# STL Algorithms

The **Standard Template Library (STL) Algorithms** provide useful, reusable functions for handling data structures like arrays, vectors, and lists. These algorithms are found in the `<algorithm>` and `<numeric>` headers and include operations for sorting, searching, modifying, counting, and more. A list of a few selected algorithms follows.

## Key Categories of STL Algorithms

1. **Sorting & Searching**
   - `sort()`
   - `binary_search()`
   - `find()`
2. **Modification**
   - `reverse()`
   - `transform()`
   - `replace()`
3. **Counting & Checking**
   - `count_if()`
   - `all_of()`
   - `any_of()`
4. **Numeric Operations**
   - `accumulate()`
   - `partial_sum()`

---

## Example 1: Working with Simpsons Characters

We will use a **vector of Simpson family members** with their **age** and apply various STL algorithms.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>  // For accumulate
using namespace std;

// Person class stores character details
```

```
class Person {
public:
    string name;
    int    age;
    Person(string name="NoName", int age=0) : name(name), age(age) {}

    void print()  {
        cout << "   Person[ " << name << ", " << age << " ]" << endl;
    }
};

// Generic function to print a vector ----------------------------------
template <typename T>
void showVector(vector<T>& v, string caption = "") {
    cout << caption << endl;
    for (T& item : v) {
        item.print();
    }
    cout << string(30, '-') << endl;
}

//-----------------------------------------------------------------------
int main() {
    vector<Person> simpsons =
    {
        Person("Homer", 39), Person("Marge", 36), Person("Bart", 10 ),
        Person("Lisa", 8),   Person("Maggie", 1 )
    };

    showVector(simpsons, "Original List:");

    // Case 1. Sorting by Age (Using sort (ASC))
    sort(simpsons.begin(), simpsons.end(),
        [](const Person& a, const Person& b) { return a.age < b.age;  } );

    showVector(simpsons, "Sorted by Age:");

    // Case 2. Find Bart using find_if
    auto it = find_if(simpsons.begin(), simpsons.end(),
                    [](const Person& s) { return s.name == "Bart"; });

    if (it != simpsons.end()) {
        cout << "Found: " << it->name << " is " << it->age << " years old.\n";
    }
    else {
        cout << "Bart not found.\n";
    }


    // Case 3. Find first older person using find_if
    auto it2 = find_if(simpsons.begin(), simpsons.end(),
                      [](const Person& s) { return s.age > 35; });

    while (it2 != simpsons.end()) {
        cout << "Older Person Found: ";
        it2->print();
        // Find next older person
        it2 = find_if(++it2, simpsons.end(),
```

```
                    [](const Person& s) { return s.age > 35; });
    }

    // Case 4. Count how many are under 18 (Using count_if)
    int under18 = count_if(simpsons.begin(), simpsons.end(),
                           [](const Person& s) { return s.age < 18; });

    cout << "Number of people under 18: " << under18 << endl;

    // Case 5. Transform names to uppercase (Using transform)
    transform(simpsons.begin(), simpsons.end(), simpsons.begin(),
                [](Person& s) {
                    for (char& c : s.name) c = toupper(c);
                    return s;
                });

    showVector(simpsons, "Names in Uppercase:");

    // Case 6. Sum of all ages (Using accumulate)
    int startingCounter = 0;
    int totalAge = accumulate(simpsons.begin(), simpsons.end(), startingCounter,
                              [](int sum, const Person& s) { return sum + s.age; });

    cout << "Total Age of all Simpsons: " << totalAge << " years.\n";

    cout << endl << endl;

    // Case 7. Using for_each to print details
    cout << "Fun facts about Simpsons characters:\n";
    for_each(simpsons.begin(), simpsons.end(),
        [](const Person& p) {
            cout << p.name << " is " << p.age << " years old. ";
            if (p.name == "HOMER") cout << "Loves donuts!";
            else if (p.name == "MARGE") cout << "Has the tallest blue hair!";
            else if (p.name == "BART") cout << "Often says - Ay, caramba";
            else if (p.name == "LISA") cout << "Enjoys playing the saxophone!";
            else if (p.name == "MAGGIE") cout << "Loves her pacifier!";
            cout << endl;
        });

    // Case 8. Using for_each to update ages
    for_each(simpsons.begin(), simpsons.end(),
        [](Person& p) {
            p.age += 1;
        });

    //show the updated list
    showVector(simpsons, "Updated List:");


    cout << "Al done!\n";
}
```

## Explanation of STL Algorithms Used

## Example Output

```
Original List:
   Person[ Homer, 39 ]
   Person[ Marge, 36 ]
   Person[ Bart, 10 ]
   Person[ Lisa, 8 ]
   Person[ Maggie, 1 ]
------------------------------
Sorted by Age:
   Person[ Maggie, 1 ]
   Person[ Lisa, 8 ]
   Person[ Bart, 10 ]
   Person[ Marge, 36 ]
   Person[ Homer, 39 ]
------------------------------
Found: Bart is 10 years old.

Older Person Found:    Person[ Marge, 36 ]
Older Person Found:    Person[ Homer, 39 ]

Number of people under 18: 3

Names in Uppercase:
   Person[ MAGGIE, 1 ]
   Person[ LISA, 8 ]
   Person[ BART, 10 ]
   Person[ MARGE, 36 ]
   Person[ HOMER, 39 ]
------------------------------

Total Age of all Simpsons: 94 years.


Fun facts about Simpsons characters:
MAGGIE is 1 years old. Communicates with her pacifier!
LISA is 8 years old. Loves playing the saxophone!
BART is 10 years old. Always says 'Eat my shorts!'
MARGE is 36 years old. Has the tallest blue hair!
HOMER is 39 years old. Loves donuts!

Updated list:
   Person[ MAGGIE, 2 ]
   Person[ LISA, 9 ]
   Person[ BART, 12 ]
   Person[ MARGE, 37 ]
   Person[ HOMER, 40 ]

Al done!
```

**Advantages of Using STL Algorithms:**

1. **Code Simplicity & Readability** – STL algorithms provide concise and expressive code, reducing boilerplate logic.
2. **Efficiency** – STL algorithms are optimized and often outperform manually implemented versions.
3. **Reusability** – STL provides a wide variety of generic algorithms, reducing the need to write custom implementations.
4. **Type Safety** – Since STL works with templates, it maintains type safety and avoids unnecessary casting.

**Disadvantages of Using STL Algorithms:**

1. **Steep Learning Curve** – Understanding iterators, function objects, and lambda expressions can be difficult for beginners.
2. **Lack of Fine Control** – STL algorithms work in a general way and may not always be optimal for specific use cases.