Concept: parts of a function

1. What is the signature of this function?
   ```
   double
   almostSquare(int x,double y)
   {
       return (x - 1) * (y + 1);
   }
   ```
   (a) `almostSquare(int x,double y)`
   (b) `double almostSquare(int x,double y)`
   (c) `return (x - 1) * (y + 1);`
   (d) `double almostSquare(int x,int y)`

2. What is the signature of this function?
   ```
   int f(void) { return 1; }
   ```
   (a) it has no signature
   (b) `int f(int)`
   (c) return 1;
   (d) `int f(void)`
   (e) `void f(int)`

3. What is the body of this function?
   ```
   double
   int almostSquare(int x,double y)
   {
       return (x - 1) * (y + 1);
   }
   ```
   (a) `return (x - 1) * (y + 1);`
   (b) `double almostSquare(int x,int y)`
   (c) `double almostSquare(int,double)`
   (d) `return double * double;`

4. What is the body of this function?
   ```
   int f(int x) { return g(x * 2); }
   ```
   (a) `return int`
   (b) `int f(g(x * 2))`
   (c) `return g(x * 2);`
   (d) `return function call`

Concept: recognizing function definitions

5. This is what?
   ```
   int square(int x) { return x * x; }
   ```
   (a) a function call
   (b) a function definition
   (c) a function body
6. What is this?
   
   ```c
   int zqr(int x) {return (x+1) * (x-1);}
   ```

   (a) a complete program  
   (b) a function signature  
   (c) a function call  
   (d) a function definition

**Concept:** recognizing function calls

7. What is this?
   
   ```c
   square(x);
   ```

   (a) A function call  
   (b) A function program  
   (c) A function definition  
   (d) A function body

8. What is this?
   
   ```c
   almostSquare(a);
   ```

   (a) a function signature  
   (b) a function definition  
   (c) a function call  
   (d) a complete program

**Concept:** recognizing errors in function definitions

9. Does this function definition exhibit good style?
   
   ```c
   int smul(x,y) { return atoi(x) * atoi(y); }
   ```

   (a) no, the return type should be `char *`  
   (b) yes, it is correct as written  
   (c) no, there should only be one formal parameter  
   (d) no, the formal parameters are missing types

10. Is this function definition syntactically and logically correct?
    
    ```c
    int mul(int x,int y)
    return x * y;
    ```

    (a) no, there should only be one formal parameter  
    (b) no, the return statement should be indented  
    (c) no, the function body needs braces  
    (d) yes, it is correct as written

11. Is this function definition syntactically and logically correct?
    
    ```c
    void mul(int x,int y) { return x * y; }
    ```

    (a) no, `void` cannot ever be used as a return type  
    (b) no, the body is not indented under the signature  
    (c) no, the return types do not match  
    (d) yes, it is correct as written

12. Is this function definition syntactically and logically correct?
    
    ```c
    int mul(int x,int y) { return x * y; }
    ```

    (a) no, `int` cannot ever be used as a return type
13. Is this function definition syntactically and logically correct?

```c
int pow(int x int y) { return x * y; }
```

(a) no, there should only be one formal parameter
(b) no, int cannot ever be used as a return type
(c) yes, it is correct as written
(d) no, there should be a comma between formal parameters
(e) no, the body is not indented under the signature

14. Is this function definition syntactically and logically correct?

```c
void return(int x,int y) { print(x * y); }
```

(a) no, there is no return statement
(b) yes, it is correct as written
(c) no, the name of the function is a keyword
(d) no, you cannot print inside a function

15. Is this function definition syntactically and logically correct?

```c
void debug(char *message,int value)
{
    printf(%s%d
",message,value);
    return;
}
```

(a) no, void can never be used as a return type
(b) no, you cannot return nothing
(c) yes, it is correct as written
(d) no, the name of the function is a keyword

