Read ALL of ch1, ALL of ch2, begin ch3 for project. ALL of both committees.

Beauty of trees

Intro to classes

2. An old thorn

Proper and improper use

1. assert() macro

0. Table X Program assignment handouts

CSI 310: Lecture 2
shut off position, AND it IS currently shut off.

Postcondition: the throttle has size positions above the

True on the call throttle(5)

False on the calls throttle(-1); throttle(0)

Precondition: size > 0.

type of that parameter is integer.

"true size" specifies it has exactly one function parameter and the

throttle( int size ) is a prototype

True? False? A jury can determine that
and the object).

is a complete English sentence ("is" is the verb, note the subject

"Porc. Chicken is a turkey.

pre, post-conditions are definite statements about process state:

C/C++ (standard C) assert() macro
judged true or false.

sentences about values and states of objects, and can be determined.

Note that grammatical form of these statements: They are all full

constructor function.

size refers to the parameter value for the call to the function.
(Of course, there can be bugs in both)

function, or if it occurs in the code that calls the function.

to distinguish whether a bug occurs inside the implementation of the

The assertion to check a precondition can enable the programmers

(3) if value \( i \neq 0 \), go on.

number the assertion was coded.

"fires" with a nice message that tells in which source file and line

(2) if the value \( = 0 \), make the process CRASH

the assertion

(1) evaluates the expression.

\[
\text{assert (useless expression)}
\]

body an assertion to check the function's preconditions.

Highly Recommended: code at the beginning of each function
do anything needed for program function.

For the same reason, expressions evaluated by assert() must not

Important.

done when performance (run-time speed of the software)

is compiled with preprocessor macro NDEBUG defined. This is often

Why assert() expressions are simply omitted when the program

Check for user input errors, or resource limit failures.

Improper use of assert():
value of top-position.

Avoids making the value of position become negative, or to exceed the
positive non-zero integer. Finally, to shift the throttle, make code that
like position = 0. Also, remember to assign top-position to some
with the throttle, make sure you initialized it by executing a statement
Kindly, please be nice and follow these tricky rules. By U do anything

Dear Programmer,

probably won't bother; something like this:

They, you might write comments or other documentation (but you

int position;
int top-position;

C/C++ code you write is

An old-fashioned style for implementing a „throttle”": The ONLY
Nothing protects them from errant access.

Disadvantages: position and top-position are global variables:

```c
{  
  position = 0;  
  (0 > position)  
  top = position;  
  (position < top)  
  position = position + x;  
}

void shut(){  
  {  
    position = 0;  
    x = position;  
  }  
  (x int) = (int x)  
  void initialize()  
  A small improvement is to code "throttle maintenance functions"
```
multiple throttles.

It is very clumsy if you want to upgrade your software to have functions used for different program objects. The names of the functions will clash with initialization and other...
be negative.

that puts the throttle in an illegal state. The position should never

Notice nothing prevents an erratic programmer from writing code

```c
struct throttle {
  int position;
};

int top;
}
```

In a header file, declare the throttle type as a structure:

Here's a way to overcome the 3rd disadvantage:
class Throttle

    protected:

        int top;
        int position;

    private:

        void shutdown(int x);

    public:

        shutdown();

};

In the throttle.h header file, declare the Throttle type.

Here's the object oriented way:
{ position = 0; top = 1; }

throttle::throttle()
{
    position = position
    (0 > position) ?
        position = top;
    (top < position) ?
        position = position + x;
}

void throttle::shift(int x)

right things:

In the throttle:::cpp implementation file, you assert the implementer
class{
outside the body of a function member belonging to the throttle
REUSE to compile a private member access like mytr.top = 9;
The private member protection rules of C++ make the compiler
......
mytr.mytr(1); // Now mytr is ON.
because the default constructor was called. // mytr is a properly initialized throttle
     throttle mytr;
    inside some functions://
}
......
#include "throttle.h"
Files whose code defines and uses a throttle can then be written:
that the programmer coded the throttle class had designed.

C++ and Java ENFORCE the rules for working with a throttle
(similar features). All functions to access and manipulate it in one place. (Java has
encapsulate everything variables to store the throttle's state plus
Object Oriented Programming features of C++ to
What we surveyed, and DOS details in Chap. 2, is how to use