UNIVERSITY OF WATERLOO

Midterm Examination
Fall 2025

Computer Science 343
Concurrent and Parallel Programming
Sections 001, 002

Duration of Exam: 2 hours
Number of Exam Pages (including cover sheet): 6
Total number of questions: 5
Total marks available: 114

CLOSED BOOK, NO ADDITIONAL MATERIAL ALLOWED

Instructor: Peter Buhr

October 29, 2025

1. (a) **3 marks** Rewrite the following code fragment using only **if**, labels, and **goto**s; no **else** or compound-statement "{}".

   ```
   do {
       S;
   } while ( C );
   ```

   (b) **3 marks** Using only control structures, i.e., no helper routines, rewrite the following nested **if** statements so S4 is not duplicated.

   ```
   if ( C1 ) {
       S1;
       if ( C2 ) {
           S2;
           if ( C3 ) {
               S3;
           } else
               S4;
       } else
           S4;
   } else
       S4;
   ```

   (c) **1 mark** For static multi-level exit, why is it good practice not to use unlabelled **break** statements?

   (d) **2 marks** Explain the data-structure created by a VLA (variable length array) and where its storage is allocated

   (e) **2 marks** uArrayPtr(S, a, 42) and unique_ptr<S> a[42] both dynamically create an array of objects allowing post-declaration initialization. What is difference between these two mechanisms?

   (f) **2 marks** Explain *stack smashing*.

   (g) **7 marks** Given the following fixup code, convert it to the equivalent resumption code.

   ```
   void f( ..., void (*fixup)( ... ) ) {
       if ( ... ) fixup( ... );
       // control returns here
   }
   void fixup1( ... ) { /* handler 1 */ }
   void fixup2( ... ) { /* handler 2 */ }
   int main() {
       f( ..., fixup1 );
       f( ..., fixup2 );
   }
   ```

2. (a) **1 mark** Explain why most programming languages only search the **catch** clauses at the end of a **try** block *once*, i.e., once E1 is caught, the catch clause for E2 cannot catch the next throw.

   ```
   try { ... throw E1;
   } catch( E1 ) { ... throw E2; }
   } catch( E2 ) { ... }
   ```

   (b) **2 marks** Name the kind of return performed by a **_CatchResume** handler. Name the kind of return performed by a **catch** handler.

   (c) **3 marks** Explain the propagation of exception R in the following:

```
_Exception R {};
void rtn() {
    try {
        _Resume R();
    } catch( R & ) { ... }
}
```

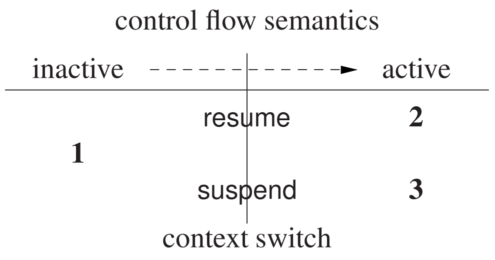
(d) **7 marks** Given the following code fragment:

```
B1  {
B2      try {
B3          try {
B4              try {
B5                  {
B6                      try {
                            ... throw (resume) E5(); ...
C1                      } catch( E7 ) { ... }
C2                        catch( E8 ) { ... }
C3                        catch( E9 ) { ... }
                    }
C4              } catch( E4 ) { ... }
C5                catch( E5 ) { ... throw; ... }
C6                catch( E6 ) { ... }
C7          } catch( E3 ) { ... }
C8      } catch( E5 ) { ... }
C9        catch( E2 ) { ... }
    }
```

i. How many unguarded and guarded blocks are on the stack?
ii. How many throws occur?
iii. How many catch clauses are examined across all the throws?
iv. Which catch clause handles the exception?
v. Which block does the handler catch-clause transfer to for *resumption*, *retry*, and *termination*?

(e) **1 mark** Where is a μC++ coroutine stack allocated?

(f) **3 marks** Name the coroutine that becomes inactive/active at locations **1**, **2**, and **3**, below.



(g) **2 marks** What property is necessary for full coroutining; why is it difficult to create this property?

3. (a) **3 marks** Programs P1 and P2 both take 10 seconds *user time* when run on 1 CPU. P1 is 100% parallel and takes 2.5 seconds *real time* on 4 CPUs. P2 is 20% sequential and 80% parallel. Compute the speedup of P2 on 4 CPUs.

(b) **3 marks** Draw a picture of the *scatter/gather* concurrency pattern.

(c) **5 marks** Recursion is used (below) to linearly create a *dynamic* number of work units using COBEGIN/COEND. Write a tree routine to create work units exponentially (base 2), where tree is called with a power of 2.

```
void loop( unsigned int N ) {
    if ( N != 0 ) {
        COBEGIN                              // linear creation
            BEGIN p( ... ); END
            BEGIN loop( N - 1 ); END         // recursively create more work units
        COEND                                // wait for work units to complete
    }
}
```

(d) **3 marks** The following is the entry protocol code for the prioritized retract intent algorithm for mutual exclusion.

```
if ( priority == HIGH ) {
    me = WantIn;
    while( you == WantIn ) {}
} else {
    while( true ) {
        me = WantIn;
      if ( you == DontWantIn ) break;
        me = DontWantIn;
        while( you == WantIn ) {}
    }
}
```

i. What is the problem with the protocol?
ii. Explain how the problem arises.
iii. How does Dekker's algorithm overcome the problem?

(e) **1 mark** Are barrier locks for synchronization or mutual exclusion?

