Advanced Programming - Supplementary Examples

Standard Template Library (STL)

This document provides examples and exercises to further your understanding of the Standard Template Library (STL) in C++. It is designed to help you practice and deepen your understanding of STL components.

Examples

Using `std::find_if_not` with Custom Predicate

The following code demonstrates the use of `std::find_if_not` to find the first number in a list that is not divisible by 3.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

bool notDivisibleBy3(int n) {
    return n % 3 != 0;
}

int main() {
    std::vector<int> numbers = {3, 6, 9, 12, 5, 15};
    auto it = std::find_if_not(numbers.begin(), numbers.end(), notDivisibleBy3);

    if (it != numbers.end()) {
        std::cout << "First number not divisible by 3 is: " << *it << std::endl;
    } else {
        std::cout << "All numbers are divisible by 3." << std::endl;
    }
    return 0;
}
```

Exercise: Write a program that uses `std::find_if` to find the first number in a vector that is both even and greater than 10. Define a custom predicate function to achieve this.

Using `std::transform` with Lambda Functions

This example shows how to use `std::transform` using inline lambda functions instead of previously defined functions.

```cpp
#include <vector>
#include <algorithm>
#include <iostream>

// define a function that calculates the squared value of an integer
int squared(int &x) {
```

```
    return x*x;
}

int main() {
    // create a vector
    std::vector<int> vec = {1, 2, 3, 4, 5};
    // declare a result vector for the transformed values
    // (in this case it will be squared values
    std::vector<int> result(vec.size());

    // using a function
    std::transform(vec.begin(), vec.end(), result.begin(), squared);

    std::cout << "Transformed vector (using the squared function):" << std::endl;
    for (int x : result) {
        std::cout << x << " ";
    }
    std::cout << std::endl;

    // using a lambda function
    std::transform(vec.begin(), vec.end(), result.begin(), [](int x) { return x * x; });

    std::cout << "Transformed vector (using a lambda function):" << std::endl;
    for (int x : result) {
        std::cout << x << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Using `std::accumulate` with Custom Operation

```
#include <vector>
#include <numeric>
#include <iostream>

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    // recalling the previous example, let's use a lambda function
    int product = std::accumulate(vec.begin(), vec.end(), 1,
                                  [](int a, int b) { return a * b; });

    std::cout << "Product of all elements in the vector: " << product << std::endl;

    return 0;
}
```

Lambda Expressions with Capture by Reference

This example demonstrates the use of lambda expressions to capture a variable by reference and use it in an algorithm.

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```cpp
int main() {
    int multiplier = 5;
    std::vector<int> vec = {1, 2, 3, 4, 5};

    std::for_each(vec.begin(), vec.end(), [&multiplier](int &n) { n *= multiplier; });

    std::cout << "Modified vector: ";
    for (int n : vec) {
        std::cout << n << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Custom comparator for `std::sort`

```cpp
#include <vector>
#include <algorithm>
#include <iostream>

// create an structure that compares two pairs of integers
// by the value of the second element of the pairs
struct CustomCompare {
    bool operator()(const std::pair<int, int>& a, const std::pair<int, int>& b)
    const {
        return a.second < b.second;
    }
};

int main() {
    // create a vector of pairs
    std::vector<std::pair<int, int>> vec = {{1, 2}, {3, 1}, {5, 4}, {7, 3}};
    // use the sort algorithm, passing our struct as sorting criterion
    std::sort(vec.begin(), vec.end(), CustomCompare());

    std::cout << "Sorted vector of pairs by second element:" << std::endl;
    for (const auto& p : vec) {
        std::cout << "{" << p.first << ", " << p.second << "} ";
    }
    std::cout << std::endl;

    return 0;
}
```

Exercise: Write a program that sorts a vector of `std::pair<int, std::string>` objects. Sort the vector first by the integer part in ascending order, and if two elements have the same integer, then sort by the string part in descending order. Use `std::sort` with a custom comparator.

Using `std::set_intersection`

```cpp
#include <vector>
#include <algorithm>
#include <iostream>

```

```cpp
int main() {
    std::vector<int> vec1 = {1, 2, 3, 4, 5};
    std::vector<int> vec2 = {3, 4, 5, 6, 7};
    std::vector<int> result;

    // use std::set_intersection to find the common elements in both vectors
    // use std::back_inserter to store the result
    // (back_inserter is an iterator that makes use of push_back to store values)
    std::set_intersection(vec1.begin(), vec1.end(), vec2.begin(), vec2.end(),
                          std::back_inserter(result));

    std::cout << "Intersection of two vectors:" << std::endl;
    for (int x : result) {
        std::cout << x << " ";
    }
    std::cout << std::endl;

    return 0;
}
```