Concept: recognizing counts and accumulations

1. Which pattern does the following function implement?

```c
int g(int *items, int size)
{
    int i, m = 0;
    for (i = 0; i < size; ++i)
        m = m + items[i];
    return m;
}
```

(a) the filtered-accumulate pattern
(b) the counting pattern
(c) the accumulate pattern
(d) the filtered-counting pattern

2. Which pattern does the following function implement?

```c
int g(int *items, int size)
{
    int i, m = 0;
    for (i = 0; i < size; ++i)
        if (isPrime(items[i]))
            m = m + items[i];
    return m;
}
```

(a) the accumulate pattern
(b) the counting pattern
(c) the filtered-counting pattern
(d) the filtered-accumulate pattern

3. Which pattern does the following function implement?

```c
int f(int *items, int size)
{
    int j, n = 0;
    for (j = 0; j < size; ++j)
        n = n + 1;
    return n;
}
```

(a) the filtered-accumulate pattern
(b) the accumulate pattern
(c) the filtered-counting pattern
(d) the counting pattern

4. Which pattern does the following function implement?

```c
int g(int *items, int size)
{
    int i, m = 0;
    for (i = 0; i < size; ++i)
        if (isX(items[i]))
            m = m + 1;
    return m;
}
```
(a) the counting pattern
(b) the accumulate pattern
(c) the filtered-counting pattern
(d) the filtered-accumulate pattern

Concept: picking patterns, counts and accumulate

5. If I wish to solve this problem: determine the sum of the numbers from a to b, I should implement the:
   (a) the filtered-counting pattern
   (b) the filtered-accumulate pattern
   (c) the counting pattern
   (d) the accumulate pattern

6. If I wish to solve this problem: count the number of letters in a string that contains letters and digits, I should implement the:
   (a) the filtered-counting pattern
   (b) the filtered-accumulate pattern
   (c) the counting pattern
   (d) the accumulate pattern

7. If I wish to solve this problem: count the number of prime numbers in the range a to b, I should implement the:
   (a) the filtered-accumulate pattern
   (b) the filtered-counting pattern
   (c) the counting pattern
   (d) the accumulate pattern

8. If I wish to solve this problem: determine the sum of the Ramanujan numbers in the range a to b, I should implement the:
   (a) the counting pattern
   (b) the filtered-counting pattern
   (c) the accumulate pattern
   (d) the filtered-accumulate pattern

9. If I wish to solve this problem: determine the product of the numbers in an array, I should implement the:
   (a) the counting pattern
   (b) the accumulate pattern
   (c) the filtered-counting pattern
   (d) the filtered-accumulate pattern

10. If I wish to solve this problem: determine the product of the odd numbers in an array, I should implement the:
    (a) the counting pattern
    (b) the filtered-counting pattern
    (c) the filtered-accumulate pattern
    (d) the accumulate pattern

11. If I wish to solve this problem: interleave two arrays, filling a third array, I should implement the:
    (a) the shuffle pattern
    (b) the shuffle-counting pattern
    (c) the filtered-shuffle pattern
    (d) the filtered-accumulate pattern

Concept: recognizing map, filter, search, extreme, and extreme-index

12. Which pattern does the following function implement?
void f(int (*h)(int), int *r, int size)
{
    int j;
    for (j = 0; j < size; ++j)
        r[j] = h(r[j]);
}

(a) the filter pattern
(b) the extreme pattern
(c) the search pattern
(d) the map pattern

13. Which pattern does the following function implement?

int f(int h, int *r, int size)
{
    int j;
    for (j = 0; j < size; ++j)
        if (r[j] == h)
            return 1;
    return 0;
}

(a) the extreme index pattern
(b) the filter pattern
(c) the search pattern
(d) the map pattern

14. Which pattern does the following function implement?

int f(int h, int *r, int size)
{
    int j, w = 0;
    for (j = 0; j < size; ++j)
        if (r[j] == h)
            w = 1;
    return w;
}

(a) the extreme pattern
(b) the search pattern
(c) the map pattern
(d) the filter pattern

15. Which pattern does the following function implement?

void f(int (*checker)(int), int *r, int size)
{
    int i, w = 0;
    for (i = 0; i < size; ++i)
        if (checker(r[i]))
            {
            r[w] = r[i];
            ++w;
            }
}

(a) the filter pattern
(b) the map pattern
(c) the extreme index pattern
(d) the search pattern

16. Which pattern does the following function implement?
void g(int *t, int size)
{
    int j, v = 0;
    for (j = 0; j < size; ++j)
        if (isY(t[j]))
            { 
                t[v] = t[j];
                ++v;
            }
}
(a) the extreme pattern
(b) the search pattern
(c) the extreme index pattern
(d) the map pattern
(e) the filter pattern