4. **20 marks** Write a *semi-coroutine* with the following public interface (you may only add a public destructor and private members):

```
_Coroutine Phone {
    char ch;                    // character passed by cocaller
    void main();                // YOU WRITE THIS ROUTINE
  public:
    enum { EOT = '\003' };      // end of text
    _Exception Match {};        // characters form a valid string in the language
    _Exception Error {};        // last character results in string not in the language
    void next( char c ) {
        ch = c;
        resume();
    }
};
```

which verifies a string of characters constitutes a valid North American telephone number. The string is described by the following grammar:

phoneno : $area_{opt}$ *trunk* *dash* *number*
area : "(" *3-digit-number* ")"
trunk : *3-digit-number*
dash : "–"
number : *4-digit-number*

where the quotation marks are metasymbols and not part of the described language, and $_{opt}$ means optional (0 or 1). The following are some valid and invalid phone numbers:

valid strings	invalid strings
(876)343-8760	789 6543
456-9807	(88)345-8790
786-5555	(888)45-8790
(800)555-1212	(888)345-879

Assume the C library routine isdigit; isdigit(c) returns true if c is a digit;

After creation, the coroutine is resumed with a series of characters (one at a time), plus an EOT (end-of-text) character after all characters are passed. The coroutine accepts characters until:

- the characters form a valid string in the language, and it then raises the exception Phone::Match at the last resumer;
- the last character results in a string not in the language, it then raises the exception Phone::Error at the last resumer.

After the coroutine raises a Match or Error exception, it must terminate; sending more characters to the coroutine after this point is undefined. **Marks will be deducted for duplicate code.**

Write **ONLY** Phone::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.

5. 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.

   A *Schmilblick* matrix of size $N \times M$ contains at least two *Schmilblick* values (here -1) in each row.

$$\underbrace{\begin{pmatrix} 1 & -\mathbf{1} & 3 & 4 & -\mathbf{1} \\ -\mathbf{1} & 1 & 4 & -\mathbf{1} & 6 \\ 3 & -\mathbf{1} & -\mathbf{1} & 6 & -\mathbf{1} \\ -\mathbf{1} & 6 & 7 & -\mathbf{1} & 1 \\ 4 & -\mathbf{1} & -\mathbf{1} & 1 & -\mathbf{1} \end{pmatrix}}_{Schmilblick} \quad \underbrace{\begin{pmatrix} 1 & -\mathbf{1} & 3 & 4 & -\mathbf{1} \\ 2 & 1 & 4 & -\mathbf{1} & 6 \\ 3 & -\mathbf{1} & -\mathbf{1} & 6 & -\mathbf{1} \\ -\mathbf{1} & 6 & 7 & -\mathbf{1} & 1 \\ 4 & 25 & 6 & 1 & 8 \end{pmatrix}}_{Non-Schmilblick}$$

   (a) **5 marks** Write a sequential routine to *efficiently* check if a row of a matrix is a Schmilblick row.

```
bool schmilblickCheck( const int row[ ], int cols, int schmilblick );
```

   where row is the matrix row, cols is the columns in the row, and schmilblick is the Schmilblick value. The function returns true if the row is a Schmilblick and false otherwise.

   (b) **3 marks** Using routine schmilblickCheck and the following declarations, write a COFOR statement to concurrently check if each row of matrix M has the Schmilblick property.

```
cin >> schmilblick >> rows >> cols;
bool found = true;                    // assume found Schmilblick property
int M[rows][cols];
COFOR( . . .                          // YOU WRITE THIS STATEMENT
    . . .
);
```

(c) **7 marks** Using routine schmilblickCheck, write a message and actor to concurrently check if each matrix row has the Schmilblick property.

```
struct WorkMsg : public uActor::Message {
    // YOU WRITE THIS TYPE
};
_Actor Schmilblick {
    // YOU WRITE THIS TYPE
};
```

(d) **5 marks** Using declarations for found and M above, write the code to start/stop the actor system, create the Schmilblick actors on the stack, and send them a WorkMsg message.

(e) **7 marks** Both the COFOR and actor solutions continue checking for Schmilblick rows even if a non-Schmilblick row is found. For the task solution, if a non-Schmilblick row is found by a task, it raises the concurrent exception NotSchmilblick at the pgmMain and the task returns (terminates). If a Schmilblick task receives the concurrent exception Schmilblick::Stop from the program main, it stops checking and returns (terminates).

Write a task to concurrently check if each matrix row has the Schmilblick property.

```
_Task Schmilblick {                        // check row of matrix
    // YOU ADD MEMBERS HERE
    void main() {
        // YOU WRITE THIS MEMBER
    }
  public:
    _Exception Stop {};                    // concurrent exceptions
    _Exception NotSchmilblick {};          // failed Schmilblick test
    Schmilblick(                           // YOU WRITE THIS MEMBER
        int r,                             // row number
        const int row[ ],                  // matrix row
        int cols,                          // columns in row
        int schmilblick,                   // schmilblick value
        uBaseTask & pgmMain                // contact when Schmilblick not found
    );
};
```

(f) **13 marks** Using Schmilblick task, complete the program main below by creating the Schmilblick tasks in the heap, and then delete each task until all tasks finish, implying a Schmilblick matrix or a concurrent NotSchmilblick exception is caught. If a concurrent NotSchmilblick exception is caught, the program main raises exception Schmilblick::Stop at any non-deleted Schmilblick tasks.

```
int main() {
    bool found = true;
    int rows, cols;
    cin >> rows >> cols;
    int M[rows][cols];
    // assume data read into matrix M
    // YOU WRITE CODE TO CHECK M, HANDLE Schmilblick::NotSchmilblick EXCEPTIONS
    // AND RAISE Schmilblick::Stop EXCEPTIONS.
}
```

No documentation or error checking of any form is required.