Exceptions

A good quality program should be:
- correct
- robust == no disaster in unexpected situations

Problem handling alternatives
==========

Function return types + error codes:
+ Trivial and exists in all languages
- Cant use return type
- Non-mandatory to handle
- Commits to action on error

Status flag:
(like C++ i/o)
+ Trivial and easy to implement
- Non-mandatory to handle
- Have to "clear" before new action is executed

Asserts
+ Trivial
- No recovery
- What the user can do with that?

Exceptions
+ Type-safe transmission of arbitrary data from throw-point to handler
+ Every exceptions should be caught by the appropriate handler
+ No extra code/space/time penalty if not used
+ Grouping of exceptions
+ Works fine in multithreaded environment
- Hard to implement
- Run-time penalty

Maybe monad := { bool, T }
C++17: std::optional
+ Flexible
+ Cheap
  - Changes the interface from T to optional<T>

History
=======
FORTRAN EOF
PL/I raise
C setjmp/longjmp

ML: the exceptions are from the same type system as other types
-> C++ -> Java -> C#

Can we specify/statically check the possible exceptions?
Java: void f() throws Exp1;
void g() { f(); }
C++ before C++11: no
C++ after C++11: void g() nothrow { f(); } // run-time abort if ex.

What is with unhandled exceptions?

Can we re-raise/retrow exceptions?

Exception-safe programming

```
X&  operator= (const X& phs)
{
  if ( this != &rhs )  // avoid x = x;
  {
    drop_old_resources();
    allocate_new_resorces();
    copy_values_from(rhs);
  }
  return *this;
}
```