**Concept:** matching calls and definitions

16. Does the function call match the function definition?

```c
int square(int x) { return x * x; }
```

...  
```
int result = square();
```

(a) yes
(b) no, there are too few formal parameters
(c) no, there are too many arguments
(d) no, there are too few arguments

17. Does the function call match the function definition?

```c
int square(int x) { return x * x; }
```

...  
```
int result = square(3);
```

(a) no, the function call should pass the variable x
(b) yes
(c) no, there are too many arguments
(d) no, there are too few formal parameters

18. Does the function call match the function definition?
int square(int x) { return x * x; }
...
int a = 3;
int result = square(a);

(a) no, there are too few formal parameters in the definition
(b) no, there are too many arguments in the call
(c) no, the function call should pass a value, not a variable
(d) no, the function call should pass the variable x, not a
(e) yes

19. Does the function call match the function definition?
int square(x) { return x * x; }
...
int result = square(3,7);

(a) no, there are too many arguments in the call
(b) yes
(c) no, there are too many formal parameters in the definition
(d) no, the function call should pass the variable x twice

20. Does the function call match the function definition?
int squish(int a,int b) { return a * (b - 1); }
...
int x = 3;
int y = 7;
int result = Squish(x,y);

(a) no, the function names do not match
(b) no, the function call should pass the variables a and b
(c) no, the function call should pass two values, not variables
(d) yes

21. Does the function call match the function definition?
int squish(int a,int b) { return a * (b - 1); }
...
int x = 3;
int y = 7;
int result = squish(x,y,x * x);

(a) no, the function names do not match
(b) no, there are too many formal parameters
(c) yes
(d) no, there are too many arguments

Concept: identifying arguments

22. Identify the function call arguments in the code below:
int compute(int x,int y)
{
    int a = x + 1,b = y - 1;
    return a * b;
}
...
int result = compute(j + 1,k - 1);

(a) the values 3 and 7
(b) the variables x and y
(c) the variables j and k
23. What are the variables $j$ and $k$ in the code below:

```c
int compute(int x, int y) {
    int a = x + 1, b = y - 1;
    return a * b;
}
...
j = 3; k = 7;
int result = compute(j, k);
```

(a) local variables defined in the body of the `compute` function
(b) the arguments given in a function call
(c) the formal parameters of a function
(d) function names

24. What are the formal parameters of the `compute` function?

```c
int compute(int x, int y) {
    int a = x + 1, b = y - 1;
    return a * b;
}
...
j = 3; k = 7;
int result = compute(j, k);
```

(a) $a$, $b$
(b) $j$, $k$
(c) the compute function has no formal parameters
(d) $x$, $y$

25. What is being bound to the formal parameters? Choose the most precise answer.

```c
int compute(int x, int y) {
    int a = x + 1, b = y - 1;
    return a * b;
}
...
j = 3; k = 7;
int result = compute(j, k);
```

(a) the values of variables $x$ and $y$
(b) the values of variables $j$ and $k$
(c) the variables $x$ and $y$
(d) the variables $j$ and $k$

**Concept:** analyzing code

26. What is the value of $x$ while the function body is being evaluated?

```c
int compute(int x, int y) {
    int a = x + 1, b = y - 1;
    return a * b;
}
...
j = 3; k = 7;
int result = compute(j, k);
```

(a) $j$
(b) $k$
27. What is the value of \( y \) while the function body is being evaluated?

```c
int compute(int x, int y) {
    int a = x + 1, b = y - 1;
    return a * b;
}
```

...  

