8. Exceptions

June 21, 2023

1 Error handling and exceptions

1.1 Sanity checks

Sanity check is the procedure to check the correctness of an instruction (or block of code) and act accordingly. Usually consist of: * Check of the return code of previous action. * Advising the user about the error. * Return an error code to the calling function (or operating system) to identify hat happened.

Remark: an error code does not imply than an error was triggered. An return code of OK is also an error code.

You should know that: * Return errors are labeled and documented (see errno) * Each error has a code number and a text message among others… * The variable errno stores the error code (return code) of the last executed instruction. * The errno variable acts as an offset indexing an error definitions table. * The function perror() takes the error message that match the error code. * Printing a customized error string can be useful in locating the problem (debugging).

[1]:
```
#include <cmath> //because of log
#include <cerrno>
#include <iostream>

//extern int errno; //This variable is already defined in the system. We are␣
↪not redefining it, just referring to it
double mynan=std::log(-1.0);
if (errno == EDOM) //EDOM = Error Domain Outside Maths
    std::perror ("log(-1) not defined!");
else
    std::cout << "mynan" << std::endl;
```

1.1.1 Well known errno codes

1.2 Exceptions

1.2.1 Motivation. Why do we need such a "concept" if we have return codes?

Imagine you are at depth N in the code and an error occurs: * At least you have to compare N times the return code of the caller function (predecesor function). * On each depth level you should print the an error message and return the code to the upper function until main. * The main returns to the system/other processes the latest error code.

It is very tedious and works only with non-void functions, since void function can not return the error code.

It will much better to be able to *interrupt* the programm trace at any point <– **Exceptions**

1.3 What are exceptions

- An exception is a problem that arises during the execution of a program.
- A C++ exception is a response to an exceptional circumstance that arises while a program is running, ie: Divide by zero, out of bounds, method not implemented…
- Exceptions provide a way to transfer control from one part of a program to another part interrumping the normal programm trace.

1.4 Concepts

C++ exception handling is built upon three keywords: * **throw**: throws an exception when a problem shows up. * **catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception. * **try**: A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

1.4.1 Try / catch combination

- Assuming that a block of code will raise an exception, a method catches an exception using a combination of the **try and catch keywords**.
- A try/catch block is placed around the code that might generate an exception.
- Code within a try/catch block is referred to as protected code.
- You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

try // protected code catch(ExceptionName e1) // catch block catch(ExceptionName e2) // catch block catch(ExceptionName eN) // catch block

1.5 Throwing exceptions

- Exceptions can be thrown anywhere within a code block using throw statements.
- Also constructors and destructors can throw exceptions.
- The operand of the throw statements:
  - determines a type for the exception
  - can be any expression
  - type of the result of the expression determines the type of exception thrown.

1.5.1 Examples

[2]:
```
int secureDivide(int x, int y)
{
    if (!y) throw ("Division by zero");
    return x/y;
}
```

[3]:
```
#include<fstream>

int countLines(std::string filename)
{
    int n=0;
    std::ifstream ifs;
    std::string line;

    ifs.open(filename.c_str());
    if (ifs.is_open())
        while (std::getline(ifs, line))
            n++;
    else
        throw (0);

    ifs.close();
    return n;
}
```

1.6 Catching exceptions

- The catch block following the try block catches any exception.
- The type of exception to be caught
  - can be specified
  - is determined by the exception declaration that appears in parentheses following the keyword catch.

[7]:
```
try
{
    secureDivide(3, 0);
    NoL = countLines("a.txt");
}
catch (const char* s)
{
    std::cout << "Error: " << s << std::endl;
}
catch (int r)
{
    std::cout << "Number of lines: " << r << std::endl;
}
```

```
Error: Division by zero
```

1.7 Questions to you!!

- Can we throw exceptions in constructors?
  - Does it make sense?
- What happens if an exception is thrown but not caught?

- Can the main function throw exceptions?
  - Does it make sense?
- Can we define our own exceptions and throw them?
  - Does it make sense?

[ ]:
```
```