Hands-on Activity 13.1	
Parallel Algorithms and Multithreading	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 11/02/25**
Section: CPE21S4	**Date Submitted: 11/02/25**
Name(s): Francis Nikko I. Andoy	**Instructor:** Engr. Jimlord Quejado

A. Output(s) and Observation(s)

Part 1: Simple One-Threaded Example

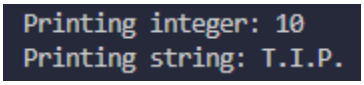
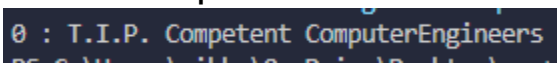
Output/Screen Shot:	**Analysis:**
	Firstly, I noticed that it used a library called thread and since we are dealing with multithreading activity. Then after that we define a function that is called print with a parameter that has an integer and string value, the string value also have the & operator, which means that there will be no copy or modification on the string value. Since we used the & operator we are passing the string reference and not the exact value, just the reference. In the main we first created an object from the thread class named t1, then we called the print function to print the defined argument, which is the 10 and the T.I.P. after that we used a built-in function inside the thread class.

Table 13-1. Simple One-Threaded Example

Part 2: Multi-Threaded Example

Output/Screen Shot:



Code:

```cpp
#include <iostream>
#include <thread>
#include <string>
#include <vector>

void print(int n, const std::string &str){
    std::string msg = std::to_string(n) + " : " + str;
    std::cout << msg << std::endl;
}

int main() {
    std::vector<std::string> s = {
        "T.I.P."
        " Competent"
        " Computer"
        "Engineers"
    };
    std::vector<std::thread> threads;
    for (int i = 0; i < s.size(); i++) {
        threads.push_back(std::thread(print, i, s[i]));
    }
    for (auto &th : threads) {
        th.join();
    }

    return 0;
}
```

Analysis:

For my analysis here, I noticed that we just used the parameter of the part 1 example. But the structure of that function is different, first is the variable that will be the storage of the integer that will be string later. In that line there is a std::to_string(n) the variable will actually take a string value but since we are using a for loop this will turn into an integer. That is why the to_string is important, because this will transform all the integer values into a string. We then created a list that is called s, it has a string value and this actually represents the for loop in the earlier explanation. The first iteration which is 0 is for T.I.P. then all the way to the last string, we also did a concatenation since we used + operator inside of the function. Then after that we used a for loop again to move inside of the vector and another & so that it will pass a reference of that value not the actual value to avoid being copied, after this we used the join again that came from the thread class.

Table 13-2. Multithreaded Example.

B. Answers to Supplementary Activity

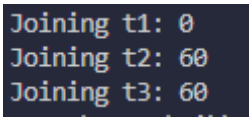
Part A: Demonstrate an understanding of parallelism, concurrency, and multithreading in C++ by answering the given questions.

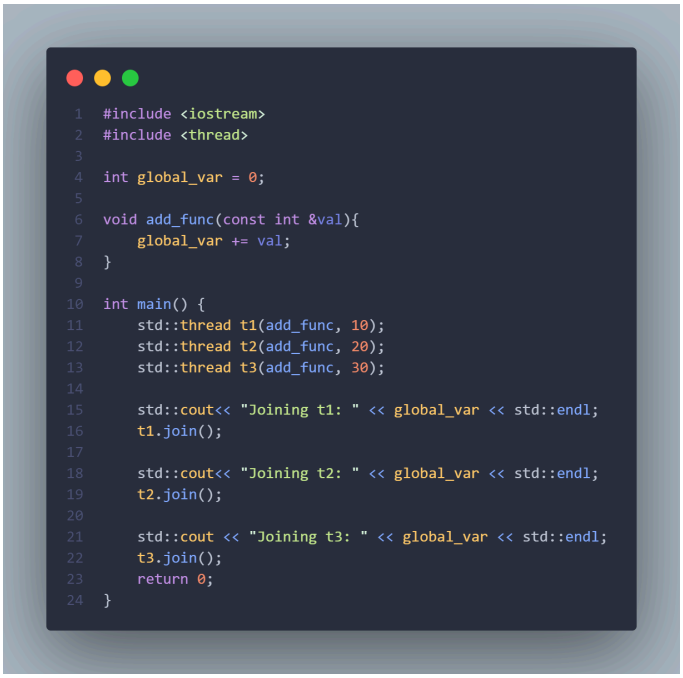
1. **Write a definition of multithreading and its advantages/disadvantages.**
   - According to GeeksforGeeks, Multithreading is a technique where a program is divided into smaller units of execution called threads. Each thread runs independently but shares resources like memory, allowing tasks to be performed simultaneously. This helps improve performance by utilizing multiple CPU cores efficiently. (2025b)
   - The advantages of multithreading are:
     - Improved Performance
     - Better Responsiveness
     - Resource Sharing
     - Simplicity of Modeling
   - As for the disadvantages
     - Synchronization Issues
     - Debugging Difficulty
     - Overhead and Context Switching
     - Complexity
   - These are based on the study of (Advantages and Disadvantages of Multithreading in Java Training Course, n.d.)

