Kivételkezelés

```
//hiba kódok
const int ERROR_OK = 0;
const int ERROR_PARAMETER_INVALID = 1;
const int ERROR_PARAMETER_NULL = 2;
const int ERROR_FILE_OPEN = 3;
```

main ->f1 -> f2 -> f3 és f1-et hívja //Vezérlési fa ezek a függvények mint int visszatérésűek

---

```
int f1()
{
	int errorCode = f2();
	if(errorCode != ERROR_OK) return errorCode;
	…
}

int f2()
{
	FILE* f = fopen(…,…);
	if(f == NULL) return ERROR_FILE_OPEN;
	int errorCode = f3();
	if(errorCode!=ERROR_OK) return errorCode;
	…
	errorCode = f4(i, ptr); // hívónak továbbítjuk a hibát
	…
}

int f3() … //ez tök mindegy (csak)

int f4(int i, int *ptr)
{
	if(i == 0) return ERROR_PARAMETER_INVALID;
	if(ptr == NULL) return ERROR_PARAMETER_NULL;
}
```

---

most ugyanez a probléma kivételkezelés használatával:

```
void main()
{
	try
	{
		f1();
	}
	catch(int errrorCode)
	{
		cerr << „Error code: „ <<errorCode << endl;
	}
```

```
        catch(const char* msg)
        {
                cerr <<mdg << endl;
        }

        …

        void f1() { f2();}

        void f2()
        {
                FILE* f = fopen(„…”, …);
                if(f == NULL) throw „Cannot open file”;
                f3();
                …
                f4(i,ptr);
        }

        void f4(int i, int* ptr)
        {
                if(i == 0) throw ERROR_PARAMETER_INVALID;
                if(ptr == NULL) throw ERROR_PARAMETER_NULL;
                …
        }
}
```

---

main
→ f1
-→f2
→f3
→f4

mindig visszafelé nézi, hogy van-e megfelelő catch blokk…

- először nézi hogy a saját try blokkjához tartozó catch blokk megfelel-e
- ha nem felel meg, akkor tovább megy egy másik tryig…
- ha nem talál megfelelő catch blokkot, akkor a main elkapja, de a program elszáll

---

```
void f2()
{
        try
        {
                f3;
                f4;
        }

        catch(int errorCode)
        {
                //speciális hibakezelés
        }
```

}

→ probléma: adatfolyás (bárhol elszállhat a program)

megoldás:

catch(…) // ez mindent elkap!

```
//f  (file megnyitása)
…
try{
        f3();
        f4(i, ptr);
}

catch(…)
{
        fclose(f);
        throw; //EZT CSAK CATCH(…) – BEN LEHET HASZNÁLNI
}

fclose(f);
```

- a paraméter nélküli throw-val tovább dobjuk a kivételt…kezelje aki tudja (így sikerült bezárni a fájlunkat ☺

```
int main()
{
        try
        {
                fv1();
        }
        catch(int errorCode)
        {
        //kezelés
        }
}

void fv1()
{
        Fifo f;
        fv2();
}

void fv2(9
{
        int i;
        throw 42;
}
```

main
→ fv1
→fv2

→adatfolyás! … veremben még maradnak cuccok ~~ verem visszacsévélése(stack rewind)
→ i felszabadul
→ f destruktor

!!! destruktorból soha ne dobjunk kivételt!

- uncaught_excpection (true/false a visszatérés)

T típusú kivétel elkapja, ha

- ha a catch kivételváltozója T
- ha a catch kivételváltozója őse a T-nek
- ha referencia/const referencia T-re
- ha referencia/const referencia T-ősére
- pointer: létezik T*-ról konverzió

→POINTERES catch-et NE használjuk (nem tudjuk hogy fel kell-e szabadítani vagy nem) → dinamikus v. statikusan lett-e foglalva

→ mindig a legspeciálisabb catchblokkot kell előre tenni !

---

exception

↑

logic error

↑ ↖

nullpointer_exception *invalid_argument exception*

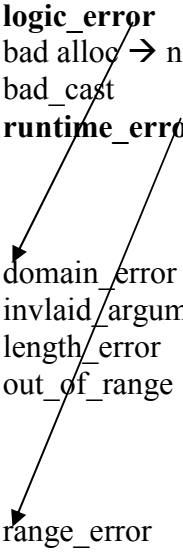
```
catch(const exception &e)
{
        e.what();
}
```

→ try blokkban van az egész main

```
int main()
{
	…
	try
	{
		… // az egész program
	}
	catch(const exception &e)
	{
		cerr << e.what() << endl;
	}
	catch(…)
	{
		//itt nem tudunk mit kiiratni, max. elköszönhetünk a felhasználótól ☺
		cerr << „Bye.” << endl;
	}
}
```

Beépített exception osztály

- **logic_error**
- bad alloc → new helyfoglalására
- bad_cast
- **runtime_error**



- domain_error
- invlaid_argument
- length_error
- out_of_range

- range_error
- overflow_error
- underflow_error

Saját kivétel írása

```
class nullpointer_exception: public exception
{
	char msg[…];
	char variable […];
public:
	nullpointer_exception(const char* vname)
	{
		sprintf(variable, vname);
		sprintf(msg, „Null pointer assigment %s”, vname);
	}

	const char * what() const
	{
		return msg;
	}
};

throw nullpointer_exception(„ptr”);
```