<int, 4> v1, v2;

    cout << "Masukkan 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;

    return 0;
}
```

```
Masukkan vektor 4 elemen: 9 5 3 2
<9,5,3,2> + <-1,-2,-3,-4> = <8,3,0,-2>
<9,5,3,2> - <-1,-2,-3,-4> = <10,7,6,6>
```

EXCEPTION

- Dalam C++, Exception melambangkan behavior yang tidak diharapkan.
- Dengan adanya exception, kita dapat menangani behaviour yang tidak diharapkan tersebut sesuai kehendak kita.

EXCEPTION

- Sebagai contoh, bagaimana jika pada akses indeks di Vector, indeksnya *out of bound*?
- Exception dilakukan dengan menuliskan **throw <suatu objek>;**

EXCEPTION

```
class Vector2 {
public:
    ...

    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)
```

Di sini, kita melempar exception berupa constant array of char.

Exception menyebabkan program berjalan tidak sempurna dan exit dengan kode bukan 0

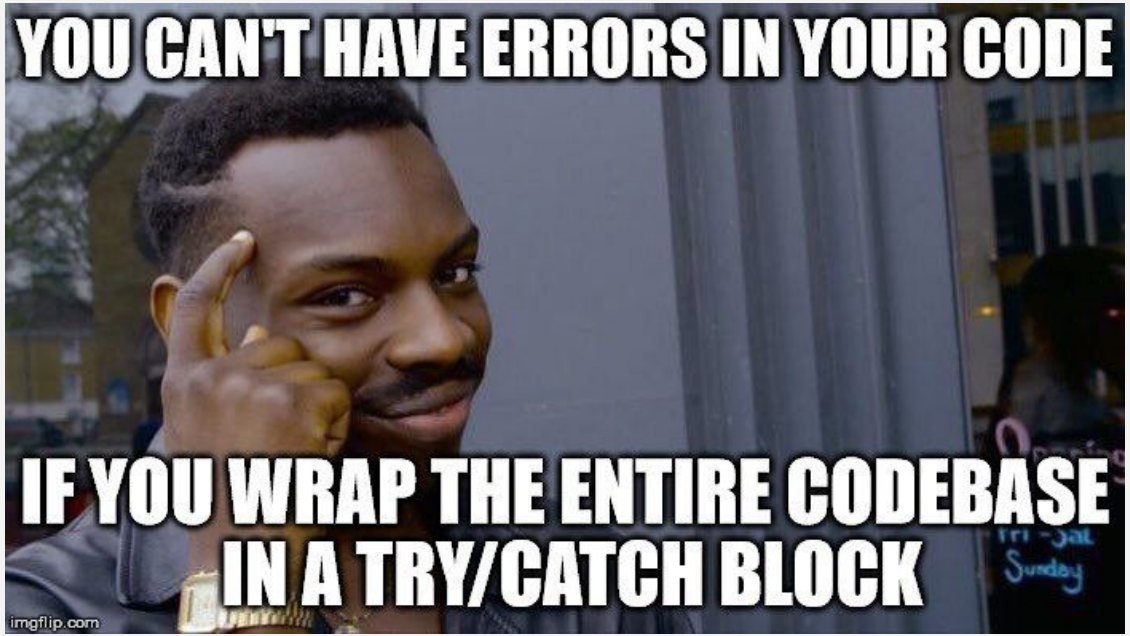
EXCEPTION CATCHED

```
class Vector2 {
public:
    ...

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

    ...
```

```
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;
}
```

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

Di sini, error yang dilempar (throw), ditangkap (catch) oleh program utama. Program juga berhenti sempurna



Selamat! Akhirnya kamu paham meme ini

OBJECT AS EXCEPTION

```
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;
    }
};
```

Atau, kamu juga bisa menerima exception berupa object

OBJECT AS EXCEPTION

```
class Vector2 {
public:
    ...

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


    ...
```

```
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

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

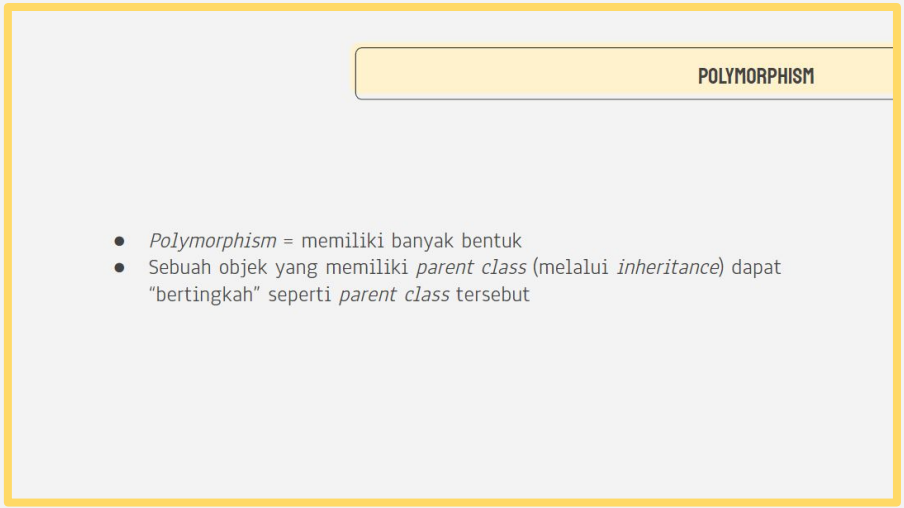
```
...
    try {
        // doing something dangerous
    } catch (Exception1 e) {
        // do something
    } catch (Exception2 e) {
        // do something
    } catch (Exception3 e) {
        // do something
    }
...
```

EXCEPTION

- Atau...
- Bisa juga kita menerima banyak bentuk dari exception

EXCEPTION

- Atau...
- Bisa juga kita menerima **banyak bentuk** dari exception
- Ingat slide ini?



