Concept: *vocabulary*

1. With respect to scope, another name for ‘visible’ is:
   (a) out of scope
   (b) in scope
2. With respect to scope, another name for ‘inaccessible’ is:
   (a) in scope
   (b) out of scope
3. With respect to scope, another name for ‘accessible’ is:
   (a) in scope
   (b) out of scope
4. T or F: Non-local variables, with respect to a particular scope, are those variables defined in that scope.
5. T or F: Local variables, with respect to a particular scope, are those variables defined in that scope.

Concept: *enclosing scopes*

6. T or F: Variables in an *enclosing* scope are in scope.
7. T or F: Variables in an *enclosed* scope are in scope.
8. T or F: The global scope is enclosed by all other scopes.
9. T or F: Every other scope encloses the global scope.
10. T or F: Suppose in the body of function $f$, an if block defines a variable. The scope of if block encloses the scope of $f$.
11. T or F: Suppose in the body of function $g$, an if block defines a variable. The scope of $g$ is enclosed by the scope of if block.

Concept: *scopes and definitions*

12. T or F: The set of names defined in the scope of a function body includes the local variables.
13. T or F: The set of names defined in the scope of a function body includes the non-local variables.
14. With respect to a function body, the function name belongs to the:
   (a) set of local variables
   (b) set of non-local variables
   (c) set of variables that are out of scope
15. With respect to a function body, the function name is defined in:
   (a) the scope of the function body
   (b) the scope enclosing the scope that holds the function definition
   (c) the scope that holds the function definition
16. T or F: The set of names defined in the scope of a function body includes the name of the function.
17. With respect to a function body, the formal parameters of the function belong to the:
   (a) set of non-local variables
   (b) the global scope
   (c) set of variables that are out of scope
   (d) same scope as the local variables
18. With respect to a function body, the formal parameters of the function are defined in the:
(a) the global scope
(b) the scope enclosing the scope that holds the function definition
(c) the scope of the function body
(d) the scope that holds the function definition

19. T or F: The set of names defined in the scope of a function body includes the formal parameters.

Concept: modules and makefiles

20. Suppose module a.c has these includes:

```c
#include <stdio.h>
#include "alpha.h"
#include <stdlib.h>
#include "beta.h"
```

What would the dependencies of a.o in a makefile be? Choose the best answer.

(a) a.c  
(b) stdio.h, alpha.h, stdlib.h, beta.h, a.c  
(c) alpha.h, stdlib.h, beta.h  
(d) alpha.h, beta.h, a.c  
(e) stdio.h, alpha.h, stdlib.h, beta.h  
(f) alpha.h, beta.h

21. Suppose module a.c has these includes:

```c
#include <stdio.h>
#include "alpha.h"
#include <stdlib.h>
#include "beta.h"
```

What would the action of a.o in a makefile be? Choose the best answer.

(a) gcc -Wall -c a.c  
(b) gcc -Wall -c a.c alpha.h beta.h  
(c) gcc -Wall a.c  
(d) gcc -Wall a.c alpha.h beta.h

22. Using the guidelines for building makefiles found in the modules activity, how many rules would there be for a program that has modules, alpha.c, beta.c, and gamma.c? Assume only gamma.c has a main function defined.

(a) 5, including the clean rule  
(b) 4, including the clean rule  
(c) 5, not including the clean rule  
(d) 3, not including the clean rule  
(e) 6, not including the clean rule

Concept: undefined variables

23. Is the following program legal?

```c
int main(int argc, char **argv)
{ int y = x + 1; return y; }
```

(a) no, x does not exist  
(b) yes, this is legal  
(c) no, y cannot be returned  
(d) no, stdio.h is not included

24. What happens when this program is compiled and run?

```c
#include <stdio.h>
int main(int argc, char **argv)
{ printf("x is %d\n",x); return 0; }
```
25. What happens when this program is compiled and run?

```c
#include <stdio.h>
int x = 3;
int main(int argc, char **argv)
{
    printf("x is %d\n", x); return 0;
}
```

(a) x is x is printed
(b) a compile error: x isn't defined
(c) a compile error: stdlib.h isn't included
(d) x is 0 is printed

26. What happens when this program is compiled and run?

```c
int main(int argc, char **argv)
{
    return x;
}
```

(a) a compile warning concerning variable definitions after main
(b) a compile error concerning x being undefined
(c) the program compiles and runs without error
(d) a compile warning concerning x being implicitly defined

27. What happens when this program is compiled and run?

```c
extern int x;
int main(int argc, char **argv)
{
    return x;
}
```

