Objectives/Terms

- Default Arguments - How to implement and use.
- Use of `assert`
- Debugging with `stubs` and `drivers`.

Default Arguments
Consider the following prototype in a program:
```c
double foo(int first=1, int middle=5, int last=10);
```
The header would appear as we have already discussed:
```c
double foo(int f, int m, int l) {……}
```
but, the presence of the assignments in the prototype allow for `f`, `m`, and `l` to be given values automatically, if in the call their corresponding arguments are missing.
```cpp
#include <iostream>
using namespace std;

int vol(int=1, int=1, int=1); // default args of 1

int main()
{
    cout << vol(2,3,5) << endl // prints 30
         << vol(2,3) << endl  // prints 6
         << vol(2) << endl    // prints 2
         << vol() << endl;    // prints 1
    return 0;
}

int vol(int h, int w, int l)
{
    return h*w*l;
}
```

---

**Additional Comments**

- In the previous example, variable names could be used in the prototype; e.g.,
  ```cpp
  int vol(int x=1, int y=1, int z=1);
  ```

- If variable names are used, they do not need to be the same as those in the definition header.

- Omitted variables in a call to the function must be from the right; i.e., once an argument is omitted, all arguments to the right must be omitted. **No comma should precede an omitted variable.**

---

**Use of assert <cassert>**

Functions often have stated preconditions on their parameters. Example:
```cpp
int gcd(int a, int b){
    // compute the greatest common
    // divisor of a and b.
    // Pre-cond: a>0 and b>0
    How can you assure that a and b are positive before proceeding with the computation?
    assert(a>0 && b>0);
```
assert (cont.)

The call to assert will test the condition specified within the ( ), and if it is not true, an exit will be performed with an error message.
In order to use assert, you need to include the proper header file in your program. Use

```c
#include <cassert>
```

---

Stubs and Drivers

Modularization of a program has been mentioned often. You will design functions to carry out the individual tasks of a larger program. As functions are designed, they need to be tested, and you should test as you go.
Since an entire program is not available to test functions, you need stubs and drivers.

---

Stubs

Suppose you wish to test a function that makes calls to one or more functions that have not yet been developed. For each one you can create a simple function that can be called and then return.
The function called may not be developed to the point that it completely performs its task. You may want it to return a value or do something much less that its specifications call for.
Such functions are called stubs.
Drivers

On the other hand, you may want to test a specific function module as part of an overall task. In order to do this, you need a function to call it, which may not yet be developed.
Just as with stubs, you can write a minimal program to call your module. This is a driver program.

Debugging

Drivers and stubs can be useful tools in debugging a program as you develop it incrementally.