

Exception di C++

Yohanes Nugroho
Departemen Teknik Informatika
Institut Teknologi Bandung

Overview

- Apakah *exception* itu?
- Menangani *error* tanpa menggunakan *exception*
- Sintaks *exception handling*
- Alur program karena adanya *exception*
- Contoh *exception handling* sederhana
- Kelas *Exception*
- Mendefinisikan method yang melempar *exception*
- *Exception* pada konstruktor
- Kegunaan lain *Exception*
- Penggunaan *Exception* yang jelek
- Catatan akhir

Konvesi dan Catatan Slide

- Semua nama Exception yang tidak diawali `std::` adalah nama buatan programmer
- Blok kode ditulis dengan indentasi dan penempatan kurung (braces) yang tidak standar agar lebih mudah dilihat (dan muat dalam satu slide)
- Quote diambil dari
 - `/usr/share/games/fortune/*`

Exception

- Suatu konstruksi bahasa khusus untuk menangani keadaan yang tidak terduga (biasanya adalah Error)
- ***Perhatikan:***
 - Error tidak harus selalu ditangani dengan exception (namun Exception mempermudah penanganan Error)
- Exception bekerja dengan cara mengubah alur eksekusi program, sambil melempar suatu objek tertentu sebagai informasi untuk alur yang baru

Error ...

- *At the source of every error which is blamed on the computer you will find at least two human errors, including the error of blaming it on the computer.*

Menangani error tanpa exception

- **Untuk setiap kemungkinan kesalahan, perlu pengecekan**

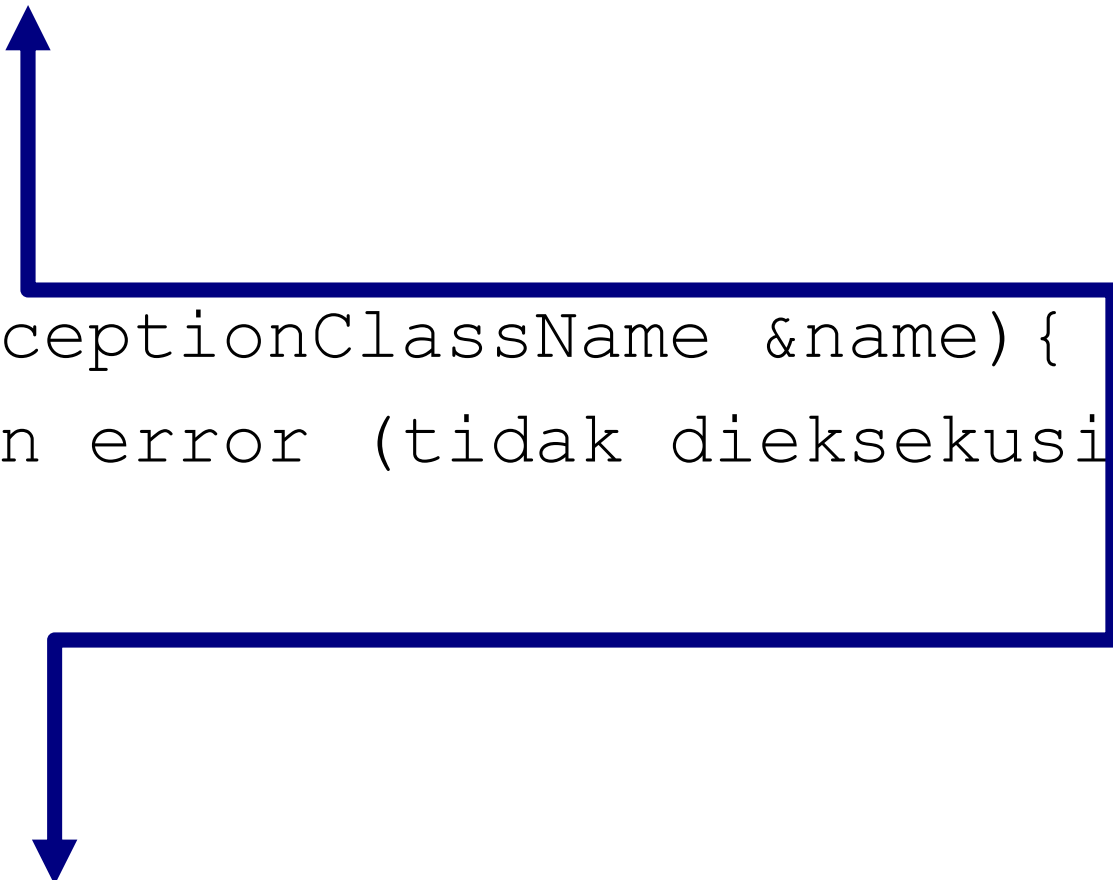
```
if (! (f = fopen("hello.txt", "r"))) {  
    printf("Error"); return;  
}  
  
if (!fgets(line1, sizeof(line1), f)) {  
    fclose(f); printf("Error"); return;  
}  
  
if (!fgets(line2, sizeof(line2), f)) {  
    .... }  
}
```