(a) a compile warning concerning x being implicitly defined
(b) a compile error concerning x being undefined
(c) a compile warning concerning variable definitions after main
(d) the program compiles and runs without error

28. What happens when this program is compiled and run?

```c
#include <stdio.h>
int squarePlus(int x)
{
    int y = x + 1;
    return y * y;
}
int main(int argc, char **argv)
{
    int z = squarePlus(4);
    printf("z is %d\n", z);
    return 0;
}
```

(a) y is None is printed
(b) y is 4 is printed
(c) y is 5 is printed
(d) a compiler error concerning y

29. What happens when this program is compiled and run?

```c
#include <stdio.h>
int main(int argc, char **argv)
{
    int z = squarePlus(4);
    printf("z is %d\n", z);
    return 0;
}
int squarePlus(int x)
{
    int y = x + 1;
    return y * y;
}
```
30. What happens when this program is compiled and run?

```c
#include <stdio.h>
int squarePlus(int);
int main(int argc, char **argv)
{ int z = squarePlus(3);
  printf("z is %d\n", z);
  return 0;
}
int squarePlus(int x)
{ int y = x + 1; return y * y; }
```

(a) a compiler error concerning `squarePlus`
(b) `z is 16` is printed
(c) a compiler error concerning `squarePlus`
(d) a compiler warning concerning `squarePlus`

Concept: global variables

31. What happens when this program is compiled and run?

```c
#include <stdio.h>
int z = 2031;
int main(int argc, char **argv)
{ int z = 1831;
  printf("%d\n", z);
  return 0;
}
```

(a) 3862
(b) a compiler error
(c) 2031
(d) 1831

32. Ignoring any warnings, what happens when this program is compiled and run?

```c
#include <stdio.h>
int z = 2031;
void b(void) { z = 100; }
void a(void) { z = 12; }
int main(int argc, char **argv)
{ a(); b();
  printf("%d\n", z);
  return 0;
}
```

(a) 2031
(b) 12
(c) a compilation error
(d) 100

33. Ignoring any warnings, what happens when this program is compiled and run?

```c
#include <stdio.h>
int z = 2031;
void b(void) { int z = 100; }
void a(void) { z = 12; }
int main(int argc, char **argv)
{ a(); b();
  printf("%d\n", z);
  return 0;
}
```
34. Ignoring any warnings, what happens when this program is compiled and run?

```c
#include <stdio.h>
int z = 2031;
void b(void) { z = 100; }
void a(void) { int z = 12; }
int main(int argc,char **argv)
{ a(); b();
  printf("%d\n",z);
  return 0; }
```

(a) 100
(b) 2031
(c) 12
(d) a compilation error

Concept: parameter names

35. Which function call (if any) causes an error?

```c
void apples(int count) {
  printf("you have \%d apples\n",count);
}

void oranges(int amount) {
  printf("you have \%d oranges\n",amount);
}

... int count = 10;
apples(count);
oranges(count);
```

(a) neither call
(b) the call to oranges
(c) the call to apples
(d) both calls

36. Which function call (if any) causes an error?

```c
void apples(int count,int amount) {
  printf("you have \%d apples\n",count + amount);
}

void oranges(int amount,int count) {
  printf("you have \%d apples\n",amount + count);
}

... int count = 10;
int amount = 5;
apples(count,amount);
oranges(count,amount);
```

(a) both calls
(b) neither call
(c) the call to apples
(d) the call to oranges
Concept: two scope levels, inner scope

37. The names visible in scope 2 are:

```c
int z = 0; //scope 1
int f(int x) {
    int a = x - 1; //scope 2
    int b = x + 1;
    return b * b - a * a;
}
```

(a) a, b
(b) a, b, f, x
(c) a, b, x
(d) a, b, f, x, z

38. The names defined in scope 2 are:

```c
int z = 0; //scope 1
int f(int x) {
    int a = x - 1; //scope 2
    int b = x + 1;
    return b * b - a * a;
}
```

(a) a, b
(b) a, b, f, x
(c) a, b, x
(d) a, b, f, x, z

Concept: two scope levels, outer scope

39. The names defined in scope 1 are:

```c
int z = 0; //scope 1
int f(int x) {
    int a = x - 1; //scope 2
    int b = x + 1;
    return b * b - a * a;
}
```

(a) z
(b) z, f, x
(c) z, f
(d) z, f, x, a, b

40. The names visible in scope 1 are:

```c
int z = 0; //scope 1
int f(int x) {
    int a = x - 1; //scope 2
    int b = x + 1;
    return b * b - a * a;
}
```

