Chapter 9. Errors and exceptions

Programming Concepts in Scientific Computing

EPFL, Master class

November 1, 2023

Management of errors: Sqrt

```
double mysqrt(double x) {
  // x has to be positive
  return std::sqrt(x);
}
```

Management of errors: Sqrt

```
double mysqrt(double x) {
  // x has to be positive
  return std::sqrt(x);
}
```

How to manage errors ?

Management of errors: Sqrt

```
double mysqrt(double x) {
  // x has to be positive
  return std::sqrt(x);
}
```

How to manage errors ?

Good practices ?

Management of errors: error code

```
double mysqrt(double x) {

  if (x < 0)
    return -1; // the code error

  return std::sqrt(x);
}
```

Management of errors: error code

```
double mysqrt(double x) {

  if (x < 0)
    return -1; // the code error

  return std::sqrt(x);
}
```

Mixing error and result

Error code is not human readable

Management of errors: assert

```
double mysqrt(double x) {

  assert(x > 0);

  return std::sqrt(x);
}
```

- No error code, nor error information

Management of errors: assert

```
double mysqrt(double x) {

  assert(x > 0);

  return std::sqrt(x);
}
```

- No error code, nor error information
- The program stops

Management of errors: assert

```
double mysqrt(double x) {

  assert(x > 0);

  return std::sqrt(x);
}
```

- No error code, nor error information
- The program stops
- Removing the assert: `g++ -DNDEBUG` (CMake config in CLion)

Management of errors: throwing exceptions

```
double mysqrt(double x) {
  if (x < 0)
    throw(-1);
  return std::sqrt(x);
}
```

Management of errors: throwing exceptions

```
double mysqrt(double x) {
  if (x < 0)
    throw(-1);
  return std::sqrt(x);
}
```

Calling:

```
mysqrt(-1);
```

Management of errors: throwing exceptions

```
double mysqrt(double x) {
  if (x < 0)
    throw(-1);
  return std::sqrt(x);
}
```

Calling:

```
mysqrt(-1);
```

- Equivalent to returning an error code

Management of errors: throwing exceptions

```
double mysqrt(double x) {
  if (x < 0)
    throw(-1);
  return std::sqrt(x);
}
```

Calling:

```
mysqrt(-1);
```

- Equivalent to returning an error code
- Independent of function return

Management of errors: catching exceptions

```
double mysqrt(double x) {
  if (x < 0)
    throw(-1);
  return std::sqrt(x);
}
```

Management of errors: catching exceptions

```
double mysqrt(double x) {
  if (x < 0)
    throw(-1);
  return std::sqrt(x);
}
```

Catching the exception:

```
try {
  mysqrt(-1);
} catch (int i) {
  std::cout << "Code: " << i << std::endl;
}
```

Exceptions can be of **any** type/class

Management of errors: exception classes

```
struct Exception {
  Exception(const std::string &mesg) : mesg(mesg){};

  const std::string &what() { return mesg; };

  std::string mesg;
};


double mysqrt(double x) {
  if (x < 0) {
    throw Exception("no negative number sqrt");
  }
  return std::sqrt(x);
}
```

Management of errors: catching exceptions

```
try {
  double res = mysqrt(-1);
  std::cout << res << std::endl;
} catch (Exception &e) {
  std::cout << e.what() << std::endl;
}
```

Management of errors: exceptions

```
struct NegativeException {};
struct InfException {};

double mysqrt(double x) {
  if (x < 0) {
    throw NegativeException();
  }
  if (std::isinf(x)) {
    throw InfException();
  }
  return std::sqrt(x);
}
```

Management of errors: exceptions

```
try {
  res = mysqrt(x);
} catch (NegativeException &e) {
  std::cout << "Negative" << std::endl;
} catch (InfException &e) {
  std::cout << "Inf" << std::endl;
} catch (...) {
  std::cout << "Unknown exception" << std::endl;
}
```

Management of errors: STL exceptions

```
class MyException1 : public std::exception {};
class MyException2 : public std::exception {};

void foo() { throw MyException1(); }
```

Management of errors: STL exceptions

```
class MyException1 : public std::exception {};
class MyException2 : public std::exception {};

void foo() { throw MyException1(); }
```

Can be caught using inheritance

```
try {
  foo();
} catch (const std::exception &e) {
  std::cout << "caught exception" << std::endl;
}
```

Management of errors: STL exceptions

```
void foo() { throw std::runtime_error("my message"); }

int main() {
  try {
    foo();
  } catch (const std::runtime_error &e) {
    std::cout << e.what() << std::endl;
  }
}
```

Error handling

Take away message

- **assert**: Conditions a code (brutal) stop
- **Exceptions**: (Good) Mechanisms to manage error
- **throw**: instruction to send error aside of the normal flow of the program
- **try/catch**: Block of instruction where exception are managed
- **std::runtime_error**: exception with message

Error handling

Take away message

- **assert**: Conditions a code (brutal) stop
- **Exceptions**: (Good) Mechanisms to manage error
- **throw**: instruction to send error aside of the normal flow of the program
- **try/catch**: Block of instruction where exception are managed
- **std::runtime_error**: exception with message
- Want more: https://www.codeproject.com/Articles/38449/C-Exceptions-Pros-and-Cons