Midterm Examination
Winter 2017

Computer Science 343
Concurrent and Parallel Programming
Sections 001

Duration of Exam: 1 hour 50 minutes
Number of Exam Pages (including cover sheet): 6
Total number of questions: 6
Total marks available: 103

CLOSED BOOK, NO ADDITIONAL MATERIAL ALLOWED

Instructor: Peter Buhr
March 1, 2017
1. (a) **2 marks** Rewrite the following `while` loop using only labels, `if` and `goto`s.

```c
while ( C ) {

```

(b) **1 mark** For static multi-level exit, why is it good practice to label all exits?

(c) **1 mark** Why are `flag variables` the variable equivalent to a `goto`?

(d) **1 mark** Why does `modularization` (refactoring) cause problems with multi-level exit?

(e) **2 marks** Explain why a `label variable` for nonlocal transfer must be a tuple of two values.

(f) **2 marks** Knowing how C `setjmp/longjmp` performs a nonlocal transfer, would C++ destructors be executed for objects on the stack during the transfer? Give a reason for your answer.

(g) **4 marks** Explain `static/dynamic call` and `static/dynamic return`.

2. (a) **6 marks** Given the following coroutine:

```c
_Coroutine Fibonacci {
    int fn;
    void main() {
        int fn1, fn2;
        fn = 0; fn1 = fn;
        suspend();
        fn = 1; fn2 = fn1; fn1 = fn;
        suspend();
        for ( ;; ) {
            fn = fn1 + fn2; fn2 = fn1; fn1 = fn;
            suspend();
        }
    }
}
```

(b) **1 mark** Explain the purpose of `communication variables` in coroutines.

(c) **3 marks** Rewrite the following coroutine code so it has better coroutine “Zen”:

```c
for ( int i = 0; i < 10; i += 1 ) {
    if ( i % 2 == 0 ) // even ?
        even += digit;
    else
        odd += digit;
    suspend();
}
```

(d) **3 marks** Both `suspend()` and `resume()` members are composed of two parts with respect to the actions taken in inactivating and reactivating coroutines. Explain exactly which coroutine is `inactivated` and which is `reactivated` for each member.

(e) **1 mark** Give an advantage for restricting the placement of `resume()` to within a coroutine’s member routines.
3. (a) **3 marks** Concurrent systems can be divided into three major types with respect to how concurrency is introduced into a program. Briefly, explain each type.

(b) **2 marks** What does it mean for a thread scheduling-strategy to be preemptive? non-preemptive?

(c) **2 marks** Give two reasons why program speedup does not scale linearly with the number of CPUs? Name them, do not explain them.

(d) **4 marks** When introducing concurrency into a programming language, what are the basic 4 mechanisms that must be supported? Name them, do not explain them.

(e) **4 marks** In software tournament algorithms

![Diagram](attachment:diagram.png)

explain a failure scenario if the binary-algorithm nodes are not released in reverse acquisition order?

(f) **1 mark** What basic special property do all atomic hardware-instructions have?

4. (a) **2 marks** How does a yielding spin-lock improve performance over a non-yielding spin-lock?

(b) **2 marks** Explain the difference between single and multiple-acquisition mutex-locks.

(c) **1 mark** Using an example, show how to protect output in µC++ so characters from multiple tasks are not intermixed or cause failures.

(d) **2 marks** Explain the high-level purpose of a barrier lock.

(e) **2 marks** A counting semaphore is a multi-value lock. Using an example, explain what it means to have a critical section protected by a lock with many values.