2. **Rationalize the use of multithreading by providing at least 3 use-cases.**
   - Based on Lucasnscr, we can use multithreading in a lot of ways. The first way is Parallel Computing this is when dividing a large computational task into smaller sub-tasks, each sub-task must be completed before the results can be combined. Secondly, Batch Processing ensures that all tasks in a batch are completed before moving to the next batch. Lastly we can also use this for gaming when Synchronizing the state of multiple players before advancing to the next level or stage. (2024)

3. **Differentiate between parallelism and concurrency.**
   - According to GeeksforGeeks, Concurrency and Parallelism are foundational concepts in computer science, especially in multithreading and distributed systems. While they sound similar, they refer to different ways of managing multiple tasks. Understanding their distinction is crucial for designing efficient, scalable applications. (2025b)
   - When we say concurrency this means that we are dealing with multiple tasks at once, but instead of simultaneously solving them, by sharing time on the same processing resource.
   - As for the parallelism this is when we actually compute the problem simultaneously, this will first divide the problem into subproblems then that's when the solving happens.

Part B: Create C++ Code and show a solution that satisfies the given requirements below.

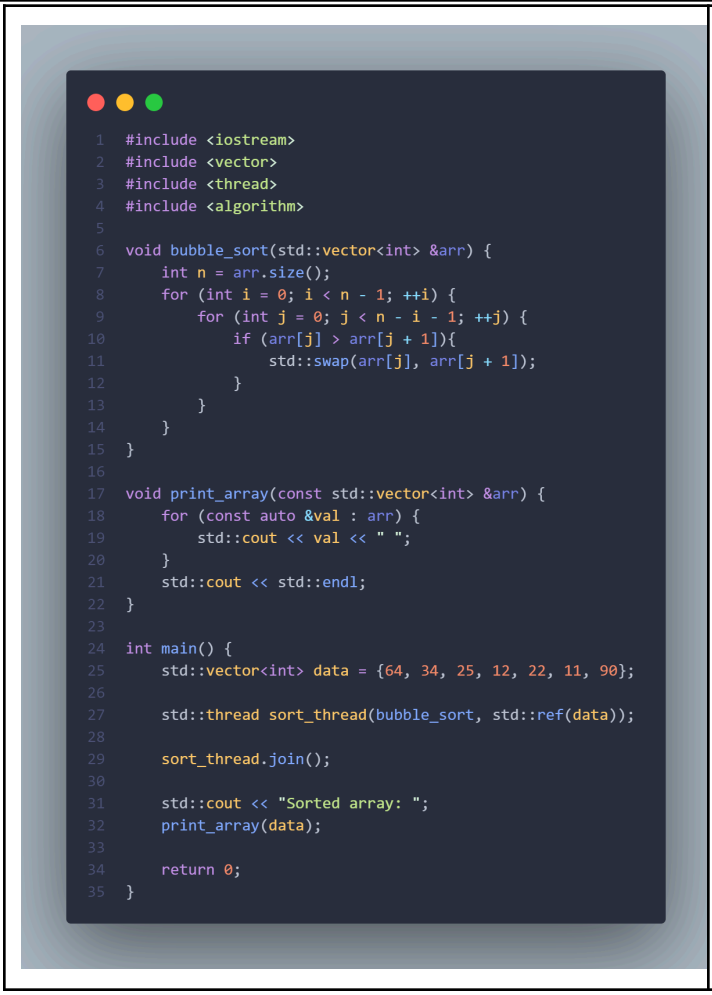
Output/Screen Shot:  **Code:**	**Analysis:** Firstly, I created a global variable and set it to 0, then I used that global variable to add it to the function. The function also has a const int &var so that there will be no modification that will happen. After that I created the main function to add the variable and the given val. I first named the object t1, t2, and t3, the t1 is 0 because I printed the actual value first, there's no joining function on that one. Since that is the instruction, as for the t2 it got the 60 because all the values are now joined together, but


`#include <iostream>`
`#include <thread>`
 	
`int global_var = 0;`
 	
`void add_func(const int &val){`
`  global_var += val;`
`}`
 	
`int main() {`
`  std::thread t1(add_func, 10);`
`  std::thread t2(add_func, 20);`
`  std::thread t3(add_func, 30);`
 	
`  std::cout<< "Joining t1: " << global_var << std::endl;`
`  t1.join();`
 	
`  std::cout<< "Joining t2: " << global_var << std::endl;`
`  t2.join();`
 	
`  std::cout << "Joining t3: " << global_var << std::endl;`
`  t3.join();`
`  return 0;`
`}`
	that is not the expected value; it should be 30 since the t1 and t2 should join together first. I think the problem lies on whether the thread is safe, since we have a global variable that will be shared among all the threads. I think the solution here is we have to use another library to make the global_var safe, since we have to add them 1 by 1, not altogether in just 1 run.

