Let's analyze each function in your code one by one, check if the declaration and returning are correct, and provide how to receive the returned values in the main function.

1. func1 (Returning std::vector<int>)

```
vector<int> func1(int size) {
   vector<int> a(size, 9);
   return a;
}
```

Analysis:

- This is **correct**. The function creates a std::vector<int> of size size, initialized with 9, and returns it.
- **Return by value** is fine here as std::vector handles memory management automatically.

To Receive in main:

```
vector<int> arr = func1(5);  // Example: Receive a vector of size 5 initialized with 9
```

2. fun2 (Returning std::vector<std::vector<int>>)

```
vector<vector<int>> fun2(int rows, int cols) {
   vector<vector<int>> arr(rows, vector<int>(cols, 0));
   return arr;
}
```

Analysis:

- This is **correct**. The function creates a 2D vector of size rows x cols, with all elements initialized to 0, and returns it.
- The initialization of the inner vectors is done using vector<int>(cols, 0).

To Receive in main:

```
vector<vector<int>> arr = fun2(3, 4);  // Example: 3 rows and 4 columns initialized with 0
```

3. fun3 (Returning a Pointer to an Array)

```
int* fun3(int size) {
   int* arr = new int[size];
   return arr;
}
```

Analysis:

- This is **correct**, but be cautious. You are using new to allocate memory dynamically for the array, and the caller is responsible for freeing the memory (using delete[]).
- It is recommended to use std::vector or std::array unless you specifically need dynamic memory allocation.

To Receive in main:

```
int* arr = fun3(5);  // Example: Receives a dynamically allocated array of size 5

// Accessing values
for (int i = 0; i < 5; i++) {
   cout << arr[i] << " ";  // Accessing array elements
}

// Don't forget to delete the dynamically allocated array to avoid memory leak
delete[] arr;
```

4. fun4 (Returning Pointer to a 2D Array)

- **Case 1: Returning a Pointer to a 2D Array (Incorrect)**

```
int** fun4(int r, int c) {
   int arr = new int[r][c];  // Incorrect: This is invalid syntax for dynamic 2D arrays
   return arr;
}
```

Analysis:

- This is **incorrect** because you cannot directly create a 2D array dynamically like this with new. You need to allocate memory for each row separately.

To fix this:

```
int** fun4(int r, int c) {
   int** arr = new int*[r];
   for (int i = 0; i < r; i++) {
      arr[i] = new int[c];  // Dynamically allocating each row
   }
   return arr;
}
```

To Receive in main:

```
int** arr = fun4(3, 4);  // Receives a dynamically allocated 2D array (3x4)

for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 4; j++) {
      cout << arr[i][j] << " ";  // Accessing elements
   }
   cout << endl;
}

// Don't forget to delete the dynamically allocated 2D array to avoid memory leaks
for (int i = 0; i < 3; i++) {
   delete[] arr[i];  // Deleting each row
}
delete[] arr;  // Deleting the array of row pointers
```

- **Case 2: Returning a Reference to a Static 1D Array (Correct)**

```
int (&fun4())[3] {
  static int arr[3] = {};
  return arr;
}
```

Analysis:

- This is **correct**. The function returns a reference to a statically allocated 1D array of size 3.
- A static array persists beyond the function's scope, which allows you to return a reference to it.

To Receive in main:

```
int (&arr)[3] = fun4();  // Receives a reference to a static 1D array of size 3

for (int i = 0; i < 3; i++) {
  cout << arr[i] << " ";  // Accessing elements
}
```

5. func5 (Returning Reference to Static 2D Array)

```
int (&func5())[3][3] {
  static int arr[3][3] = {};
  return arr;
}
```

Analysis:

- This is **correct**. The function returns a reference to a static 2D array of size 3x3.
- As with the previous case, the array persists beyond the function's scope due to the static keyword.

To Receive in main:

```
int (&arr)[3][3] = func5();  // Receives a reference to a static 2D array

for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 3; j++) {
      cout << arr[i][j] << " ";  // Accessing elements
   }
   cout << endl;
}
```

6. fun6 (Returning std::array<int, 3>)

```
array<int, 3> fun6() {
   array<int, 3> arr(0);  // Initializes array with 0
   return arr;
}
```

Analysis:

- This is **incorrect**. std::array should be initialized with its size and values, but array<int, 3> arr(0) is not valid since it expects a size and cannot be initialized with a value like this.

Fix it:

```
array<int, 3> fun6() {
   array<int, 3> arr = {0};  // Initializing the array with 0 (all elements)
   return arr;
}
```

To Receive in main:

```
array<int, 3> arr = fun6();  // Receives a fixed-size array of size 3 initialized with 0
```

```
for (int i = 0; i < 3; i++) {
   cout << arr[i] << " ";  // Accessing elements
}
```

7. fun7 (Returning std::array<std::array<int, 4>, 3>)

```
array<array<int, 4>, 3> fun7() {
   array<array<int, 4>, 3> arr(3);  // Initializes the array with 3 elements, each of size 4
   return arr;
}
```

Analysis:

- This is **incorrect**. You cannot initialize a std::array with a single value (e.g., 3). Each element should be initialized in a way that matches the type.

Fix it:

```
array<array<int, 4>, 3> fun7() {
   array<array<int, 4>, 3> arr = {{{1, 1, 1, 1}, {2, 2, 2, 2}, {3, 3, 3, 3}}};  // Example initialization
   return arr;
}
```

To Receive in main:

```
array<array<int, 4>, 3> arr = fun7();  // Receives a fixed-size 2D array

for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 4; j++) {
      cout << arr[i][j] << " ";  // Accessing elements
   }
   cout << endl;
}
```

Summary of Corrected Code and How to Receive:

1. **func1:** vector<int> arr = func1(5);
2. **fun2:** vector<vector<int>> arr = fun2(3, 4);
3. **fun3:** int* arr = fun3(5);
4. **fun4 (1D static array):** int (&arr)[3] = fun4();
5. **func5:** int (&arr)[3][3] = func5();
6. **fun6:** array<int, 3> arr = fun6();
7. **fun7:** array<array<int, 4>, 3> arr = fun7();

Make sure to correct the errors where indicated and you will be able to successfully receive and use the return values in the main function.