Sintaks Exception Handling

```
try {  
    aksi_1 ();  
    aksi_2 ();  
    aksi_3 ();  
} catch (ExceptionClassName &name) {  
    //penanganan error  
}  
  
aksi_4 ();  
aksi_5 ();
```

Alur Normal

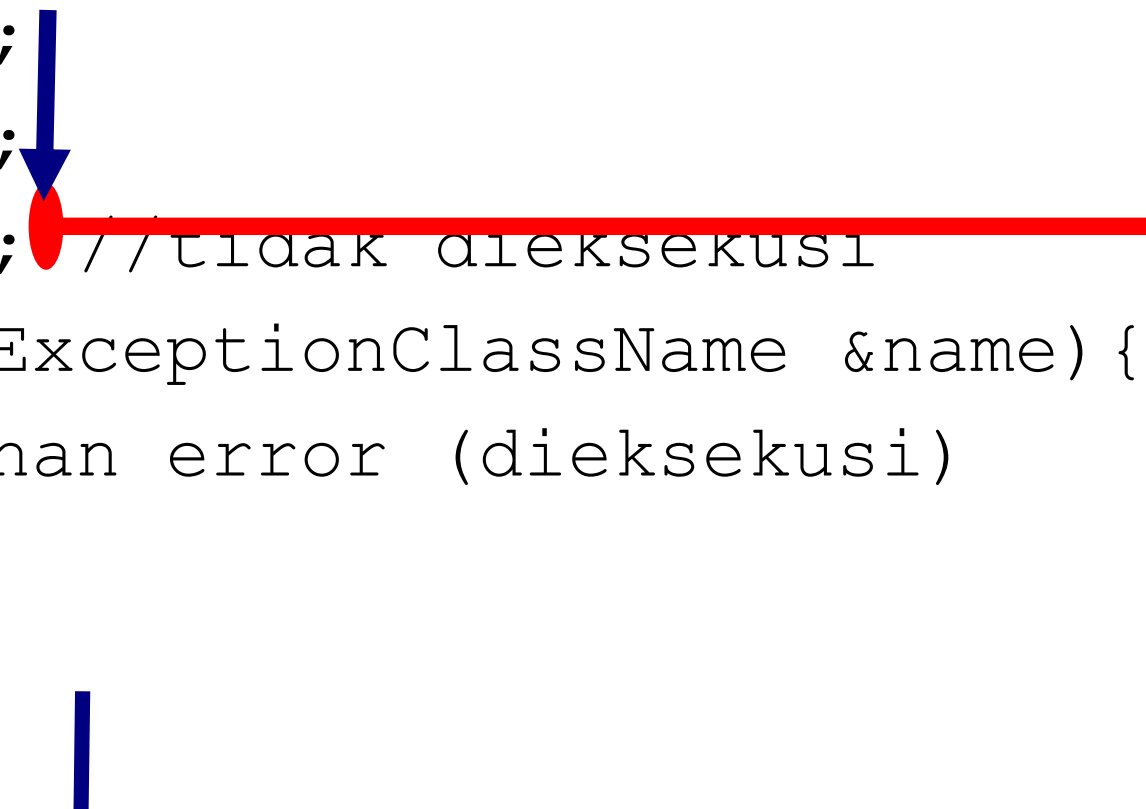
```
try {  
    aksi_1();  
    aksi_2();  
    aksi_3();  
} catch (ExceptionClassName &name) {  
    //penanganan error (tidak dieksekusi)  
}  
aksi_4();  
aksi_5();
```



Alur Jika Exception Dilempar

Aksi_2();

```
try {  
    aksi_1();  
    aksi_2();  
    aksi_3(); //tidak dieksekusi  
} catch (ExceptionClassName &name) {  
    //penanganan error (dieksekusi)  
}  
aksi_4();  
aksi_5();
```



Contoh Penggunaan Exception

```
#include <iostream>
using namespace std;
int main ()
{
    try {
        cout << "Masukkan Angka:";
        int num; cin >> num;
        if (num>10) throw std::exception();
        cout << "Angka kurang dari atau sama dengan 10" <<
            endl;
    } catch (std::exception& s) {
        cout << "Angka Lebih dari 10" << endl;
    }
    return 0;
}
```

