1. (a) **6 marks** Rewrite the following code using only if, labels, and **goto**; no **else** or compound-statements “{}”.

i. if ( \( C \) ) {
    ... // then-clause
} else {
    ... // else-clause
}

ii. **while** ( \( C \) ) {
    ... // loop-body
}

(b) **2 marks**

i. How does a **multi-exit loop** differ from **static multi-level exit**?

ii. How does a **static multi-level exit** differ from **dynamic multi-level exit**?

(c) **1 mark** True or False: programs should never use flag variables.

(d) **2 marks** Give a common C example where return code and status flag are combined.

(e) **2 marks** Name the properties of a Sequel call/return that are different from a routine-pointer call/return.

(f) **2 marks** When can a destructor not raise an exception and why?

(g) **2 marks** Explain what is meant by C++ zero-cost exceptions, and why is it misleading (fake news)?

2. (a) **1 mark** What is the purpose of a **communication variable** in a coroutine?

(b) **1 mark** What does it mean to find the “Zen” of the coroutine?

(c) **2 marks** Discuss the issue of stack size with respect to \( \mu \)C++ coroutines, and give an example of how to deal with this issue.

(d) **2 marks** Why does the type signature of the coroutine **main** member not take any parameters or return a result?

(e) **1 mark** True or False: Coroutines are just a strange idea in \( \mu \)C++.

(f) **5 marks** Given the following coroutine:

```cpp
Coroutine Fc {
    void main() {
        ... mem();    // 1
        ... resume(); // 2
        ... suspend(); // 3
        ... return;   // 4
    }

    public:
    void mem() { resume(); }
};
```

Explain the control flow that occurs at points 1, 2, 3, and 4.

(g) **1 mark** What is the purpose of an **_Enable** block in a coroutine?
3. (a) **2 marks** Explain the difference between preemptive and non-preemptive scheduling.

(b) **3 marks** Explain why `i += 1` on shared variable `i` is an unsafe operation in a concurrent program.

(c) **2 marks** Explain the threading model in μC++?

(d) **1 mark** Explain why COBEGIN/COEND and COFOR are classified as an implicit concurrent-systems.

(e) **1 mark** What kind of synchronization is necessary for divide-and-conquer problems?

(f) **3 marks** The following is a self-testing critical section. Explain how it works.

```c
void CriticalSection() {
    ::CurrTid = &uThisTask();
    for ( int i = 1; i <= 100; i += 1 ) {
        // work
        if ( ::CurrTid != &uThisTask() )
            abort( "interference" );
    }
}
```

(g) **2 marks** For software solutions for mutual exclusion, explain unbounded and bounded overtaking in terms of declaring and retracting intent.

(h) **2 marks** The following is Peterson’s software-solution for mutual exclusion:

```c
me = WantIn;
::Last = &me;
while ( you == WantIn && ::Last == &me ) {}
CriticalSection();
me = DontWantIn;
```

i. Explain why there is no indefinite postponement (satisfies rule 4).
ii. Explain why there is no starvation (satisfies rule 5).

4. (a) **2 marks** Explain the difference between a no yield and yield spin lock.

(b) **2 marks** Given:

```
yieldNoSchedule( lock );
```

i. Explain why `yieldNoSchedule` is different from `yield`.
ii. Explain why `yieldNoSchedule` takes an argument.

(c) **2 marks** Explain the difference between barging avoidance and prevention.

(d) **3 marks** Write the code pattern to implement a cyclic barrier using a coordinator task and N worker tasks.

(e) **6 marks** Using a binary semaphore:

i. **3 marks** show the pattern to synchronize S1 before S2,
ii. **3 marks** show the pattern to provide mutual exclusion for a critical section C.
5. **17 marks** Write a *semi-coroutine* with the following public interface (you may only add a public destructor and private members):

```cpp
_Event Eof {};
_Coroutine Compact {
    char ch; // character passed by cocaller
    void main(); // YOU WRITE THIS ROUTINE
    public:
        void next( char c ) {
            ch = c;
            resume();
        }
};
```

Compact removes all spaces and tabs (isblank) from the start and end of lines, collapses multiple spaces and tabs within a line into a single space, and eliminates empty lines (consisting only of whitespace). Otherwise, it prints the characters. Lines are delimited by the newline character (‘\n’). The exception Eof is raised at coroutine Compact when there are no more characters, indicating the coroutine must terminate.

