Function Documentation in ENSF 337

Principles of Function Documentation

- The documentation standards for functions in ENSF 337:
  - REQUIRES
    - Is a precondition that states what must be true before function runs.
  - PROMISES
    - Is one or more statements that describes what the function does.
  - See the following example:

Principles of Function Documentation

- Example : For a function to compute the root of a number:

```
double mysqrt (double x);
// REQUIRES: x>=0
// PROMISES: Return value is square root of x
```

  - The precondition, x >= 0 indicates that, it is the responsibility of the caller of the function to make sure argument x is greater than or equal zero.
  - In other words, if caller doesn't satisfy this requirement, unexpected results may happen.
- A function may need a substantial amount of code to prevent faulty things to happen.
  - We should always design a function in a clever way to minimize the possible errors.
  - If doable those preventive logics should be addressed in the PROMISES part of the.

Examples

Examples

```
double ratio (int a, int b);
/*  PROMISSES
 *      If b == 0, gives an error message and terminates the program
 *      Otherwise: Return value is the ratio of a over b.
 */

double average(const int *a, int size_a);
/*  REQUIRES
 *      size_a > 0.
 *      Elements a[0], a[1], ..., a[size_a - 1] exist.
 *  PROMISES
*       Return value is average of a[0], a[1], ... a[size_a - 1].
*/
```

Error Checking

- In C there are different ways to check program errors:
  - Using *if …else* statement
  - Writing functions to check the errors
  - Using ***assert*** function-like macro
- C++ supports the exception handing concept, but it is not available to C.

What is Assert?

- **Assert** is a preprocessor macro.
- If the argument of **assert** is true, the program aborts and an error message will be displayed.

```
#include <assert.h>
double ratio (int a, int b)
{
    assert (b != 0);
    return (a / double(b));
}
```

- Using **assert** is not is not the best method for end-product error-checking and giving error messages to the end-users.
- However:
  - It helps with a “fail fast” development strategy
  - Normally, it is used for debugging version of the program
    - The final release of the program needs more user friendly method that provides sufficient and meaningful information about the error.