Catatan

- Dalam statement
 - `throw std::exception()`, sebuah instans dari kelas `std::exception()` diciptakan (konstruktor defaultnya dipanggil)
- Jika kelas memiliki konstruktor yang lain, konstruktor itu dapat dipanggil:
 - `throw std::out_of_range("lebih dari 12");`
- Dalam except, method kelas exception dapat dipanggil:
 - `cout << e.what();`

Klausur Catch pada Exception

- Boleh ada beberapa catch dalam satu try

```
try {  
    //aksi  
  
} catch (ExceptionA &a) {  
    //Aksi jika ada ExceptionA  
  
} catch (ExceptionB &b) {  
    //Aksi jika ada ExceptionB  
  
}
```

Nested Exception Handling

- Boleh ada blok try catch dalam try catch, penangkap adalah blok terdekat

```
try {  
    fungsi_a();  
    try {  
        fungsi(b);  
    } catch (NumberFormatException &n) { }  
} catch (FatalException &fatal) {  
}
```

Aneka cara menangkap Exception

- Dilempar dengan **throw** `Error(123);`
 - Ditangkap dengan

```
catch (Error e)
catch (Error &e),
```
 - tapi gunakan selalu cara yang kedua (lebih efisien)
- Dilempar dengan **throw** **new** `Error(123);`

```
catch (Error *e) {
    delete e;
}
```

Melempar kembali exception

- Dalam catch, exception bisa dithrow:

```
Catch (Exception &e) {throw e;}
```

- Exception baru bisa dibuat:

```
Catch (FileNotFoundException &e)  
{throw FatalException(); }
```

- Gunanya agar exception ditangkap oleh klausa exception terluar, atau method yang memanggil method ini.

Exception pada method

```
void connect_internet() {  
    //coba melakukan koneksi  
    if (error) throw  
        connect_exception("error");  
    //....  
}  
  
//...  
  
//bagian main  
  
try { connect_internet(); }  
catch (connect_exception& e) { /*do  
    something*/ }
```


Catatan

- `std::exception` adalah *nama kelas* exception standar milik C++
- Programmer bisa membuat sendiri kelas Exception, contoh kelas exception yang paling sederhana:
 - **class** my_exception { };
- Ada beberapa petunjuk untuk membuat kelas exception yang baik

Membuat Kelas Exception Yang Baik

- Buat beberapa variabel private untuk:
 - Mencatat jumlah exception yang terjadi (variabel statik)
 - Mencatat jenis exception
 - Mencatat Pesan Exception
- Semua hal di atas tidak wajib ada, tapi sebaiknya ada
- Buat kelas exception yang diturunkan dari kelas exception lain yang lebih umum

Contoh: smallintexcept

```
// file smallint.cc  
// kelas yang merepresentasi small integer, integer  
// 0..10  
// jika terjadi penjumlahan di luar 0..10 maka  
// exception
```

```
class smallintexcept  
{  
private:  
    static int num_except;  
    static char *msg[];  
    const int msg_id;
```

Konstruktor Kelas smallintexcept

public:

```
// constructor dengan parameter nomor exception
// ketika dihidupkan "memberskan" nilai konstanta msg_id
smallintexcept (int X):msg_id (X)
{
    num_except++;
}

// constructor dengan parameter nomor exception
// ketika dihidupkan "memberskan" nilai konstanta msg_id
smallintexcept (const smallintexcept & si):msg_id (si.msg_id)
{
    cout << "ctor except ";
}
```

Method

```
static int NumException ()  
{  
    return num_except;  
}  
void response ()  
{  
    cout << msg[msg_id];  
}  
};
```

```
// inisialisasi static member
```

```
int smallintexcept::num_except = 0;
```

```
Char * smallintexcept::msg[] = { "TOOBIG", "TOOSMALL" };
```

Kelas SmallInt yang melempar exception

```
class smallint
```

```
{
```

```
public:
```

```
    smallint () { value = 0;}
```

```
    smallint (int X) { value = X;}
```

```
    ~smallint () {cout << " dtor, do nothing" << endl; }
```

```
    operator int () {return value; }
```

```
    void PrintVal () {cout <<"Value="<<value<< endl;}
```

```
    void ReadVal () { cin >> value;}
```

Lanjutan kelas smallint

```
// menambahkan X ke current value
void Plus (smallint X) {
    value = value + X.value;
    if (value > 10){throw (smallintexcept (0));}
    if (value < 0) {throw (smallintexcept (1));}
}

private:
    int value;
};
```