(f) **8 marks** The following precedence graph shows the optimal concurrency possible for a series of statements S1..S6:

![Diagram](attachment:diagram2.png)

Code the statements using only one COBEGIN and COEND in conjunction with binary semaphores using P and V to achieve the concurrency of the precedence graph. Use pseudo-code for this problem, not µC++. Use BEGIN and END to make several statements into a single statement and show the initial value (0/1) for all semaphores. Name your semaphores Ln, e.g., L1, L2, ..., to simplify marking.
5. **17 marks** Write a *semi-coroutine* filter with the following public interface (you may only add a public destructor and private members):

```c++
Coroutine Filter {
    protected:
    Event Eof {}; // no more characters
    Filter *next; // next filter in chain
    unsigned char ch; // communication variable

    public:
    void put( unsigned char c ) {
        ch = c;
        resume();
    }
};

Coroutine Contractions : public Filter {
    bool search( const char *key ) {
        static const char *prefix[] = {
            "are", "could", "did", "does", "had", "have", "is", "must", "should", "was", nullptr
        };

        for ( unsigned int i = 0; prefix[i]; i += 1 ) {
            if ( strcmp( key, prefix[i] ) == 0 ) return true;
        }
        return false;
    }

    void scanword( char word[], bool send ) {
        // YOU WRITE THIS ROUTINE
    }

    public:
    Contractions( Filter &f ) {
        next = &f;
    }
};
```

which receives a sequence of characters, and combines pairs of words into English contractions, e.g., “are not” becomes “aren’t”. Specifically, if a prefix word (see above) is followed by a single blank and the string “not”, it is converted into the prefix followed by “n’t”. Member search returns true if the word is a prefix and false otherwise. Member scanword scans for consecutive alpha characters placing them into the parameter word and terminating word with the ‘\0’. When the parameter send is true, the word characters are also forwarded to the next filter and false means the word characters are not forwarded. Member main, in general, scans for a word, checks if it is a prefix followed by a blank, and if so, scans for the word “not”, and if so, replaces it with “n’t”. Assume the C library routine isalpha( c ), which returns true if c is a letter, and false otherwise.

For example, the input file:
I've been to sea Billy.
I am not going, is not a good show.
I stuttered are are are not was was not must not must ?not must not go

is converted into:

I've been to sea Billy.
I am not going, isn't a good show.
I stuttered are are aren't was wasn't mustn't must ?not mustn't go

Write ONLY Contractions::scanword and the designated portion of Contractions::main; do NOT write any other filters or main program that uses them! 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. 26 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 µC++ program to efficiently check if all the rows of a matrix of size $N \times M$, where $M$ is odd, are palindromic. For example, in:

$$
\begin{bmatrix}
1 & 2 & 3 & 2 & 1 \\
2 & 1 & 4 & 1 & 2 \\
3 & 4 & 1 & 4 & 3 \\
4 & 5 & 6 & 5 & 4 \\
5 & 6 & 7 & 6 & 5 \\
\end{bmatrix}
$$

all the rows read the same forward or backwards, i.e., mirror image pivoted around the centre column. The matrix is checked concurrently along its rows. Each task has the following interface (you may only add a public destructor and private members):

```cpp
bool stop = false; // global variable: true => stop all work
_Task Palindrome {
    ...
    // YOU ADD HERE
    void main(); // YOU WRITE THIS ROUTINE
    public:
        _Event Stop {}; // concurrent exception
        Palindrome( const int row[], // one row of the matrix
                     const int cols // number of columns in row
                 ); // YOU WRITE THIS ROUTINE
    }
};
```

The program main reads from standard input the matrix dimensions $(N \times M)$, 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 palindromic. No documentation or error checking of any form is required.
As an optimization, the global (flag) variable stop is set when a task finds a non-palindromic row (may be set multiple times), and the program main performs a resumption raise of exception Stop at any non-deleted tasks once the stop flag indicates the matrix is non-palindromic. When the concurrent Stop exception is propagated, each checking task stops performing the palindromic check and terminates.

An example of the input for the program is:

```
5 5

1 2 3 2 1
2 1 4 1 2
3 4 1 4 3
4 5 6 5 4
5 6 7 6 5
```

(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:

```
1, 2, 3, 2, 1, original matrix
2, 1, 4, 1, 2,
3, 4, 1, 4, 3,
4, 5, 6, 5, 4,
5, 6, 7, 6, 5,

matrix is palindromic

1, 2, 3, 2, 1, original matrix
2, 1, 4, 8, 2,
3, 4, 1, 4, 3,
4, 5, 6, 5, 4,
5, 6, 7, 6, 5,

matrix is not palindromic
```

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