Error handling and testing

Victor Eijkhout, Susan Lindsey

Fall 2022

`last formatted:  September 6, 2022`

1. **Programming and correctness**

Find your favorite example of costly programming mistakes . . .

What to do about it?

- Never make mistakes.
- Prove that your program is correct.
- Test your program before deploying it.
- Handle errors as they occur.

Error handling

2. Assertions to catch logic errors

Sanity check on things ‘that you just know are true’:

```
#include <cassert>
...
assert( bool expression )
```

Example:

```
x = sin(2.81);
y = x*x;
z = y * (1-y);
assert( z>=0. and z<=1. );
```

3. Using assertions

Check on valid input parameters:

```
#include <cassert>

// this function requires x<y
// it computes something positive
float f(x,y) {
  assert( x<y );
  return /* some result */;
}
```

Check on valid results:

```
float positive_outcome = f(x,y);
assert( positive_outcome>0 );
```

4. Example

```
int collatz_next( int current ) {
  assert( current>0 );
  int next{-1};
  if (current%2==0) {
    next = current/2;
    assert(next<current);
  } else {
    next = 3*current+1;
    assert(next>current);
  }
  return next;
}
```

5. Use assertions during development

Assertions are disabled by

```
#define NDEBUG
```

before the include.

You can pass this as compiler flag:

```
icpc -DNDEBUG yourprog.cxx
```

6. Exceptions

Not every error is fatal:

$$\text{Exception} \equiv \begin{cases} \text{`this should not happen'} \\ \text{but we can handle it} \end{cases}$$

1. recover from the problem
2. graceful exit

7. Exceptions

Have you seen the following?

Code:

```
vector<float> x(5);
x.at(5) = 3.14;
```

Output
[except] boundthrow:

```
libc++abi.dylib: terminating with
    uncaught exception of type
    std::out_of_range: vector
```

The Standard Template Library (STL) can generate many exceptions.

- You can let your program crash, and start debugging
- You can try to catch and handle them yourself.

8. Exception structure

Code with problem:

```
if ( /* some problem */ )
  throw(5);
  /* or: throw("error"); */
```

```
try {
  /* code that can go wrong */
} catch (...) { // literally
    three dots!
  /* code to deal with the
    problem */
}
```

9. Exceptions

Assume a routine only works for certain values, and you want to generate an error if called with an inappropriate value.

```
double compute_root(double x) {
  if (x<0) throw(1);
  return sqrt(x);
}
int main() {
  try {
    y = compute_root(x);
  } catch (...) {
    /* handle error */
    cout << "Root failed, using default\n";
    y = 0;
  }
```

See book for more details.