(a) z, f, x, a, b
(b) z
(c) z, f
(d) z, f, x

41. Variable x is visible in which scopes?
int z = 0;  //scope 1
int f(int x) {
    int a = x - 1;  //scope 2
    int b = x + 1;
    return b * b - a * a;
}

(a) scope 1
(b) scope 2
(c) scopes 1 and 2

42. The name \textit{f} is visible in which scopes?

(a) scope 2
(b) scope 1
(c) scopes 1 and 2

Concept: \textit{three scope levels, inner scope}

43. The names defined in scope 3 are:

(a) \(x\)
(b) \(m, r\)
(c) \(m, r, z\)
(d) \(m\)

44. The names visible in scope 3 are:

(a) \(m, r, x\)
(b) all of them
(c) \(m, r, a, b\)
(d) \(m, r\)

45. The name \(f\) is visible in which scopes?
int z = 0;       //scope 1
int f(int x) {
    int a = x - 1;   //scope 2
    int b = x + 1;
    if (z) {
        int r = a + 1; //scope 3
        int m = r * x;
        return m * m;
    }
    return b * b - a * a;
}

(a) scope 1
(b) scopes 1 and 2
(c) scopes 1, 2, and 3
(d) scopes 2 and 3

Concept: three scope levels, middle scope

46. The names visible in scope 2 are:

(a) all of them, except r
(b) r, m
(c) all of them
(d) all of them, except r and m

47. The names defined in scope 2 are:

(a) all of them, except r and m
(b) a, b
(c) x, a, b
(d) x, a, b, r, m

Concept: three scope levels, outer scope

48. The names defined in scope 1 are:
int z = 0;          //scope 1
int f(int x) {
    int a = x - 1;   //scope 2
    int b = x + 1;
    if (z) {
        int r = a + 1; //scope 3
        int m = r * x;
        return m * m;
    }
    return b * b - a * a;
}

(a) z, f
(b) z, f, x, a, b
(c) z
(d) z, f, x

49. The names visible in scope 1 are:

int z = 0;          //scope 1
int f(int x) {
    int a = x - 1;   //scope 2
    int b = x + 1;
    if (z) {
        int r = a + 1; //scope 3
        int m = r * x;
        return m * m;
    }
    return b * b - a * a;
}

(a) all of them
(b) z, f, x
(c) z
(d) z, f

50. The names defined in scope 2a are:

int z = 0;          //scope 1
int f(int x) {
    int a = x - 1;   //scope 2a
    return a * a;
}
int g(int y) {
    int b = y + 1;   //scope 2b
    return b * b;
}

(a) a, b, x, y
(b) a, x
(c) a
(d) a, b

51. The names defined in scope 2b are:

int z = 0;          //scope 1
int f(int x) {
    int a = x - 1;   //scope 2a
    return a * a;
}
int g(int y) {
    int b = y + 1;   //scope 2b
    return b * b;
}
(a) $b$
(b) $a, b, x, y$
(c) $b, g, y$
(d) $b, y$

52. The names visible in scope 2a are:

```java
int z = 0; //scope 1
int f(int x) {
    int a = x - 1; //scope 2a
    return a * a;
}
int g(int y) {
    int b = y + 1; //scope 2b
    return b * b;
}
```

(a) all of them
(b) $a, f, x, z$
(c) $a, x$
(d) $a, f, g, x, z$

53. The names visible in scope 2b are:

```java
int z = 0; //scope 1
int f(int x) {
    int a = x - 1; //scope 2a
    return a * a;
}
int g(int y) {
    int b = y + 1; //scope 2b
    return b * b;
}
```

(a) $b, y$
(b) $b$
(c) all of them
(d) $b, f, g, y, z$

Concept: parallel scope levels, outer scopes

54. The names defined in scope 1 are:

```java
int z = 0; //scope 1
int f(int x) {
    int a = x - 1; //scope 2a
    return a * a;
}
int g(int z) {
    int b = x + 1; //scope 2b
    return b * b;
}
```

(a) $z$
(b) $z, x, g, f, b, a$
(c) $z, x, g, f$
(d) $z, g, f$

55. The names visible in scope 1 are:
int z = 0;                 //scope 1
int f(int x) {
    int a = x - 1;        //scope 2a
    return a * a;
}

int g(z) {                  //scope 2b
    int b = x + 1;
    return b * b;
}

(a) z
(b) z, f, g
(c) z, f, x, g, z
(d) all of them