```
1  // Demonstrate various high-level syntax for the C language
2  // ECE2036
3  // George F. Riley, Georgia Tech, Fall 2012
4
5  // Comments in C and C++ can be entered using the "slash-slash" double
6  // character as done here.  In this type of comment, the comment is
7  // only to the end of the current line.
8
9  /* Another way to include comments is to open a comment block with the
10    slash-star double character, and terminate it with the star-slash
11    double character as done here
12 */
13
14 // Nearly all C and C++ programs start with "includes" which tell the
15 // compiler to find some other file (in this example it is stdio.h)
16 // and insert all of the text found in that file into this compilation
17 // unit.  Normally, the contents of the ".h" (header) file only contain
18 // function prototypes, not the actual implementation.  More on this
19 // later.
20
21 #include <stdio.h>
22
23 // In C and C++ variables must be "declared" before they can be used.
24 // Declaring a variable is simple, just say the "type" of the variable
25 // followed by the variable name. Below defines three variables each
26 // of a different type.  The three types used below (int, char, double)
27 // are part of the "built-in" types defined by the language.  There
28 // are many built-in types, but for now we will concentrate on just
29 // these three.  There are also ways to define new types.
30 // In the example below, we assigned an initial value to intVariable
31 // and left the others "uninitialized"
32
33 int    intVariable = 10; // Define an integer variable (32 or 64 bit)
34 char   byteVariable;     // Define a singe 8-bit byte variable
35 double floatVariable;    // Define an 8-byte "floating point" variable
36
37 // A "function prototype" is a way to tell the compiler of the existence
38 // of a particular "function" (some languages call this a "subroutine")
39 // without actually providing the implementation of the function.
40 // C and C++ differ on the requirements for this.  The C language
41 // allows functions to be called without prototypes, but C++ does not.
42 // In 2036 we will always use the C++ compiler, so we need to
43 // include prototypes. Actually, this is not completely true as will
44 // be explained later.
45
46 // Below we provide a function prototype for a function called "func1"
47 // that accepts two arguments (an integer and a double) and returns
48 // a computed value of type double.  Note the trailing semicolon below.
49
50 double func1(int arg1, double arg2);
51
52 // Below is the implementation of a function called func2 that accepts
53 // two arguments and returns an integer. Notice the difference here
54 // as compared to func1; here the function is actually implemented.
55 // Here is a case where a prototype is not necessary, as the function
56 // is not actually called prior to the function being defined.
```

Program csyntax.cc

```
57  // Note the use of the "open curly" and "close curly", as well as
58  // "return" statement.  "return" is a reserved word in C and C++ and
59  // cannot be used for something else.
60
61  int func2(int arg1, int arg2)
62  { //  Compute arg1 times arg2 and return the product
63    return arg1 * arg2;
64  }
65
66  // Below defines and implements "func3", which illustrates the use
67  // of a "pointer" in C/C++.  A pointer is a variable just like any
68  // other C/C++ variable, but the difference is the the VALUE of the
69  // pointer is the address of some other variable somewhere in memory.
70  // func3 also illustrates "de-referencing" the variable using the "star"
71  // operator.  Finally, func3 illustrates the "void" return type, which
72  // indicates the function does not actually compute a return value.
73
74  void func3(int* pInt1, // pInt1 is a "pointer"
75             int  int2)  // int2 is a normal integer
76  { // The open-curly starts the implementation of the func3 function
77    // The next line says to take the value found in int2 and store it
78    // in whatever address is found in pInt1. Pointers are used extensively
79    // in C and C++
80    *pInt1 = int2;
81  }
82
83  // C and C++ allow "defined" constants, as illustrated below.
84  // K2Length is defined as 20 and will be used later in the main.
85
86  #define K2Length 20
87
88  // Below illustrates the definition of a C/C++ "structure".
89  // A structure is a way to state that a single variable has
90  // multiple "sub-variables".  In the example below we have
91  // a structure called "myStruct" with subvariables a, b, and c.
92  // It is important to note that the typedef DOES NOT define a variable.
93  // It simply defines a type (analagous to int, char, etc); variables
94  // of that type must later be declared as is shown below in the
95  // main function
96
97  typedef struct
98  { // myStruct has three subvariables (or components).
99    // Note the components can be different types, but don't have to be
100   int    a;
101   char   b;
102   double c;
103 } myStruct;
104
105 // All C and C++ programs start with a function called "main" that
106 // returns an integer (generally ignored) and accepts two arguments.
107 // The two arguments "argc" and "argv" are the count of the command
108 // line arguments entered when the program was started, and an array
109 // containing the actual arguments.  We will discuss this in more
110 // detail later in the class.
111
112 int main(int argc, char** argv)
```

