* Exception Handling:

→ Farklı mekanizması var. Ya handle edilir, ya da sonlanır.

→ Happy path ve unhappy path birbirinden temiz bir şekilde ayrılmış

→ Hatayı fırlatan kod ile, hatayı handle eden kod arasında doğrudan bir link vardır.

- Exception throwing:

- Fonksiyon işini yapamadığında bir nesne oluşturur. Bu nesnenin varlık nedeni, işini yapamadığını daha üst katmanlardaki kodlara bildirmek.
- Bu nesneyi yakalayan kod müdahale eder.
- Gönderilen hata ya yakalanmalı, ya da kaynak kaybı yaşanmadan program sonlanmalı (resumptive / terminative)

- Try Block:

- Try Block da block scope kurallarına tabi. (otomatik ömürlü nesneler, block sonu hayatı biter)

- Catch:

```
try {
    ///
}
catch(int) {
    ///
}
catch(double) {
    ///
}
catch(unsigned int) {
    ///
}
```

Gönderilen hata nesnesine bağlı olarak handler'a girer

→ Çift parametre ile catch yapılmaz.

→ ellipsis (...) ile type gönderilemez catch edilir.

- Set_terminate function:

```
// using terminate_handler = void (*)();
typedef void (*terminate_handler)();
```

```
terminat_handler set_terminate(terminate_handler);
```

```
void necfunc()
{
    std::cout << "necfunc cagrildi cunku necfunc islevini std::terminate cagirdi\n";
    std::exit(EXIT_FAILURE);
}

int main()
{
    set_terminate(&necfunc);
    std::cout << "main cagrildi\n";
    f1();
    std::cout << "main sona erdi\n";
}
```

bu olmasaydı abort çağırırdı.

- terminate default olarak abort'u çağırır.
- biz set_terminate'e bir fonksiyon adresi verirsek abort yerine o func çağırılır.
- bir daha çağırıldığında, set_terminate geri dönüş değeri, necfunc adresi → daha önce kaybolmuş func addr.

```cpp
int main()
{
    std::cout << "main cagrildi\n";

    try {
        f1();
    }
    catch (int) {
        std::cout << "main fonksiyonu icinde hata yakalandi\n";
    }

    std::cout << "main sona erdi\n";
}
```

→ Tür dönüşümü ve integral promotion, catch parametresinde çalışmaz X Aynı tür olmalı.

→ Bir switch case gibi yapı var. İşlem bittikten sonra break. ve koda devam eder.

#Önemli İstisna:
- Catch parametremiz, base reference olduğunda, hem base, hem derived class ref catch edilir.

```cpp
catch(std::exception &ex)
{
    std::cout << ex.what() << "\n";
}
```

Kopyalama işlemi yapmamak / object slicing yaşanmamak için. class reference

std::exception içerisinde virtual func.

Farklı overloadları var.

***Exception Hiyerarşisi:**

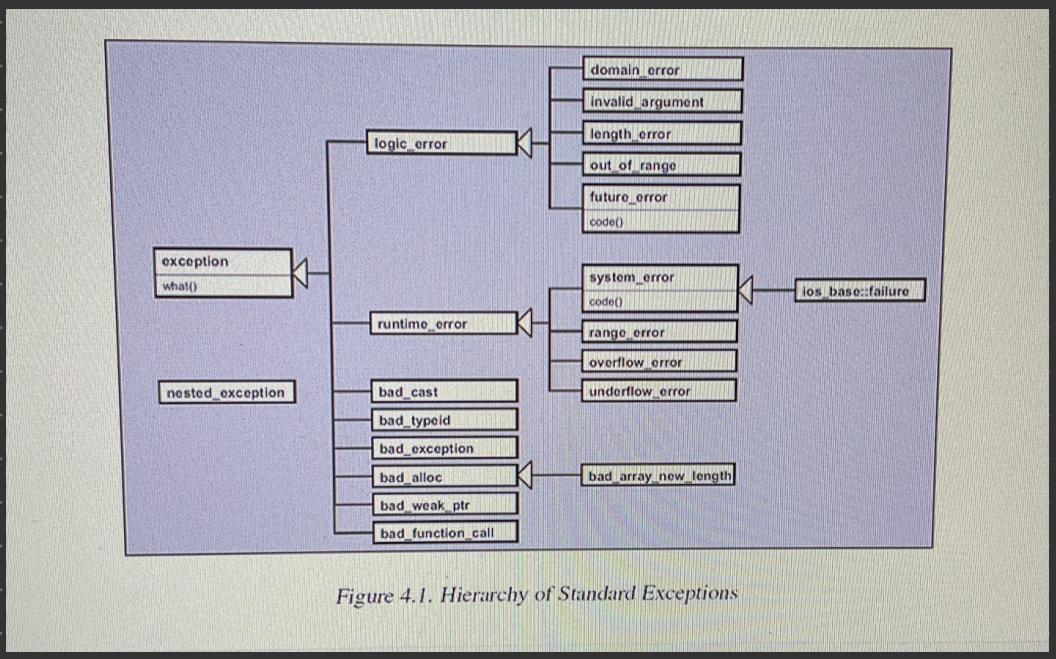

Figure 4.1. Hierarchy of Standard Exceptions

```cpp
int main()
{
    try {
        func();
    }
    catch (const std::exception& ex) {
    //catch (const std::logic_error& ex) {
    //catch (const std::out_of_range& ex) {
        std::cout << "exception caught: " << ex.what() << '\n';
    }
}
```

→ Bu üç catch bloğu, aynı exception'ı yakalayabilir. Çünkü

out of range → logic err → exception hiyerarşisi var.

* Fakat burada switten case gibi, en baştan en sona tarayarak gider. Bu yüzden en özelden genele doğru bir sıralama yapılmalı

* Hemen her zaman throw ifadeleri, geçici nesne oluşturur gibi class çağırmalı.

```
void func()
{
    BadDate bd;

    throw bd;
}
```

Bu kod bd throw etmez ✗

→ Derleyicinin BadDate sınıfı tarafından bir hata nesnesi oluşturması için BadDate'in copy constructorına çağrı yapılacak.

```
void func()
{
    throw std::bad_alloc{};
}
```

* Catch bloğunda ne yapacağını bilmiyorsan → yakalama

* kaynağı catch bloğu vermez → Exception yığınından / destructor ile kaynak verilir. Biz, bu destructor'ı çağırmayı garantilemeliyiz.

* Exception Translate: - Exception yakalanır, bazı işlemler yapılır ya da yapılmaz, fakat aynı class tarafından değil, kendi seçtiğimiz kendi oluşturduğumuz anlayacağı şekilde başka exception throw edilmesi

* Exception Retrow: - Yakalanan exception, kendi tarafından exception gönderir.

```
void func()
{
    try {

    }
    catch (const std::exception& ex) {
        //...
        throw;
    }
}
```

→ hiç bir class belirtilmezse, retrow edilir. Syntax bu şekilde

Fakat karıştırma

```
void func()
{
    try {

    }
    catch (const std::exception& ex) {
        //...
        throw ex;
    }
}
```

Bu retrow değil!!