Error handling and testing

Victor Eijkhout, Susan Lindsey

Fall 2025

last formatted: November 4, 2025

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':

```
1 #include <cassert>
2 ...
3 assert( bool expression )
```

Example:

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

3. Using assertions

Check on valid input parameters:

```
1 #include <cassert>
2
3 // this function requires x<y
4 // it computes something positive
5 float f(x,y) {
6   assert( x<y );
7   return /* some computation */;
8 }
```

Code design to facilitate testing the result:

```
1 float positive_outcome = /* some computation */
2 assert( positive_outcome>0 );
3 return positive_outcome;
```

4. Example

```
1 // collatz/modular.cpp
2 int collatz_next( int current ) {
3   assert( current>0 );
4   int next{-1};
5   if (current%2==0) {
6     next = current/2;
7     assert(next<current);
8   } else {
9     next = 3*current+1;
10    assert(next>current);
11  }
12  return next;
13 }
```

5. Use assertions during development

Assertions are disabled by

```
1 #define NDEBUG
```

before the include.

You can pass this as compiler flag:
`icpc -DNDEBUG yourprog.cpp`

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:

```
1 // except/boundthrow.cpp
2 vector<float> x(5);
3 x.at(5) = 3.14;
```

Output:

```
1 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

Illustration of exception throwing and catching.

Code with problem:

```
1 if ( /* some problem */ )
2   throw(5);
```

Calling code:

```
1 try {
2   /* code that can go wrong */
3 } catch (...) { // literally three
      dots!
4   /* code to deal with the problem
       */
5 }
```

9. Exceptions

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

```
1 double compute_root(double x) {
2   if (x<0) throw(1);
3   return sqrt(x);
4 }
5 int main() {
6   try {
7     y = compute_root(x);
8   } catch (...) {
9     /* handle error */
10    cout << "Root failed, using default\n";
11    y = 0;
12  }
```

See book for more details.