Menangkap Exception yang tidak ditangani

- ***Never test for an error condition you don't know how to handle.***
- ***-- Steinbach***
- Ada banyak exception yang mungkin tidak bisa ditangani, gunakan `std::exception` (yang memiliki method **what()**) dan `"..."` untuk menangani exception lain yang tidak tertangani


```
try { /*...*/  
    } catch (Exception &a) { /*do  
        something*/  
        /**/  
    } catch (Exception &b) {  
        /**/  
    } catch (std::exception &c) {  
        //standard exception  
        cout << a.what() << endl;  
    } catch (...) { /*unknown error*/ }
```

Main

```
#include <iostream>
using namespace std;
#include "smallint.h"
int main () {
    cout << "start of smallint ...\n";
    smallint S1 (1);
    cout << "S1="; S1.PrintVal ();
    smallint S;
    cout << "baca nilai = "; S.ReadVal ();
    cout << "S="; S.PrintVal ();
}
```

Lanjutan Main

```
try {  
    S1.Plus (S);  
    cout << "hasil S1= S1+S =";  
    S1.PrintVal ();  
}  
catch (smallintexcept & e) {  
    e.response ();  
}  
int n;  
n = smallintexcept::NumException ();  
if (n > 0) { cout << "ada exception"; }  
return 0;  
}
```

Contoh Penurunan Kelas Exception

```
class NetException { };
```

```
class DisconnectedException :  
    public NetException{ };
```

```
class TimeoutException: public  
    NetException { };
```

- Exception juga bisa diturunkan dari dua kelas atau lebih (multiple inheritance)
 - Misalnya error mengakses file di jaringan yang merupakan FileNotFoundException dan NetException

Menangkap Exception yang diturunkan

/*tangkap dari yang paling spesifik*/

```
try {  
    /* ... */  
} catch (DisconnectedException &d) {  
    /* eksepsi karena koneksi putus */  
} catch (TimeoutException &t) {  
    /* eksepsi karena timeout */  
} catch (NetException &n) {  
    /*eksepsi jaringan yang tidak dikenal*/  
}
```

Mendeklarasikan Method yang Melempar Exception

- **Deklarasi**

- **void** logout() **throw** (NetException, TimeoutException);

- method/fungsi logout hanya boleh melempar NetException dan/atau TimeoutException

- **Definisi** method/fungsi harus sama dengan deklarasinya

Exception pada Konstruktor

- Jika terjadi exception pada konstruktor, maka object tidak akan terbentuk

```
BujurSangkar *bs;
```

```
try {
```

```
    bs = new BujurSangkar(10);
```

```
} catch (TooBigException& e) {
```

```
    //bs tidak perlu di-delete
```

```
}
```

Kasus Khusus: Exception pada Konstruktor

- Jika sudah terlanjur ada memori yang dialokasi (atau *resource* yang di-*acquire*) pada konstruktor, dan ada exception, maka akan ada memory leak
- Salah satu solusi termudah: coba pindahkan bagian yang mengalokasi resource ke method lain, atau gunakan *auto_ptr*.
- Catatan: explore *auto_ptr*

Contoh memory leak pada konstruktor

```
GambarBujurSangkar (int sisi) {  
    Buffer b = new Buffer(sisi*sisi);  
    /*anggap ini selalu sukses*/  
    if (!loadFile("bujursangkar.bmp")) {  
        throw std::exception();  
    }  
}
```

Salah satu solusi

```
GambarBujurSangkar (int sisi) {  
    this.sisi = sisi;  
}  
  
void loadFile() {  
    Buffer b = new Buffer(sisi*sisi);  
    /*anggap ini selalu sukses*/  
    if (!loadFile("bujursangkar.bmp")) {  
        throw std::exception();  
    }  
}
```

Penggunaan Lain Exception

- Exception bisa digunakan selain untuk penanganan kesalahan, misalnya untuk mengembalikan hasil suatu fungsi

```
void search(Buffer *b) {  
    //...  
    if (found) throw element;  
    //...  
}
```

- Dan cara mencarinya adalah ...

Menangkap nilai dengan exception

```
try {  
    search(b) ;  
} catch (element &e) {  
    cout << "found:" << e;  
}
```

- Tapi cara ini disarankan hanya jika memperbaiki keterbacaan program
- Dan cara di atas jelas **tidak memperbaiki keterbacaan program**

Penggunaan Exception yang Jelek

- Sebaiknya jangan gunakan:

```
try {  
    //read string  
} catch (EndOfFileException &e) {  
    //done on EOF  
}
```

- Tapi gunakan:

```
while (!endOfFile()) { /*read  
    string*/ }
```

Catatan Akhir

- C++ memiliki banyak cara dan konvensi dalam menangani kesalahan, exception adalah salah satunya (Baca buku Stroustrup)
- Exception dapat digunakan untuk banyak hal, tapi sebaiknya hanya untuk menangani kesalahan
 - Bijaksanalah kapan menggunakan exception
- And remember
 - One person's error is another person's data.