DEPARTMENT OF CYBER SECURITY

Parallel and Distributed Computing Lab	
SUBMITTED BY	211042 Noman Masood Khan A
SEMESTER	Seven
SUBJECT	Parallel and Distributed Computing
SUBMITTED TO	Sir Osama

Lab Task 1: Basic Multithreading in C++

CODE:

```
#include <iostream>
#include <thread>

// Function to print numbers from 1 to 5
void printNumbers() {
    for (int i = 1; i <= 5; i++) {
        std::cout << i <<"\t";
    }
}

// Function to print squares of numbers from 1 to 5
void printSquares() {
    for (int i = 1; i <= 5; i++) {
        std::cout << i * i <<"\t";
    }
}

int main() {
    // Create two threads
    std::thread t1(printNumbers);
    std::thread t2(printSquares);

    // Wait for both threads to finish
    t1.join();
    t2.join();

    return 0;
}
```

```
C:\Users\HP\OneDrive\Desktop\Question 1.exe
1       2       3       4       5       1       4       9       16      25
--------------------------------
Process exited after 6.44 seconds with return value 0
Press any key to continue . . .
```

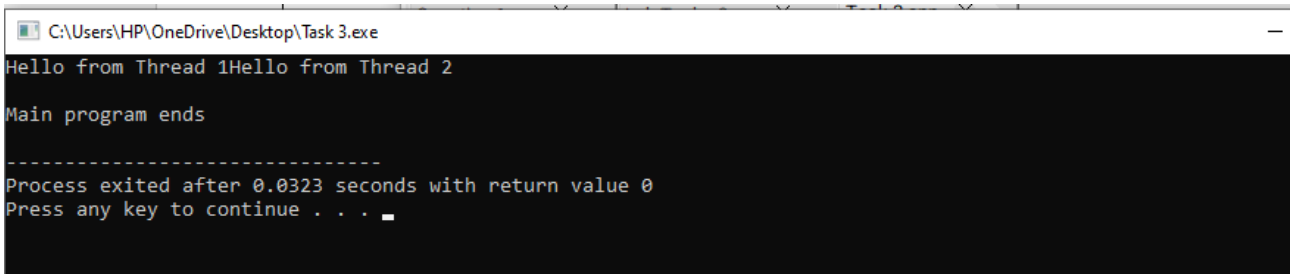
Lab Task 2: Thread Synchronization Using Mutex

```
#include <iostream>
#include <thread>
#include <mutex>

std::mutex mtx; // mutex for protecting shared variable
int sharedVariable = 0; // shared variable

void incrementVariable() {
    for (int i = 0; i < 100; i++) {
        std::lock_guard<std::mutex> lock(mtx); // acquire lock
        sharedVariable++; // increment shared variable
    }
}

int main() {
    std::thread t1(incrementVariable); // create thread 1
    std::thread t2(incrementVariable); // create thread 2

    t1.join(); // wait for thread 1 to finish
    t2.join(); // wait for thread 2 to finish

    std::cout << "Final value of shared variable: " << sharedVariable << std::endl;

    return 0;
}
```

```
Select C:\Users\HP\OneDrive\Desktop\Lab Tasks 2.exe
Final value of shared variable: 200

--------------------------------
Process exited after 3.481 seconds with return value 0
Press any key to continue . . .
```

Lab Task 3: Multithreading with Lambda Expressions

```
#include <iostream>
#include <thread>

int main() {
    // Create thread 1 using a lambda expression
    std::thread t1([&]() {
        std::cout << "Hello from Thread 1" << std::endl;
    });

    // Create thread 2 using a lambda expression
    std::thread t2([&]() {
        std::cout << "Hello from Thread 2" << std::endl;
    });

    // Wait for both threads to finish
    t1.join();
    t2.join();

    std::cout << "Main program ends" << std::endl;

    return 0;
}
```



Lab Task 4: Multithreading with Sorting

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

// Function to sort an array in ascending order
void sortAscending(std::vector<int>& arr) {
    std::sort(arr.begin(), arr.end());
}

// Function to sort an array in descending order
void sortDescending(std::vector<int>& arr) {
    std::sort(arr.begin(), arr.end(), std::greater<int>());
}

int main() {
    // Initialize arrays
    std::vector<int> ascendingArray = {5, 2, 9, 1, 5, 6};
    std::vector<int> descendingArray = {3, 8, 2, 4, 7, 5};

    // Create threads for sorting
    std::thread thread1(sortAscending, std::ref(ascendingArray));
    std::thread thread2(sortDescending, std::ref(descendingArray));

    // Wait for both threads to complete
    thread1.join();
    thread2.join();

    // Print sorted arrays
    std::cout << "Sorted array in ascending order: ";
    for (const auto& num : ascendingArray) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    std::cout << "Sorted array in descending order: ";
    for (const auto& num : descendingArray) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

```
C:\Users\HP\OneDrive\Desktop\Task 4.exe
Sorted array in ascending order: 1 2 5 5 6 9
Sorted array in descending order: 8 7 5 4 3 2

--------------------------------
Process exited after 0.02938 seconds with return value 0
Press any key to continue . . .
```

Lab Task 5: Thread Communication Using Condition Variables

```
#include <iostream>
#include <thread>
#include <condition_variable>
#include <mutex>

std::condition_variable cv;
std::mutex mtx;
bool x_turn = true;
int count = 0;

void print_x() {
    for (int i = 0; i < 5; i++) {
        std::unique_lock<std::mutex> lock(mtx);
        cv.wait(lock, [&]{ return x_turn; });
        std::cout << "X ";
        x_turn = false;
        cv.notify_all();
        ++count;
    }
}

void print_o() {
    for (int i = 0; i < 5; i++) {
        std::unique_lock<std::mutex> lock(mtx);
        cv.wait(lock, [&]{ return !x_turn; });
        std::cout << "O ";
        x_turn = true;
        cv.notify_all();
        ++count;
    }
}

int main() {
```

```
    std::thread t1(print_x);
    std::thread t2(print_o);

    t1.join();
    t2.join();

    std::cout << std::endl;
    std::cout << "Main program ends" << std::endl;

    return 0;
}
```