17. Which pattern does the following function implement?

int h(int *r, int size)
{
    int i, w = r[0];
    for (i = 0; i < size; ++i)
        if (r[i] < w)
            w = r[i];
    return w;
}
(a) the map pattern
(b) the search pattern
(c) the extreme pattern
(d) the filter pattern

18. Which pattern does the following function implement?

int h(int *r, int size)
{
    int i, w = 0;
    for (i = 0; i < size; ++i)
        if (r[i] < r[w])
            w = i;
    return r[w];
}
(a) the extreme index pattern
(b) the search pattern
(c) the extreme pattern
(d) the filter pattern

19. Which pattern does the following function implement?

int g(int *t, int size)
{
    int i, v = 0;
    for (i = 0; i < size; ++i)
        if (t[i] < t[v])
            v = i;
    return v;
}
(a) the extreme index pattern
(b) the filter pattern
(c) the extreme pattern
(d) the search pattern
20. If I wish to solve this problem: square every number in an array, I should implement the:
   (a) the search pattern
   (b) the filter pattern
   (c) the map pattern
   (d) the accumulate pattern

21. If I wish to solve this problem: find if there is an even number in an array, I should implement the:
   (a) the filter pattern
   (b) the search pattern
   (c) the map pattern
   (d) the accumulate pattern

22. If I wish to solve this problem: extract the Ramanujan numbers from an array, I should implement the:
   (a) the filter pattern
   (b) the map pattern
   (c) the extreme pattern
   (d) the search pattern

23. If I wish to solve this problem: sum the squares of every number in an array, I should implement:
   (a) count followed by accumulate
   (b) filter followed by accumulate
   (c) map followed by accumulate
   (d) map followed by search

24. If I wish to solve this problem: find the largest even number in an array, I should implement the:
   (a) filter followed by search
   (b) filter followed by extreme
   (c) map followed by accumulate
   (d) map followed by search

25. If I wish to solve this problem: extract the even numbers from an array and then square each item in the resulting array, I should implement the:
   (a) map followed by filter
   (b) map followed by accumulate
   (c) filter followed by map
   (d) filter followed by extreme

26. If I wish to solve this problem: increment each number in an array and then extract the prime numbers, I should implement the:
   (a) filter followed by extreme
   (b) map followed by filter
   (c) filter followed by map
   (d) map followed by accumulate

27. Consider satisfying the following problem statement: sum the numbers from a to b (inclusive). What is wrong, if anything, with this attempt?
int sum(int a, int b)
{
    int i, total = 0;
    for (i = 0; i < b + 1; ++i)
        total = total + i;
    return total;
}

(a) total should start at a, not 0
(b) The function is correct
(c) i should start at a, not 0
(d) i should end at b, not b + 1

28. Consider satisfying the following problem statement: *sum the numbers from a to b (inclusive).* What is wrong, if anything, with this attempt?

```c
int sum(int a, int b)
{
    int i, total = b;
    for (i = a; i < b; ++i)
        total = total + i;
    return total;
}
```

(a) total should start at 0, not b
(b) The function is correct
(c) i should start at 0, not a
(d) i should end at b +1, not b

29. Consider satisfying the following problem statement: *sum the numbers from a to b (inclusive).*

```c
int sum(int a, int b)
{
    int i, total = a;
    for (i = a + 1; i < b + 1; ++i)
        total = total + i;
    return total;
}
```

What is wrong, if anything, with this attempt?

(a) i should end at b, not b +1
(b) i should start at a, not a +1
(c) The function is correct
(d) total should start at 0, not a

30. Consider satisfying the following problem statement: *sum the numbers from a to b (inclusive).* What is wrong, if anything, with this attempt?

```c
int sum(int a, int b)
{
    int i, total = b;
    for (i = a; i < b; i = i + 0)
        total = total + i;
    return total;
}
```

(a) i should start at a +1, not a
(b) i should end at b +1, not b
(c) The step should be 1, not 0
(d) The function is correct

31. Consider satisfying the following problem statement: *sum the numbers from a to b (inclusive).* What is wrong, if anything, with this attempt?
int sum(int a, int b)
{
    int i, total = a;
    for (i = a; i < b+1; ++i)
        total = total + i;
    return total;
}

(a) The function is correct
(b) The total should start at 0, not a
(c) i should end at b, not b +1
(d) i should start at 0, not a

32. Consider satisfying the following problem statement: sum the numbers from a to b (inclusive). What is wrong, if anything, with this attempt?

int sum(int a, int b)
{
    int i, total = 0;
    for (i = a; i < b+1; ++i)
        total = total + i;
    return total;
}

