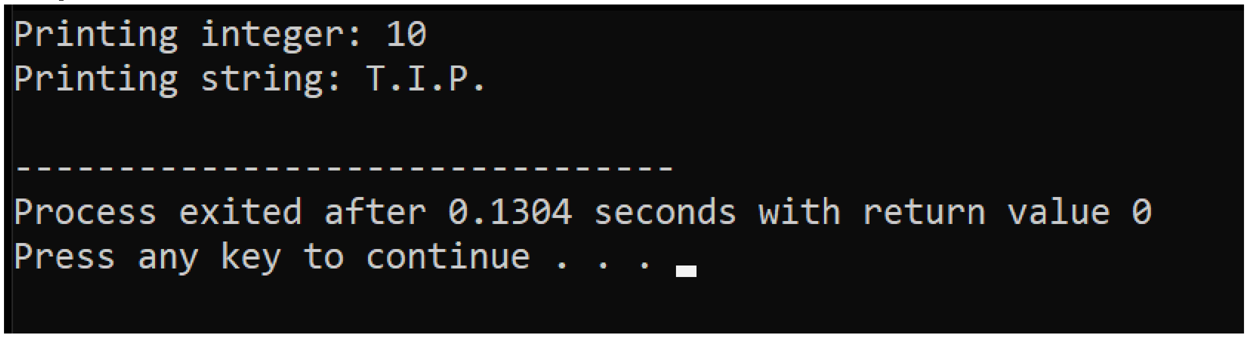
Hands-on Activity 13.1	
Parallel Algorithms and Multithreading	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 11/4/25
Section: CPE21S3	**Date Submitted:** 11/4/25
Name(s): Anastacio, Lester Arvid P.	**Instructor:** Engr. Jimlord Quejado
A. Output(s) and Observation(s)	

Part 1:

```
#i→ıcludc <iosticam>
#i→ıcludc <tmicad>
#i→ıcludc <stii→ıo>

void pii→ıt(i→ıt →ı, co→ıst std::stii→ıo hsti) (
   std::cout << "Pii→ıti→ıo i→ıtcoci: " << →ı << std::c→ıdl;
   std::cout << "Pii→ıti→ıo stii→ıo: " << sti << std::c→ıdl;
}

i→ıt mai→ı() (
   std::tmicad t1(pii→ıt, 10, "".I.P.");
   t1.joi→ı();
   ictui→ı 0;
}
```

Output:



```
Printing integer: 10
Printing string: T.I.P.

--------------------------------
Process exited after 0.1304 seconds with return value 0
Press any key to continue . . .
```

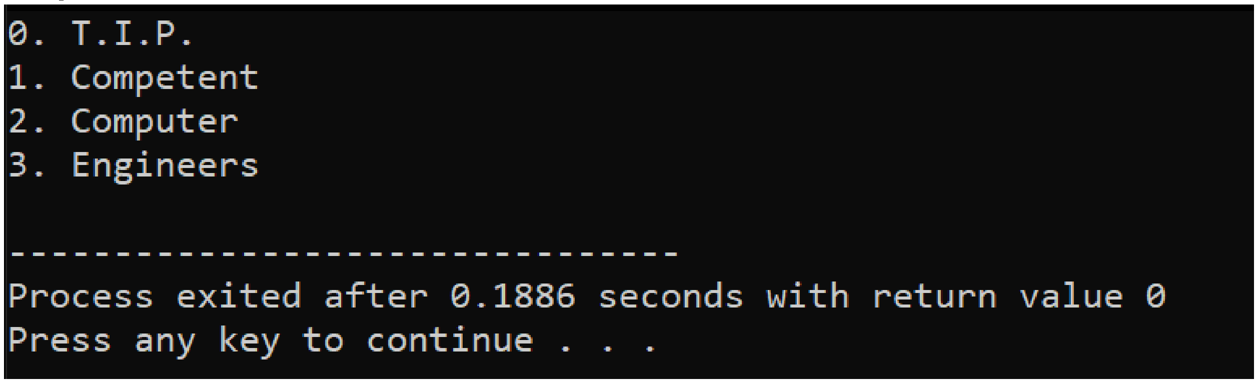
Analysis:

I noticed that the following code demonstrates the behavior of threads in the code and by using a thread to invoke a function that prints an integer and a string, the program makes a new thread, and then connects that thread to the main program so that the thread is finished with the function call before the main program ends..