Part C: Use multi-threading with one of the algorithms previously developed in the course; provide an analysis of the result.

Output/Screen Shot:	Analysis:

`PS C:\Users\nikka\OneDrive\Desktop\c++>`
`'--stdout=Microsoft-MIEngine-Out-xlfs04c`
`Sorted array: 11 12 22 25 34 64 90`
`PS C:\Users\nikka\OneDrive\Desktop\c++>`
 	
Code:
	Here I first declared all the necessary libraries including the thread and the algorithm since I'll be using bubble sort for the algorithm. I picked bubble sort since it is easy to implement and easy to understand, these 2 concepts are easy to connect. After that I created a function for the bubble sort, in this function I defined the "n" as the size of the array and I used a vector here and & operator to avoid a copy of another value, this will just pass a reference of the given value on the main later.Then the first loop to run on the array then the second loop for comparison. After this function I created another function for printing the sorted array, in the printing_array I used the same parameter for the bubble sort but with the const, since we don't want to change anything on the array or any modification for it. As for the loop the const is also to avoid modification in the array, the auto is just for finding out what type is the element, and this will just again pass a reference to make it more efficient and no copy of that value this will run inside of the vector which is the "arr" After this I created the main and defined variable for the vector, this will be the arr that I used in the functions I created earlier. Then I created an object for the sorting thread this will print the bubble sort and pass the data to the arr the "ref" is a built in function for the thread and this will avoid to pass the copy and it should pass the original data to the arr. after this I used join function. then print the



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

void bubble_sort(std::vector<int> &arr) {
    int n = arr.size();
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] > arr[j + 1]){
                std::swap(arr[j], arr[j + 1]);
            }
        }
    }
}

void print_array(const std::vector<int> &arr) {
    for (const auto &val : arr) {
        std::cout << val << " ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> data = {64, 34, 25, 12, 22, 11, 90};

    std::thread sort_thread(bubble_sort, std::ref(data));

    sort_thread.join();

    std::cout << "Sorted array: ";
    print_array(data);

    return 0;
}
```

sorted array.

References:

Advantages and Disadvantages of multithreading in Java training course. (n.d.). Multisoft Virtual Academy. https://www.multisoftvirtualacademy.com/articles/common-advantages-and-disadvantages-of-multithreading-in-java-training-course

GeeksforGeeks. (2025c, October 3). Multithreading in C++. GeeksforGeeks. https://www.geeksforgeeks.org/cpp/multithreading-in-cpp/

GeeksforGeeks. (2025b, August 7). Difference between Concurrency and Parallelism. GeeksforGeeks. https://www.geeksforgeeks.org/operating-systems/difference-between-concurrency-and-parallelism/

Lucasnscr. (2024b, August 5). Multithreading and patterns. DEV Community. https://dev.to/lucasnscr/multithreading-and-patterns-4nmk

C. Conclusion & Lessons Learned

To conclude, I learned that multithreading enables us to run multiple tasks simultaneously, and this may enhance speed and responsiveness. Exchanging data among threads like the global variable in the example 1, this leads to the consideration of controlling the access to such data in order to prevent unexpected outcomes. When two threads attempt to modify the same variables simultaneously the outcome can be the incorrect final value to avoid this, we should know

how data is pass and how data works for threads. An example of this is when using & operator to pass a reference this helps make sure that the threads are operating on the real data, rather than a duplicate of it. It is here that comes in handy, to enable a thread to make alterations to the original variable directly. in it absence, alterations int he thread would not have an impact on the actual data. this idea is important particularly in sorting arrays, just like I did on part C, These concepts were applied to the parallelized bubble sort example in which multiple arrays were being sorted simultaneously. Each thread has its own array and thus did not interfere with each other and make the program more effective. Even though Bubble Sort cannot be used in parallel sorting of a single array, it allows illustrating how threads can be used independently of each other. Another syntax that we experimented with is such as, that iterates over a container, and uses a reference to optimize performance.

D. Assessment Rubric

E. External References