Manipulation of Constant and Non-Constant Pointers with Const and Non-Const Variables

Variable Declarations:

- `int i = 10;` // Non-constant integer variable `i` initialized to 10.
- `const int j = 20;` // Constant integer variable `j` initialized to 20.

Modification of Non-Constant Variable `i` via Pointer `p`

1. **Pointer Syntax:**
   - `int * p = &i;` // Pointer `p` is initialized to the address of variable `i`.
2. **Description:**
   - The assignment `*p = 30;` succeeds because `p` points to a non-constant variable (`i`).
   - Execution: `*p = 30;`
3. **Conclusion:**

The modification of `i` via `p` succeeds

Attempting to Assign Constant Variable `j` to Non-Constant Pointer `q`

1. **Pointer Syntax:**
   - `int * q = &j;` // Attempt to assign constant variable `j` to non-constant pointer `q`.
2. **Description:**
   - This assignment results in the error `invalid conversion from "const int*" to "int*"` because `q` is a non-constant pointer, and it is being assigned the address of a constant variable (`j`).
   - `Error: invalid conversion from "const int*" to "int*" indicates that there is an attempt to convert a constant pointer to a non-constant pointer, which is not allowed due to type safety reasons.`
3. **Conclusion:**

The assignment of constant variable `j` to non-constant pointer `q` fails

Modification Attempt on Constant Variable `j` via Pointer `q`

1. **Pointer Syntax:**
   - `const int * q = &j;` // Pointer `q` is initialized to the address of constant variable `j`.
2. **Description:**
   - The assignment `*q = 30;` fails because `q` points to a constant variable (`j`).
   - `Error: "assignment of read-only location" signifies an attempt to modify a value or variable that has been declared as read-only or constant.`
3. **Conclusion:**

The modification of `j` via `q` fails

Reassignment of Pointer `q` to Non-Constant Variable `i`

1. **Pointer Syntax:**
   - `const int * q = &j;` // Pointer `q` is initialized to the address of constant variable `j`.
2. **Description:**
   - Reassigning `q` to point to non-constant variable `i` is valid because `q` itself is not a constant pointer.
   - Execution: `q = &i;`
3. **Conclusion:**

Reassigning `q` to point to `i` is valid

Modification of Non-Constant Variable via Constant Pointer `r`

1. **Pointer Syntax:**
   - `int * const r = &i;` // Pointer `r` is initialized to the address of non-constant variable `i`.
2. **Description:**
   - The modification of the value pointed by `r` to `40` succeeds because `r` is a constant pointer to a non-constant type.
   - Execution: `*r = 40;`
3. **Conclusion:**

The modification of `i` via `r` succeeds

Reassignment of Pointer `r` to Non-Constant Variable `j`

1. **Pointer Syntax:**
   - `int * const r = &i;` // Pointer `r` is initialized to the address of non-constant variable `i`.
2. **Description:**
   - Attempting to reassign `r = &j;` to point to a different address (`&j`) fails because `r` is a constant pointer, and its value cannot be modified once initialized.
   - `Error: "assignment of read-only location" signifies an attempt to modify a value or variable that has been declared as read-only or constant.`
3. **Conclusion:**

Reassignment of `r` fails

Modification Attempt on Constant Pointer `s`

1. **Pointer Syntax:**
   - `const int * const s = &j;` // Pointer `s` is initialized to the address of constant variable `j`.
2. **Description:**
   - Attempting to modify the pointer itself (`s = &i;`) results in a compilation error because `s` is a constant pointer.
   - Similarly, attempting to modify the value pointed by `s` (`*s = 30;`) results in a compilation error because `s` points to a constant variable (`j`).
   - `Error: "assignment of read-only location" signifies an attempt to modify a value or variable that has been declared as read-only or constant.`
3. **Conclusion:**

Compilation error occurs due to attempts to modify `s`