Exercise 1

Using static scope and pass-by-value:

```
int x = 2;
int fie(int y){ x = x + y; }
{int x = 5;
 fie(x);
 write(x);
}
write(x);
```

Output:

$$\boxed{5, 2}$$

Exercise 2

Using dynamic scope and call-by-reference:

$$\boxed{5, 7}$$

Exercise 3

Using static scope and pass-by-reference:

$$\boxed{6, 2}$$

Exercise 4

Using static scope and pass-by-value with post-increment:

$$\boxed{6, 2}$$

Exercise 5

Using static scope and call-by-name:

$$\boxed{8, 2}$$

Exercise 6

Using dynamic scope and call-by-reference:

$$\boxed{5, 1}$$

Exercise 7

Using static scope and call-by-reference:

$$\boxed{6}$$

Exercise 8

Using reference parameters:

$$\boxed{0, 0, 1}$$

Exercise 9

Using value-result parameters:

$$\boxed{2, 4}$$

Exercise 10

Using constant parameters:

$$\boxed{2, 2}$$

Probable behavior: The assignment to `Y` inside `foo` will cause an error.

Exercise 11

Using name parameters:

$$\boxed{3, 4}$$

Exercise 12

Deep binding with closures in dynamic scope ensures that the referencing environment is the one that was in place when the function was created rather than when it was invoked. This is implemented using closures, which package the function's code along with the environment in which it was defined. When functions are passed as arguments in dynamically scoped languages, closures allow the correct variables to be accessed even if the function is called in a different scope.

Exercise 13

(i) By value-result:

$$\boxed{1}$$

(ii) By reference:

$$\boxed{0}$$

Exercise 14

Output of `main()` execution:

$$\boxed{3, 0}$$

Exercise 15

```
f(4) calls f(3), which calls f(2), which calls f(1) and throws E.
E is caught in f(2), returning 2+1 = 3.
f(3) returns 3.
f(4) returns 3.
```

Final result:

$$\boxed{3}$$