(a) The function is correct
(b) i should end at b, not b +1
(c) total should start at 1, not 0
(d) i should start at a +1, not a

33. Consider satisfying the following problem statement: shuffle two equal length arrays, filling a third array. What is wrong, if anything, with this attempt? Assume the third array has enough room.

void shuffle(int *a1, int *a2, int *a3, int size)
{
    int i;
    for (i = 0; i < size; ++i)
    {
        a3[i*2] = a2[i];
        a3[i*2+1] = a1[i];
    }
}

(a) The function is legal, but a3 will be missing values
(b) The function is correct
(c) The function is legal, but does not shuffle
(d) The function can fail with an error

34. Consider satisfying the following problem statement: shuffle two unequal length arrays (a1 longer than a2), filling a third array. What is wrong, if anything, with this attempt? Assume sN is the size of aN and that the third array has enough room.

void shuffle(int *a1, int s1, int *a2, int s2, int *a3)
{
    int i;
    for (i = 0; i < s1; ++i)
    {
        a3[i*2] = a2[i];
        a3[i*2+1] = a1[i];
    }
}

(a) The function can fail with an error
(b) The function is correct
(c) The function is legal, but a3 will be missing values
(d) The function is legal, but does not shuffle

35. Consider satisfying the following problem statement: *shuffle two unequal length arrays (a2 longer than a1), filling a third array*. What is wrong, if anything, with this attempt? Assume $sN$ is the size of $aN$ and that the third array has enough room.

```c
void shuffle(int *a1,int s1,int *a2,int s2,int *a3)
{
    int i;
    for (i = 0; i < s1; ++i)
    {
        a3[i*2] = a1[i];
        a3[i*2+1] = a2[i];
    }
}
```

(a) The function can fail with an error
(b) The function is correct
(c) The function is legal, but $a3$ will be missing values
(d) The function is legal, but does not shuffle

36. Consider satisfying the following problem statement: *shuffle two unequal length arrays, filling a third array*. What is wrong, if anything, with this attempt? Assume $sN$ is the size of $aN$ and that the third array has enough room.

```c
int shuffle(int *a1,int s1,int *a2,int s2,int *a3)
{
    int i = 0;
    while (i < s1 && i < s2)
    {
        a3[2*i] = a1[i];
        a3[2*i+1] = a2[i];
        i += 1;
    }
}
```

(a) The function is legal, but $a3$ will be missing values
(b) The function can fail with an error
(c) The function is correct
(d) The function is legal, but does not shuffle

37. Consider satisfying the following problem statement: *shuffle two equal length arrays, filling a third array*. What is wrong, if anything, with this attempt? Assume the third array has enough room.

```c
void shuffle(int *a1,int *a2,int size,int *a3)
{
    int j = 0,i = 0;
    while (j < size && i < size)
    {
        a3[i+j] = a1[j];
        a3[i+j+1] = a2[j];
        j += 1; i += 1;
    }
}
```

(a) The function is correct
(b) The function can fail with an error
(c) The function is legal, but $a3$ will be missing values
(d) The function is legal, but does not shuffle

38. Consider satisfying the following problem statement: *shuffle two equal length arrays, filling a third array*. What is wrong, if anything, with this attempt? Assume the third array has enough room.
void shuffle(int *a1,int *a2,int *a3,int size)
{
    int j = 0,i = 0;
    while (j < size || i < size)
    {
        a3[i+j] = a1[j];
        a3[i+j+1] = a2[i];
        j += 1; i += 1;
    }
}

(a) The function is legal, but does not shuffle
(b) The function is correct
(c) The function can fail with an error
(d) The function is legal, but a3 will be missing values

Concept: Input-Output using Loops

39. Which criticism of the following scanner code is valid?

FILE *fp = fopen(fileName,"r");
char *token = readToken(fp);
while (!feof(fp))
{
    printf("%s\n",token);
}
fclose(fp);

(a) the file is opened at the wrong time
(b) a file name is not given to fopen
(c) the code is correct
(d) the token is only read once

40. Which criticism of the following scanner code is valid?

int count = 0;
FILE *fp = fopen(fileName,"r");
char *token = readToken(fp);
while (!feof(fp))
{
    token = readToken(fp);
    printf("%s\n",token);
}
fclose(fp);

(a) a literal string must be given to fopen
(b) the code is correct
(c) the file pointer is closed at the wrong time
(d) the token is only read once

41. Which criticism of the following scanner code is valid?

char *token;
int count = 0;
FILE *fp = fopen(fileName,"r");
while (!feof(fp))
{
    printf("%s\n",token);
    token = readToken(fp);
}
fclose(fp);

(a) the file pointer is closed at the wrong time
(b) a literal string must be given to fopen
(c) there is no initial read to set token
(d) the code is correct