For example, the input file:

```
    start of text
    more text
    last text
```

is converted into:

```
    start of text
    more text
    last text
```

Write ONLY Compact::main, do NOT write a main program that uses it! **No documentation or error checking of any form is required.**

**Note:** Few marks will be given for a solution that does not take advantage of the capabilities of the coroutine, i.e., you must use the coroutine’s ability to retain data and execution state.

6. **30 marks** Divide and conquer is a technique that can be applied to certain kinds of problems. These problems are characterized by the ability to subdivide the work across the data, such that the work can be performed independently on the data. In general, the work performed on each group of data is identical to the work that is performed on the data as a whole. What is important is that only termination synchronization is required to know the work is done; the partial results can then be processed further.

Write a COMPLETE \texttt{\mu C++} program to efficiently check if a matrix of size $N \times N$ is a diagonally-symmetric matrix. (Notice, the matrix must be square and assume $N \leq 10$.) A diagonally-symmetric matrix has identical values along the diagonal and is equal to its transpose, i.e., $M = M^T$. That is, given $A = a_{i,j}$, then $a_{0,0} = a_{i,i}$ and $a_{i,j} = a_{j,i}$, for all indices $i$ and $j$. The following are all diagonally-
Solve the problem using task objects not the COFOR statement. Create one task per row of the matrix to concurrently check the values of that particular row for the diagonal-symmetric property. Each checking task has the following interface (you may only add a public destructor and private members):

```cpp
_Event NotDS {}; // concurrent exception
_Task DiagSymmetric { // check row of matrix
    // YOU ADD HERE
    void main(); // YOU WRITE THIS ROUTINE
}

public:
    _Event Stop {}; // concurrent exception
    DiagSymmetric(const int M[,][10], // matrix to check for diagonally symmetric
                   const int row, // row to be checked
                   const int cols, // number of columns in row
                   uBaseTask & pgmMain, // contact when not diagonally symmetric
    );
};
```

The program main reads from standard input the matrix dimension \( N \times N \), where \( 1 \leq N \leq 10 \), declares any necessary matrix, arrays and variables, reads (from standard input) and prints (to standard output) the matrix, concurrently checks the matrix values in each row, and prints a message to standard output if the matrix is or is not diagonally symmetric. **No documentation or error checking of any form is required.**

As an optimization, the moment a DiagSymmetric task determines a row is not diagonally symmetric, it raises the concurrent exception NotDS at the pgmMain and then returns, and when the program main receives this concurrent exception, it raises exception DiagSymmetric::Stop at any non-deleted DiagSymmetric tasks. When the concurrent Stop exception is propagated in a DiagSymmetric task, it stops performing the diagonally-symmetric check and returns.

An example of input for the above is:

```
4 // matrix dimensions
7 2 3 4 // matrix values
2 7 4 5
3 4 7 6
4 5 6 7
```

(The phrases “matrix dimensions” and “matrix values” do not appear in the input.) In general, the input format is free form, meaning any amount of white space may separate the values.

Example outputs are:
| 7, 2, 3, 4,   | original matrix | 7, 2, 3, 2,   | original matrix |
| 2, 7, 4, 5,   | matrix is diagonal symmetric | 2, 7, 4, 5,   | matrix is not diagonal symmetric |
| 3, 4, 7, 6,   |                 | 3, 4, 6, 6,   |                 |
| 4, 5, 6, 7,   |                 | 4, 5, 6, 7,   |                 |

(The phrase “original matrix” does not appear in the output.) Note, the comma is a terminator not a separator.