Function Libraries

CS 150 – C++ Programming I
Lecture 8

The Algorithms & While Loops

- The ancient algorithms we'll implement
  - Euclid's Algorithm: greatest common divisor *(330 BC)*
  - Newton's Square Root (*actually Hammurabi 1800 BC*)
  - Find a better approximation for PI than 355/113
- Each needs an indefinite loop
  - `while (condition) { … }  // preferred`
  - `do { . . . } while(condition); // seldom`
- Each algorithm uses a limit loop
  - Reduce a number to zero, successive approximations, bisection, and non-convergence tests

An Ancient Algorithms Library

- Function call: compiler needs to know argument & return type
  - Define function before use; smallest programs
  - Prototype (declare) before use; small programs
  - Separately compile the functions and link together
  - Last is most flexible; we'll do that from now on
- Exercise: open *algolib* folder
  - *algo.h* - interface; prototypes & documentation
  - *algo.cpp* - implementation; source code
  - *tester.cpp* - client or testing file
  - *makefile* - the project file

The Interface: Header Files

- Header guards prevent multiple inclusions
  -
```
#ifndef ALGO_H
#define ALGO_H
...
#endif
```
- Header files may contain:
  - Documentation, prototypes, constant definitions, class or structure definitions, templates, global variable declarations (extern)
- They may not contain:
  - Function or variable definitions or `using namespace std;`

Namespaces

- Exercise: add *gcd()*, *sqrt()*, and *pi()* to header
  - `int gcd(int a, int b);`
  - `double sqrt(double n);`
  - `void pi(unsigned& n, unsigned& d);`
- Problem? Now have two *sqrt()* functions
  - Called a name clash or name collision
- Solution? Put your functions in the *aa* namespace
  - Need in both the header and implementation
  - ```
    namespace aa {
        // prototypes here
    };
    ```
    ```

The Client File

- The client calls or tests the functions
  - In the Course Reader you learned to do this manually, and by using the CS 150 automated testing framework
- The CS 150 framework testing process
  - `#include` the header for the functions to be tested
  - Place testing code between `beginFunctionTest`, and `endFunctionTest`
  - Use `assertEquals` varieties (see `tester.cpp`)
- Exercise: type `make tester` (linker errors only)

The Implementation & Stubs

- Implementation goes in `.cpp` file
- Always start with stubs (or skeletons)
  - 1. Add `#include "header"` to client (note double-quotes)
  - 2. Add any other headers needed to implement
  - 3. May add `using namespace std;`
  - 4. Copy prototypes & namespace from header file
  - 5. Remove semicolons, add body and return statements
- Still doesn't build; need to create a `makefile`

Building Your Program

- Cross-platform builder program is called make
- Makefile: 3 parts: variables, rules, actions
  - 1. MACROs/variables: EXE=, OBJS=
  - 2. Target, dependency: $(EXE): $(OBJS)
  - 3. <tab> $(CXX) $(CXXFLAGS) $(OBJS) -o $(EXE)
  - 4. Pseudo-target:

```
test: $(EXE)
	./$(EXE) -t  # must start with a tab
```

- Exercise: make and make test

Implementing gcd

- Developed by Euclid about 300 BC (*Wikipedia Link*)
  - Essential insight: divisor of larger and smaller is same as divisor of smaller and (larger - smaller)
  - When we reduce smaller to `0`, have answer
  - Must repeat, so a loop; reduce to `0` so use a limit loop
- Pseudocode for the algorithm `gcd(a, b)`
  - *While b is not 0*
    Let t = b, b = a % b, a = t
    GCD is a
- Exercise: implement and test `gcd(a, b)`

Implementing Newton's Square Root

- Attributed to Isaac Newton, but actually recorded in base-60 cuneiform from tablets in the Hammurabi dynasty (around 1700 BC).
- An approximation algorithm

```
– Make a new guess as to the root of n
  Loop
     Set old-guess to new-guess
     Set new-guess to ((n/old-guess) + old-guess) / 2
  Until difference between guesses is < epsilon
```

- Exercise: implement and test

Implementing a pi Approximation

- A rational approximation for PI
  - `19/6` - Ahmes Papyrus 1650 BC, 25/8 - Babylonian
  - `223/71` < PI < `220/70` - Archimedes 150 BC
  - `377/120` - Ptolemy 1st Century
  - `355/113` - Tsu Ch'ung-chih - 5th Century
- Our Goal – find a better approximation than Tsu
  - a) Make an initial approximation
  - b) While it isn't close enough, try to get closer
  - c) Use `acos(-1.0)` as our test oracle `PI`
  - d) Use `abs(TSU - PI)` as `EPSILON`

Documentation

- Use DOXYGEN (Javadoc) style function comments
  - Place in your header file
  - Add file comment with `@file`, namespace comment
  - 1. `@param` tags for each parameter
  - 2. `@return` tags describe what is returned
  - 3. `@code{.cpp}` … `@endcode` for examples
- Exercise: document and shoot a screenshot