42. Which criticism of the following scanner code is valid?

```c
int count = 0;
FILE *fp = fopen(fileName,"r");
char *token = readToken(fp);
while (!feof(fp))
{
    token = readToken(fp);
}
fclose(fp);
```

(a) the file pointer is closed at the wrong time
(b) the tokens are read but not processed in any way
(c) the code is correct
(d) a literal string must be given to the `fopen` function

43. Which criticism of the following scanner code is valid?

```c
int count = 0;
FILE *fp = fopen(fileName,"r");
char *token = readToken(fp);
while (!feof(fp))
{
    printf("%s\n",token);
    token = readToken(fp);
    fclose(fp);
}
```

(a) the tokens are printed before the read
(b) the file pointer is closed at the wrong time
(c) the code is correct
(d) there should be no call to readToken before the loop

44. Which criticism of the following scanner code is valid?

```c
int count = 0;
FILE *fp = fopen(fileName,"r");
char *token = readToken(fp);
while (!feof(fp))
{
    printf("%s\n",token);
    token = readToken(fp);
}
```

(a) there should be no call to readToken before the loop
(b) tokens are not printable
(c) the file pointer is not closed
(d) the token should be compared to the empty string

45. T or F: This code correctly counts the number of tokens in a file:

```c
int count = 0;
FILE *fp = fopen(fileName,"r");
count = 1
char *token = readToken(fp);
while (!feof(fp))
{
    count = count + 1
    token = readToken(fp);
}
fclose(fp);
```

46. T or F: This code correctly counts the number of tokens in a file:
int count = 0;
FILE *fp = fopen(fileName,"r");
char *token = readToken(fp);
while (!feof(fp))
{
    count = count + 1
    token = readToken(fp);
}
fclose(fp);

47. T or F: This code correctly counts the number of tokens in a file:

    int count = 0;
    FILE *fp = fopen(fileName,"r");
    char *token = readLine(fp);
    while (!feof(fp))
    {
        count = count + 1;
        token = readline(fp);
    }
    fclose(fp);

48. T or F: This code correctly counts the number of lines in a file:

    int count = 0;
    FILE *fp = fopen(fileName,"r");
    char *token = readLine(fp);
    while (!feof(fp))
    {
        count = count + 1;
        token = readLine(fp);
    }
    fclose(fp);

Concept: implementing loops

49. (2pts) Implement this accumulation loop: total up the product of the numbers from 1 to \(x\), inclusive.

50. (2pts) Implement this accumulation loop: total up the product of the numbers from \(a\) to \(b\), inclusive.

51. (2pts) Implement this accumulation loop: total up the sum of the numbers from \(a\) to \(b\), inclusive.
52. (2pts) Implement this accumulation loop: total up the sum of the numbers from 1 to $x$, inclusive.

53. (2pts) Implement this counting loop: count the number of characters in a string $s$.

54. (2pts) Implement this filtered-counting loop: count the number of uppercase characters in a string $s$. Assume $\text{isUpper}(s[i])$ returns true if $s[i]$ is uppercase and false otherwise.

55. (2pts) Implement this filtered-counting loop: count the number of vowels in a string $s$. Assume $\text{isVowel}(s[i])$ returns true if $s[i]$ is a vowel and false otherwise.

56. (2pts) Implement this filtered-counting loop: count the number of prime numbers from $a$ to $b$ inclusive. Assume $\text{isPrime}(i)$ returns true if $i$ is prime and false otherwise.

57. (2pts) Implement this filtered-counting loop: count the number of numbers that are divisible by 2 or 3 from $a$ to $b$ inclusive.
58. (2pts) Convert this while loop to a for loop:
   
   ```
   i = 0;
   while (i < x + 1)
   {
   printf("%d\n",x);
   i = i + 1;
   }
   ```

59. (2pts) Convert this while loop to a for loop:

   ```
   i = 3;
   while (i < x)
   {
   printf("%d\n",x);
   i = i + 2;
   }
   ```

60. (2pts) Convert this while loop to a for loop:

   ```
   i = 3;
   while (i < x)
   {
   printf("%d\n",x);
   i = i + 2;
   }
   ```

61. (2pts) Convert this while loop to a for loop:

   ```
   i = 0;
   while (i < strlen(s))
   {
   printf("%c\n",s[i]);
   i = i + 1;
   }
   ```
62. (2pts) Convert this for loop to a while loop:

```c
for (i = 2; i < strlen(s); i += 1)
    printf("%c\n", s[i]);
```

63. (2pts) Convert this for loop to a while loop:

```c
count = 0;
for (i = 0; i < strlen(s); i += 2)
    if (isupper(s[i]))
        count += 1;
```