```
C++ Syntax (Cheat Sheet)
------------------------
Terms in <> are tokens which describe generically what goes in there.  All
other terms are literally themselves.
EXAMPLE: <name> = <expression>       represents
         x = 5                       or
         name = "John" + "Stewman"
Also, a <statement> can be a function call, an assignment statement, an
if or if-else statement, a while or do-while loop, a switch statement, a for loop, etc.

// comment
/* another comment */

bool int float char long short string      //  types
'a' 124  -25.0 1.33e5 "hello" true         // literal constants (values)

+ - / * %                                  // arithmetic operations
< <= > >= == !=                            // comparison operations
&& || !                                    // boolean operations

<statement>; <statement>; ...              // statements

#include < <library_name> >             // include file directive
#include "stuff.h"                      // include header file directive

using namespace <name>                     // namespace directive

enum <name> { <value_name_list> };         // enumeration definition

<name>, <name>, <name>, ...                // value name list

#define <name> <value>        // defined constant compiler directive

const <type> <name> = <value>;    // constant definition

<type> <name>, <name>, ... ;      // object definition(s)

<type> <name> = <value>;         // object definition and initialization
<type> <name> ( <value> );       // another object defn & initialization

<element_type> <name> [ <number_of_elements> ];     // array declaration

<type> <name> ( <formal_parameter_list> );     // function prototype

<type> <name>, <type> <name>, ...       // formal parameter list

<type> <name> ( <formal_parameter_list> )      // function definition
{
   <definitions>
   <statements>
}

<function_name> ( <actual_parameters> );     // call to void function

// call to (or use of) function which returns a value
<name> = <function_name> ( <actual_parameters> );

<name>, <name>, ...                      // actual parameters

{                                    // block -- can replace ANY statement
    <definitions>                    // has its own LOCAL SCOPE
    <statements>
}
```

```
<name> = <expression>;                 // assignment statement

cout << fixed  setprecision(2) endl    // output stream related terms

cin >>                                 // input stream related terms

ifstream <name>("filename");           // declare and open input file stream
ofstream <name>("filename");           // declare and open output file stream

<stream_name>.open( char *<fname> )    // open file with name in char array <fname>
<stream_name>.close()                  // close a stream

<sting_name>.c_str()                   // convert string object to char array

if ( <boolean_expression> )            // if statement
    <statement>;

if ( <boolean_expression> )            // if-else statement
    <statement>;
else
    <statement>;

switch ( <expression> )                // switch statement
{    case <constant> :
          <statements>
          break;
     case <constant> :
          <statements
          break;
    default :
         <statements>
}

while ( <boolean_expression> )         // while loop
    <statement>;

do {                                   // do-while loop
    <statements>
} while ( <boolean_expression> );

for ( <initialization>; <continuation_expression>; <increment_statement>)
     <statement>;

class <class_name>           // class header (prototype)
{
public:
    <function_prototypes>
protected:
    <function_prototypes>
private:
    <function_prototypes>
    <data_attributes>
};

<class_name>::<function_name> ( <parameter_list> ) // member function
{                                                  // implementation
    <declarations>
    <statements>
}

<object_name>.<function_name>(<actual_parameters>); // member function call
```

```
struct <structure_name> {            // structure definition
   <type> <name>, <name>, ... ;      // field definition(s)
   <type> <name>, <name>, ... ;
}

typedef <type> <name>;                                // definition of new type name
typedef <element_type> name <dimensions>;     // definition of new array type
<dimensions> = [ <int_value> ] [ <int_value> ] ...

template < typename T >                   // template header

<type> *<name>;                           // pointer declaration
&<name>                                   // address of <name>
*<ptr>                                    // data <ptr> points to
<ptr>-><member>                           // access to member through pointer
(*<ptr>).<member>                         // access to member of object
<object_name>.<member>                    // ditto

new <type> or new <type>[ <size> ]        // operator to dynamically allocate memory
delete <ptr> or detlete [] <ptr>          // return memory to dynamic memory store
```