string toDollars(double amt)

{

ostringstream out;

out << setfill('0');

out << "$ ";

if (amt >= 1E6) {

out << static\_cast<long long>(amt / 1E6) << "," << setw(3);

}

if (amt >= 1E3) {

out << static\_cast<long long>(amt / 1E3) % 1000 << "," << setw(3); //amt divided by 1000, then take last 3 digits

}

out << static\_cast<long long>(amt) % 1000; //gives last 3 digits before change

out << "." << setw(2) << static\_cast<long long>(amt \* 100 + 0.5) % 100;

return out.str();

Why is .5 being added there (when the cents are being sent to output)?

**Precondition violations**

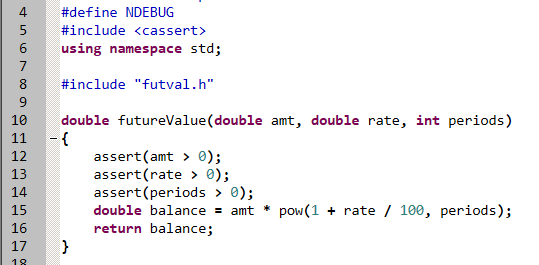
* What should a function do when given an **inappropriate** input value?
* Five options:
  + Fail “safely”: defensive programming (bad)
    - Makes it hard to find errors in your code
  + Return an error code the user can check
  + Throw an exception that can be caught
  + Terminate with an error message
  + Do nothing

**What is the “right thing”?**

* Errors caused by user input or errors caused by unforeseen circumstances
  + User types a filename incorrectly
  + Disk full when saving a file
  + Should be handled with exceptions
* Errors caused by the programmer
  + You want the error to “announce” itself
  + You **don’t** want to fail silently
  + This is where you should use **assert** (used to tell you, the programmer, what’s wrong as you’re developing the code)

**Using assert**

* To use **assert**, #include <cassert>
* To disabled assertions, and turn debugging off, #define NDEBUG before <cassert> is included



**Throwing Exceptions**

* Like an alternate return statement if things go wrong

