



Bahasa C++: Exception Handling

IF2210 – Semester II 2022/2023

Tujuan

- › Di akhir sesi, peserta diharapkan mampu untuk:
 - › Mendefinisikan *exception*
 - › Menangani *exception* dengan menggunakan blok `try-catch`
 - › Membuat kelas yang dilengkapi dengan class `Exception` sendiri

Penanganan Kesalahan pada Program

- › Penambahan kode khusus untuk menangani kesalahan dalam eksekusi program membuat program menjadi rumit

```
int LakukanAksi() {  
    // ...  
    if ( ... ) // periksa kondisi kesalahan  
        return NILAI_KHUSUS;  
    else  
        return NILAI_FUNGSI;  
}  
  
-----  
if (LakukanAksi() == NILAI_KHUSUS) {  
    // ... jalankan instruksi untuk menangani kesalahan  
} else {  
    // instruksi-1  
    // instruksi-2  
    // ...  
    // instruksi-n  
}
```

Exception

- › Kesalahan pada eksekusi program = *exception*
- › Penanganan *exception* pada C++: **throw**, **catch**, and **try**
- › **return** = kembali dari fungsi secara normal;
throw = kembali fungsi secara abnormal (*exception*)

```
int LakukanAksi() {  
    // ...  
    if ( ... ) // periksa kondisi kesalahan  
        throw "Ada kesalahan";  
    else  
        return NILAI_FUNGSI;  
}
```

Exception

- › *Error* tidak harus ditangani dengan *exception handling* (namun *exception* mempermudah penanganan *error*)
- › *Exception* bekerja dengan cara mengubah alur eksekusi program sambil melempar suatu objek tertentu sebagai informasi untuk alur yang baru
- › Sebuah *event* yang akan menginterupsi alur proses program normal dari sebuah program

Membuat Exception

- › Objek yang diciptakan disebut *exception object*
- › *Exception object* mengandung informasi tentang *error* tersebut (termasuk tipe dan state program ketika *error* terjadi)
- › Menciptakan *exception object* dan melempar ke *runtime system* disebut *throwing an exception* (keyword `throw` di C++)
- › Setelah *method* melempar *exception*, *runtime system* akan mencari sesuatu untuk *meng-handle* itu = disebut *exception handler*
- › *Exception handler* akan melakukan *catch the exception*
 - › Jika tidak di-handle (di-catch) akan mengakibatkan program *terminate abnormally*

Exception Handling

- › **try block** (keyword `try` di C++)
 - › Berisi kode yang mungkin memunculkan *exception*
 - › *Try block* bisa dibuat untuk setiap kode yang mungkin menimbulkan *exception*
 - › Bisa juga dengan mengumpulkan banyak kode dalam sebuah *try block*
- › **catch block** (keyword `catch` di C++)
 - › Berisi kode yang merupakan *exception handler*
 - › Menangani *exception* dengan tipe yang sesuai dengan tipe yang ditunjukkan pada argumen
- › Jika sebuah *method* ditulis menangani *exception*, sebaiknya dilakukan invokasi dalam sebuah blok **try ... catch**

Exception Handling

- › **catch** dituliskan setelah sebuah blok **try** untuk menangkap exception yang di-throw

```
try {  
    // instruksi-1  
    // instruksi-2  
    LakukanAksi();  
    // instruksi-4  
    // ...  
    // instruksi-n  
}  
catch (const char*) {  
    // ... jalankan instruksi untuk menangani kesalahan  
}  
  
// eksekusi berlanjut pada bagian ini...
```


Exception Handling

- › Perintah dalam blok **catch** = *exception handler*
- › Sebuah blok try dapat memiliki > 1 *exception handler*, masing-masing untuk menangani tipe *exception* yang berbeda
- › Tipe *handler* ditentukan “*signature*”-nya. catch-all handler memiliki “signature”

```
try {  
    // instruksi  
}  
catch (StackExp&) {  
    // handler untuk tipe StackExp  
}  
catch (...) { // literally pakai tiga tanda titik  
    // handler untuk semua jenis exception lainnya  
}
```

Pemilihan Handler

- › Jika terjadi *exception*, hanya satu *handler* yang dipilih berdasarkan “*signature*”-nya
- › Instruksi di dalam *handler* yang terpilih diajarkan
- › Eksekusi dari blok **try** tidak dilanjutkan
- › Eksekusi berlanjut ke bagian yang dituliskan setelah bagian **try-catch** tersebut.

Contoh Kelas untuk Stack Exception

```
const int STACK_EMPTY = 0;
const int STACK_FULL = 1;

class StackExp {
public:
    // ctor, cctor, dtor, operator=
    // services
    void DisplayMsg() const;
    static int NumException();
private:
    // static member, shared by all objects of this case
    static int num_ex; // pencacah jumlah exception
    static char* msg[]; // tabel pesan kesalahan
    const int msg_id; // nomor kesalahan
}
```

Fungsi Anggota StackExp

```
#include <iostream>
using namespace std;
#include "StackExp.h"

int StackExp::num_ex = 0;
char* StackExp::msg[] = { "Stack is empty!", "Stack is full!" };

StackExp::StackExp (int x) : msg_id(x) {
    num_ex++; // increase the exception counter
}

StackExp::StackExp (const StackExp& s) : msg_id (s.msg_id) {}

void StackExp::DisplayMsg() const {
    cerr << msg[msg_id] << end;
} console error

int StackExp::NumException() {
    return num_ex;
}
```

Modifikasi Kelas Stack

```
// File: Stack.cpp
// Deskripsi: Kelas Stack dengan exception handling

#include "StackExp.h"

void Stack::Push(int x) {
    if (isFull()) throw (StackExp (STACK_FULL)); // Raise exception
    else {
        // algoritma Push
    }
}

void Stack::Pop(int& x) {
    if (isEmpty()) throw (StackExp (STACK_EMPTY)); // Raise exception
    else {
        // algoritma Pop
    }
}
```

Pemakaian Kelas Stack

```
#include "Stack.h"
#include <iostream>
using namespace std;

int main () {
    Stack s;
    int n;
    try {
        // instruksi
        s << 10;
        // instruksi
    }
    catch (StackExp& ) {
        s.DisplayMsg();
    }

    n = StackExp::NumException();
    if (n > 0) { cout << "Muncul " << n << " stack exception" << endl; }

    return 0;
}
```

Chain Exception

- › Method bisa merespon terjadinya exception dengan melempar exception lagi

```
try {  
    // ...  
}  
catch (IOException e) {  
    throw new SampleException("Other IOException", e);  
}  bisa lgsg throw e
```

CPP Standard Exception

exception	description
<code>bad_alloc</code>	thrown by <code>new</code> on allocation failure
<code>bad_cast</code>	thrown by <code>dynamic_cast</code> when it fails in a dynamic cast
<code>bad_exception</code>	thrown by certain dynamic exception specifiers
<code>bad_typeid</code>	thrown by <code>typeid</code>
<code>bad_function_call</code>	thrown by empty function objects
<code>bad_weak_ptr</code>	thrown by <code>shared_ptr</code> when passed a bad <code>weak_ptr</code>

exception	description
<code>logic_error</code>	error related to the internal logic of the program
<code>runtime_error</code>	error detected during runtime

```
// Example: bad_alloc standard exception
#include <iostream>
#include <exception>
using namespace std;
int main () {
    try {
        int* myarray= new int[1000];
    } catch (exception& e) {
        cout << "Standard exception: " << e.what() << endl;
    }
    return 0;
}
```