\( j = 3; k = 7; \)  

```c
int result = compute(j,k);
```

(a) \( k \)
(b) \( y \)
(c) 3
(d) 7

28. What are the arguments, formal parameters, and local variables of the `compute` function?

```c
int compute(int x, int y) {
    int a = x + 1, b = y - 1;
    return a * b;
}
```

...  

```c
int result = compute(3,7);
```

(a) 3 and 7, \( a \) and \( b \), \( x \) and \( y \), respectively  
(b) \( x \) and \( y \), 3 and 7, \( a \) and \( b \), respectively  
(c) the compute function has no local variables  
(d) 3 and 7, \( x \) and \( y \), \( a \) and \( b \), respectively

29. What is printed by this code fragment:

```c
int squish(int x, int y) { (x-1)*(y+1); }
```

...  

```c
printf("%d\n",squish(5,5));
```

(a) 24  
(b) 26  
(c) garbage  
(d) 25  
(e) nothing is printed because of a compiler error

30. What is printed by this code fragment:

```c
int squish(int x, int y) { return (x-1)*(y+1); }
```

...  

```c
int x = squish(5,5);  
printf("%d\n",x);
```

(a) 26  
(b) 25  
(c) garbage  
(d) nothing/incorrect code/ \( x \) is not in scope  
(e) 24

**Concept:** program structure

31. Which of the following statements about the `main` function is true with regards to C programs:

(a) `main` can be defined anywhere in the file  
(b) `main` must be the last function defined
(c) *main* must be the first function defined
(d) a *main* function is optional

**Concept:** running programs

32. Assume you are in the same directory as the C program *go.c*; there are no other files in the directory. To run this program, you would use the command:
   (a) .*/go.c
   (b) gcc -Wall go.c -o go; .*/go
   (c) gcc go
   (d) .*/a.out
   (e) gcc -Wall go.c; .*/go

**Concept:** comments

33. What is the simplest way to comment out the remainder of a line?
   (a) `#ifdef JUNK and #endif`
   (b) `//`
   (c) there is no way to do this
   (d) `/* and */`

34. What is the simplest way to logically remove code from the middle of a line?
   (a) there is no way to do this
   (b) `#ifdef JUNK and #endif`
   (c) `//`
   (d) `/* and */`

35. What is the simplest way to logically remove a section of code that contains no comments?
   (a) `#ifdef JUNK and #endif`
   (b) `/* and */`
   (c) there is no way to do this
   (d) `//`

36. What is the simplest way to logically remove a section of code that contains /* ... */ comments?
   (a) `/* and */`
   (b) `//`
   (c) there is no way to do this
   (d) `#ifdef JUNK and #endif`

37. What is the final value returned by the sequence of function calls? Assume the functions do what their names imply.

   ```plaintext
   x = 5;
x = timesTwo(x);
y = timesTwo(x + x);
z = addOne(y);
   ```

   (a) 11
   (b) 21
   (c) 41
   (d) 31

38. What is the value of *x* after this sequence of function calls? Assume the functions do what their names imply.

   ```plaintext
   x = 5;
x = addOne(timesTwo(timesTwo(x)));
   ```

   (a) 21
   (b) 11
Concept: function pointers

39. Which of the following is a proper long form function pointer definition. Assume the variable defined is to point to the function `int f(int x);`?
   (a) int (*g)(int) = f(0);
   (b) int (*g)(int) = f;
   (c) int (g)(int) = f(0);
   (d) int (g)(int) = f;

40. Which of the following is a proper long form function pointer definition. Assume the variable defined is to point to the function `int *f(double x);`?
   (a) int (**g)(double) = f;
   (b) int *(*g)(double) = f(1.2);
   (c) int *(g)(double) = f;
   (d) int (**g)(double) = f;
   (e) int (**g)(double) = f(3.2);
   (f) int (**g)(double) = f(2.3);

41. Which of the following is a proper typedef for a short form function pointer definition? Assume pointers defined using the new type `FPtr` are to point to the function `int f(int x);`?
   (a) typedef int (*FPtr)(int);
   (b) typedef int (*)(int) FPtr;
   (c) typedef (int *)(int) FPtr;
   (d) typedef FPtr = int (*)(int);
   (e) typedef (int *) FPtr(int);
   (f) typedef FPtr = (int *)(int);