Part 2:

```
#i→ıcludc <iosticam>
#i→ıcludc <tmicad>
#i→ıcludc <vcctoi>
#i→ıcludc <stii→ıo>
#i→ıcludc <mutc…>

std::mutc… cout_mutc…;

void pii→ıt(i→ıt →ı, co→ıst std::stii→ıo hsti) (
    std::stii→ıo mso = std::to_stii→ıo(→ı) + ". " + sti;

    std::lock_ouaid<std::mutc...> ouaid(cout_mutc...);
    std::cout << mso << std::c→ıdl;
}

i→ıt mai→ı() (
    std::vcctoi<std::stii→ıo> s = (
        "".I.P.",
        "Compctc→ıt",
        "Computci",
        "E→ıoi→ıccis"
    };

    std::vcctoi<std::tmicad>  tmicads;

    foi (i→ıt i = 0; i < s.si...c(); i++) (
        tmicads.cmplacc_back(pii→ıt, i, s[i]);
    }

    foi (auto htm : tmicads) (
        tm.joi→ı();
    }

    ictui→ı 0;
}
```

Output:



```
0. T.I.P.
1. Competent
2. Computer
3. Engineers

--------------------------------
Process exited after 0.1886 seconds with return value 0
Press any key to continue . . .
```

Analysis:

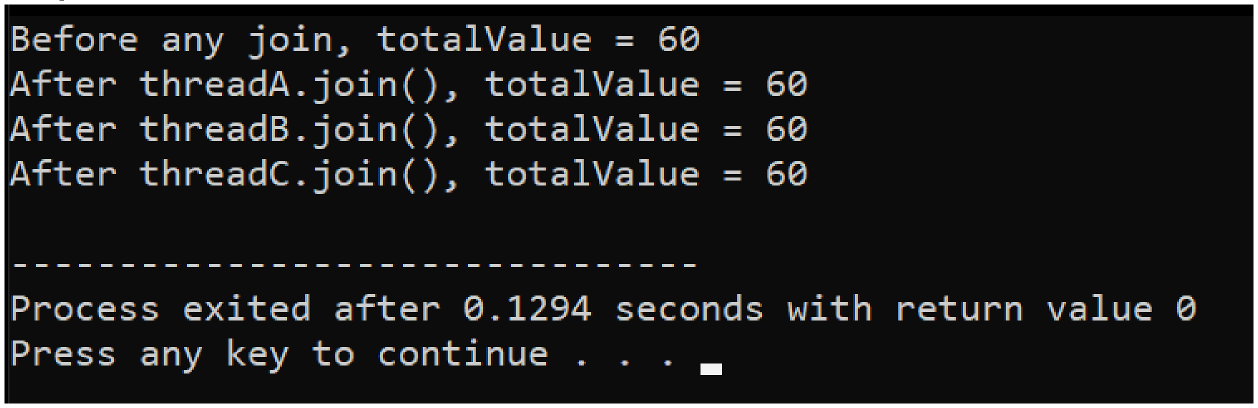
I observed that this code uses multiple threads and a mutex in which has a mutual exclusion lock to concurrently print different strings and also make sure that only one thread is accessing the output stream at a time thus avoiding mixed or jumbled output.

B. Answers to Supplementary Activity

Part B

```
#i→ıcludc <iosticam>
#i→ıcludc <tmicad>

i→ıt totalValuc = 0;

void i→ıcicasc(i→ıt →ıum) (
    totalValuc += →ıum;
}

i→ıt mai→ı() (
    std::tmicad tmicad6(i→ıcicasc, 10);
    std::tmicad tmicadB(i→ıcicasc, X0);
    std::tmicad tmicadC(i→ıcicasc, 30);

    std::cout << "Bcfoic a→ıQ joi→ı, totalValuc = " << totalValuc << std::c→ıdl;

    tmicad6.joi→ı();
    std::cout << "6ftci tmicad6.joi→ı(), totalValuc = " << totalValuc << std::c→ıdl;

    tmicadB.joi→ı();
    std::cout << "6ftci tmicadB.joi→ı(), totalValuc = " << totalValuc << std::c→ıdl;

    tmicadC.joi→ı();
    std::cout << "6ftci tmicadC.joi→ı(), totalValuc = " << totalValuc << std::c→ıdl;

    ictui→ı 0;
}
```

Output:



```
Before any join, totalValue = 60
After threadA.join(), totalValue = 60
After threadB.join(), totalValue = 60
After threadC.join(), totalValue = 60


--------------------------------
Process exited after 0.1294 seconds with return value 0
Press any key to continue . . .
```

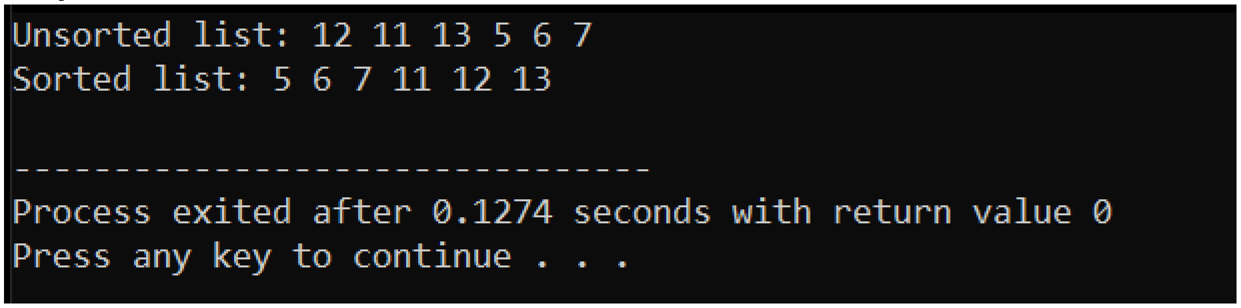
Analysis:

In this program it employs three threading units, each of which can change a shared variable in parallel by adding different values. It shows that the threads can operate concurrently, however, they will not necessarily yield the same results, since all threads can refer to the same variable without synchronization.