Program csyntax.cc (continued)

```
{
  // Functions often declare "local variables" that exist only while
  // the function is being executed.  Here we define several
  // local variables including an integer array.
  // Note the open-close square braces indicating
  // an array variable.  The size of the array (using this syntax) must
  // be known at compile time.  The array will be accessed a few
  // lines later.
  int i = 5;  // declare local variable i and initialize to
  int j;      // declare local varialbe j and leave uninitialized.
  int k[10];  // k is an array of 10 integer values, uninitialized
  int k2[K2Length];  // k2 is an array of 20 integer values, uninitialized
  double d = 1.5;    // d is a single 8-byte floating point value.
  // Declare a variable of type myStruct.
  // THis is referenced later.
  myStruct mySt;

  // Initialize the sub-variables in mySt
  mySt.a = 1;
  mySt.b = 'C';  // Note the character constant with single quotes
  mySt.c = 2.0;

  // Both C and C++ make extensive use of "for loops".
  // In this example, we initialize the k array to known values.
  // We also use a local variable "i1" as the loop variable, which
  // has a lifetime only within the loop.  This is common and good practice.
  // Finally note the use of the "++" operator, which in this case
  // essentially says to set variable i1 to i1 + 1.
  for (int i1 = 0; i1 < 10; i1++)
    { // the open curly brace indicates the start of the code repeated
      // by the for loop
      k[i1] = i1;  // Note the array reference with square brackets
      // printf is one way to print things to the console window.
      // In C++ we generally use  a different way using "cout"
      // discussed later in class.
      printf("Initialized k[%d] to %d\n", k[i1], i1);
    }
  // Illustrate another for loop iterating over k2; this loop is slightly
  // different, but much better (Why?)
  for (int i2 = 0; i2 < K2Length; i2++)
    { // the open curly brace indicates the start of the code repeated
      // by the for loop
      k2[i2] = i2;
      printf("Initialized k2[%d] to %d\n", k2[i2], i2);
    }

  // Note the below line of code won't comple. Why?
  // j = i2;

  // Illustrate "calling" a function, in this case func3.  Note use of the
  // "address of" operator "&".  The value of the first argument to func3
  // is not the value "j" but the address in memory of j.
  func3(&j, i);
  printf("j is %d\n", j); // What do you think is printed here.

  // Illustrate calling a function in an expression.
```

```
169     i = i + func2(10, 20);
170     printf("i is %d\n", i); // What do you think is printed here.
171
172     //Illustrate calling func1 even though func1 has not yet been
173     // implemented.
174     d = k2[1] + func1(20, 10);
175     printf("d is %f\n", d); // What do you think is printed here.
176
177     // Since the "main" function is declared to return an integer
178     // we return 0.  For main this is often omitted.
179     return 0;
180     // Notice you can legally put more code after a return statement
181     // but of course that code is never executed
182     printf("Should not be printed\n");
183   }
184
185   // Now provide the implementation of func1
186   double func1(int arg1, double arg2)
187   { // just return the quotient
188     return arg1 / arg2;
189   }
```

Program csyntax.cc (continued)