EXCEPTION

- Idenya, kita dapat membuat sebuah kelas `Exception` yang memiliki banyak kelas anak.
- Kita dapat membuat member seperti `printMessage`, `getMessage`, atau lainnya yang dapat dipanggil oleh kode yang melakukan `catch` pada exception
- Contoh kode tidak diberikan, dapat dicoba sendiri di rumah

STL

- C++ menyediakan banyak standard template library, yakni Algoritma (sort, search), Container (list, vector), Iterator, dan fungsi-fungsi lainnya.
- Untuk menggunakannya, perlu meng-*include* header, misal `#include <vector>` atau `#include <algorithm>`

CONTOH CONTAINER

```cpp
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;
}
```

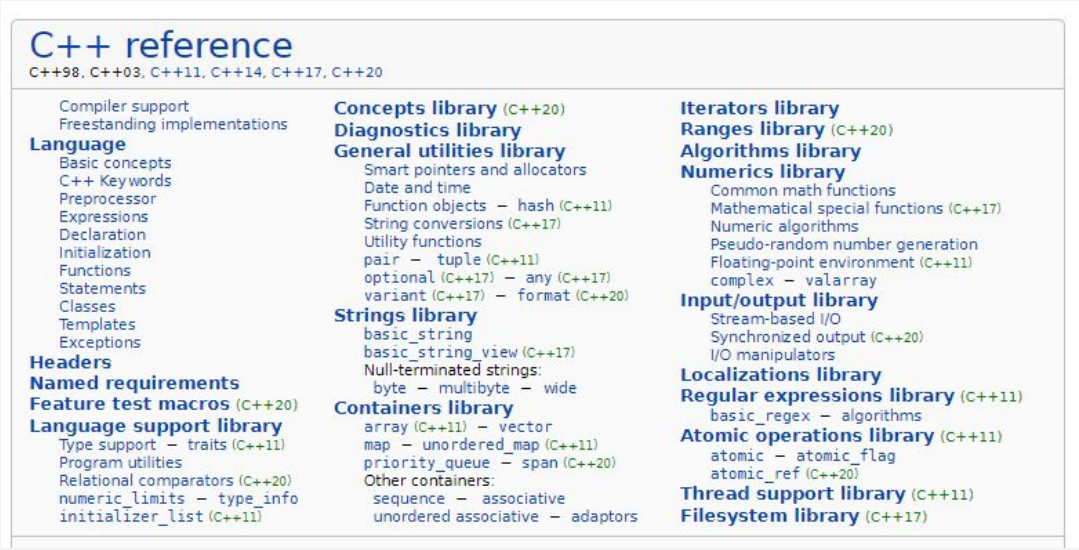
Contoh container:
- vector: array dinamis
- stack
- queue
- deque: stack sekaligus queue
- list: linked list
- priority_queue
- set
- map

CONTOH ALGORITMA

```
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 v

    return 0;
}
```

STL

- Ada banyak, jadi pelajari ya :)



C++ reference

C++98, C++03, C++11, C++14, C++17, C++20

Compiler support
Freestanding implementations

Language
Basic concepts
C++ Keywords
Preprocessor
Expressions
Declaration
Initialization
Functions
Statements
Classes
Templates
Exceptions

Headers
Named requirements
Feature test macros (C++20)
Language support library
Type support – traits (C++11)
Program utilities
Relational comparators (C++20)
numeric_limits – type_info
initializer_list (C++11)

Concepts library (C++20)
Diagnostics library
General utilities library
Smart pointers and allocators
Date and time
Function objects – hash (C++11)
String conversions (C++17)
Utility functions
pair – tuple (C++11)
optional (C++17) – any (C++17)
variant (C++17) – format (C++20)

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

Containers library
array (C++11) – vector
map – unordered_map (C++11)
priority_queue – span (C++20)
Other containers:
sequence – associative
unordered associative – adaptors

Iterators library
Ranges library (C++20)
Algorithms library
Numerics library
Common math functions
Mathematical special functions (C++17)
Numeric algorithms
Pseudo-random number generation
Floating-point environment (C++11)
complex – valarray

Input/output library
Stream-based I/O
Synchronized output (C++20)
I/O manipulators

Localizations library
Regular expressions library (C++11)
basic_regex – algorithms

Atomic operations library (C++11)
atomic – atomic_flag
atomic_ref (C++20)

Thread support library (C++11)
Filesystem library (C++17)