Part C

```
#i→ıcludc <iosticam>
#i→ıcludc <vcctoi>
#i→ıcludc <tmicad>

void combi→ıc(std::vcctoi<i→ıt>h data, i→ıt bcoi→ı, i→ıt midPoi→ıt, i→ıt c→ıd) (
    i→ıt lcftSi…c = midPoi→ıt - bcoi→ı + 1;
    i→ıt iiomtSi…c = c→ıd - midPoi→ıt;

    std::vcctoi<i→ıt> lcftPait(lcftSi…c), iiomtPait(iiomtSi…c);

    foi (i→ıt a = 0; a < lcftSi…c; a++)
        lcftPait[a] = data[bcoi→ı + a];
    foi (i→ıt b = 0; b < iiomtSi…c; b++)
        iiomtPait[b] = data[midPoi→ıt + 1 + b];

    i→ıt i = 0, j = 0, k = bcoi→ı;
    wmilc (i < lcftSi…c hh j < iiomtSi…c) (
        if (lcftPait[i] <= iiomtPait[j]) (
            data[k] = lcftPait[i];
            i++;
        } clsc (
            data[k] = iiomtPait[j];
            j++;
        }
        k++;
    }

    wmilc (i < lcftSi…c) (
        data[k] = lcftPait[i];
        i++;
        k++;
    }

    wmilc (j < iiomtSi…c) (
        data[k] = iiomtPait[j];
        j++;
        k++;
    }
}

void tmicadcdMciocSoit(std::vcctoi<i→ıt>h data, i→ıt bcoi→ı, i→ıt c→ıd) (
    if (bcoi→ı < c→ıd) (
        i→ıt midPoi→ıt = bcoi→ı + (c→ıd - bcoi→ı) / X;

        std::tmicad woikci1(tmicadcdMciocSoit, std::icf(data), bcoi→ı, midPoi→ıt);
        std::tmicad woikciX(tmicadcdMciocSoit, std::icf(data), midPoi→ıt + 1, c→ıd);

        woikci1.joi→ı();
        woikciX.joi→ı();

        combi→ıc(data, bcoi→ı, midPoi→ıt, c→ıd);
    }
}

i→ıt mai→ı() (
    std::vcctoi<i→ıt> →ıumbcis = (1X, 11, 13, S, ó, 7};

    std::cout << "U→ısoitcd list: ";
    foi (i→ıt →ı : →ıumbcis)
        std::cout << →ı << " ";
    std::cout << std::c→ıdl;

    tmicadcdMciocSoit(→ıumbcis, 0, →ıumbcis.si…c() - 1);

    std::cout << "Soitcd list: ";
    foi (i→ıt →ı : →ıumbcis)
        std::cout << →ı << " ";
    std::cout << std::c→ıdl;

    ictui→ı 0;
}
```

Output:



```
Unsorted list: 12 11 13 5 6 7
Sorted list: 5 6 7 11 12 13


--------------------------------
Process exited after 0.1274 seconds with return value 0
Press any key to continue . . .
```

Analysis:

This code leverages multithreading to execute merge sort, which allows the sorting of numbers to occur at a faster rate by distributing the sorting task between threads. The output illustrates the sorting of each segment of the list separately and a merge of each segment into the final sorted list.

C. Conclusion & Lessons Learned

During this activity I learned to make visible of the efficiency advantages of multithreading in which I used to optimize the Merge Sort algorithm and clearly see the performance improvement. Even though thread synchronization and the initial setup were quite challenging, and thus I had to figure it out by myself through research and trial, in the end, it gave me a deep understanding of the logic of concurrent programming. I have got a great deal of practical experience and my level of confidence in the execution of threads in C++ has risen enormously.

D. Assessment Rubric

E. External References

GeeksforGeeks. (2025d, July 23). *Parallel algorithm models in parallel computing*. GeeksforGeeks.

https://www.geeksforgeeks.org/mobile-computing/parallel-algorithm-models-in-parallel-computing/

GeeksforGeeks. (2025n, October 3). *Multithreading in C++*. GeeksforGeeks.

https://www.geeksforgeeks.org/cpp/multithreading-in-cpp/