Midterm Answers – CS 343 Fall 2025

Instructor: Peter Buhr

October 29, 2025

These are not the only answers that are acceptable, but these answers come from the notes, assignments, or lectures.

1. (a) **3 marks**

```
1   SDW: ;
1       S;
1   if ( C ) goto SDW;
```

   (b) **3 marks**

```
1   C: {                                  {
       if ( C1 ) {                          if ( C1 ) {
           S1;                                 S1;
           if ( C2 ) {                         if ( C2 ) {
               S2;                                 S2;
               if ( C3 ) {                         if ( C3 ) {
                   S3;                                 S3;
1          break C;                     1                  goto C;
               }                                   }
           }                                   }
       }                                    }
1      S4; // only once                  1    S4; // only once
   }                                     1  } C: ;
```

   (c) **1 mark** Nested control structures can be added without changing existing code.

   (d) **2 marks** A VLA is an array that is dynamically sized at creation and it appears on the stack.

   (e) **2 marks** uArrayPtr performs a single dynamically allocation $O(1)$ in the heap for an array of N elements. unique_ptr creates an array of N pointers in the heap *and* performs $O(N)$ dynamic allocations for each element.

   (f) **2 marks** (one of) Specifically, stack smashing occurs when a non-local **goto** (longjmp) transfers to a stack frame that has been deallocated.
   Generally, stack smashing occurs when the stack is corrupted by incorrectly writing data over stack frames, e.g., buffer overrun.

   (g) **7 marks**

```
1   _Exception E;
1   void f( ..., /* no fixups */ ) {
1       if ( ... ) _Resume E();
        // control returns here
    }
    int main() {
1       try {
1           f( ... ); // no fixups
1       } _CatchResume( E ) {
            // handler 1
        }
-       try {
-           f( ... ); // no fixups
-       } _CatchResume( E ) {
1           // handler 2
        }
    }
```

2. (a) **1 mark** An *infinite recursion* of **catch** and **throw** can occur among the handlers at the same level.

   (b) **2 marks** **_CatchResume** handler is *dynamic* return. **catch** handler is *static* return.

   (c) **3 marks** Propagation searches the stack for a _CatchReume handler for R, and does not find one. The defaultResume is called for R, which throws R. Propagation now unwinds the stack from the raise until the matching **catch** clause for R.

   (d) **7 marks**

   i. 2 unguarded, 4 guarded
   ii. 2 throws
   iii. 7 examined catch clauses
   iv. C8
   v. B6 resumption, B2 retry, B1 termination

   (e) **1 mark** heap

   (f) **3 marks** uThisCoroutine, **this**, last resumer

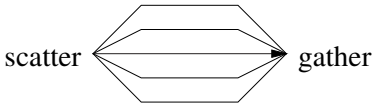
   (g) **2 marks** *cycle* is call graph, *definition-before-use* issues building the cycle

3. (a) **3 marks**
   Normalize: $T_1 = 10/10 = 1$, $T_4 = 2.5/10 = .25$.
   Amdahl's Law:

$$S_4 = \frac{1}{(1-.8)+.8\times.25} = \frac{1}{.2+.2}$$

   2.5 times

   (b) **3 marks**



   (c) **5 marks**

```
      void tree( unsigned int N ) {
1         if ( N != 0 ) {
1             COBEGIN
1                 BEGIN tree( N / 2 ); END
1                 BEGIN tree( N / 2 ); END
              COEND
1         } else p();
      }
```

   (d) **3 marks**

   i. low-priority task can starve
   ii. high-priority code can go right back into critical section after exit protocol
   iii. it alternates priority so tasks take turns backing down if the other wants in

   (e) **1 mark** synchronization

4. **20 marks**

```
1   void error() {
1       _Resume Error() _At resumer();
1       _Throw Error();                          // raise local and terminate
    } // Phone::error

1   void getNdigits( int N ) {
1       for ( int i = 0; i < N; i += 1 ) {       // must have 3 digits
1           if ( ! isdigit( ch ) ) { error(); }  // must have digit
1           suspend();                           // get next character
        } // for
    } // Phone::getNdigits

    void main() {
1       try {
1           if ( ch == '(' ) {                   // area code ?
1               suspend();                       // get digit
1               getNdigits( 3 );
1               if ( ch != ')' ) { error(); }    // must have ')'
1               suspend();                       // get digit
            } // if

1           getNdigits( 3 );
1           if ( ch != '-' ) { error(); }        // must have '-'

1           suspend();                           // get digit
1           getNdigits( 4 );

1           if ( ch != EOT ) { error(); }        // must be done
1           _Resume Match() _At resumer();
1       } catch ( Error & ) {}
    } // Phone::main
```

-5 duplicate code -5 if not using coroutine state.

5. (a) **5 marks**

```
1   for ( int c = 0, cnt = 0; c < cols; c += 1 ) {
1       if ( row[c] == schmilblick ) {
1           cnt += 1;
1           if ( cnt == 2 ) return true;
        } // if
    } // for
1   return false;
```

(b) **3 marks**

```
1   COFOR( r, 0, rows,
2       if ( ! schmilblickCheck( M[r], cols, schmilblick ) ) found = false;
    );
```

(c) **7 marks**

```
    struct WorkMsg : public uActor::Message {           // derived message
1       const int * row, cols, schmilblick;
        bool & found;
        WorkMsg( const int row[], int cols, int schmilblick, bool & found ) :
1           Message( uActor::Delete ), row( row ), cols( cols ), schmilblick( schmilblick ), found( found ) {}
    };
    _Actor Schmilblick {
1       Allocation receive( Message & msg ) {
1           iftype( WorkMsg, msg ) {                    // discriminate derived message
2               if ( ! schmilblickCheckmsg.row, msg.cols, msg.schmilblick ) ) msg.found = false;
            } endiftype
1           return Finished;                            // one-shot
        }
    };
```

(d) **5 marks**

```
1   uActor::start();                                    // start actor system
1   Schmilblick schmilblicks[rows];
1   for ( unsigned int r = 0; r < rows; r += 1 ) {
1       schmilblicks[r] | *new WorkMsg( M[r], cols, schmilblick, found );
    }
1   uActor::stop();                                     // wait for all actors to terminate
```

(e) **7 marks**

```
_Task Schmilblick {
1   const int r, * row, cols, schmilblick;
-   uBaseTask & pgmMain;

    void main() {
1       try {
1           _Enable {
1               if ( ! schmilblickCheck( row, cols, schmilblick ) )
1                   _Resume NotSchmilblick() _At pgmMain;
            }
1       } catch( Stop & ) {}
    }
  public:
    Schmilblick( int r, const int row[], int cols, int schmilblick, uBaseTask & pgmMain ) :
1       r( r ), row( row ), cols( cols ), schmilblick( schmilblick ), pgmMain( pgmMain ) {}
};
```

(f) **13 marks**

```
1   Schmilblick * workers[rows];
1   for ( r = 0; r < rows; r += 1 ) {                    // create tasks to process rows
1       workers[r] = new Schmilblick( r, M[r], cols, schmilblick, uThisTask() );
    } // for
1   try {
1       r = 0;                                           // initialize before Enable
1       _Enable {
1           for ( ; r < rows; r += 1 ) {                 // wait for completion and delete tasks
1               delete workers[r];
            } // for
        } // _Enable
1   } _CatchResume( Schmilblick::NotSchmilblick & ) {
1       if ( found ) {
1           found = false;
1           for ( int i = r + 1; i < rows; i += 1 ) { // immediately stop any more checking
1               _Resume Schmilblick::Stop() _At *workers[i];
            } // for
        } // if
    } // try
```