# *Examination #1*

Name:_____ SUID:_____

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All examinations will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be easiest.

1. Describe all the syntax and semantics for passing arguments to a function by reference. Please consider all the relevant cases.

   Answer:

**Passing method arguments by Reference:**

**Syntax:**
**- declare function argument with ref qualifer**
**- invoke function with ref qualifier**
**Semantics:**
**- caller's value type sees change of value made to reference in method**
**- caller's reference type sees both changes of value and changes of instance to reference made in method**

See MT1Q1 code

2. How do you write code for a thread that accepts arguments and returns a value? You may use any of the threading constructs we discussed in class for your answer. Please provide an example where the thread will accept a string and return a boolean result.

   Answer: Two alternate answers shown

```
class MT1Q2
{
  bool result { get; set; }

  bool hasSpecificSubstring(string s)
  {
    return s.Contains("CSE681");
  }

  void alsoHasSpecificSubstring(object s)
    // need this signature for parameterized thread start
  {
    result = ((string)s).Contains("CSE681");
  }

  static void Main(string[] args)
  {
    MT1Q2 q2 = new MT1Q2();
    string s = "hello CSE681 world";
    Task<bool> t = Task<bool>.Run(() => { return q2.hasSpecificSubstring(s); });

    bool result = t.Result;
    if(result)
    {
      Console.Write("\n  {0} contains \"CSE681\"", s);
    }
    else
    {
      Console.Write("\n  {0} does not contain \"CSE681\"", s);
    }

    Thread trd = new Thread(q2.alsoHasSpecificSubstring);
    trd.Start(s);
    trd.Join();

    if (q2.result)  // safe to access result because thread has stopped
    {
      Console.Write("\n  {0} contains \"CSE681\"", s);
    }
    else
    {
      Console.Write("\n  {0} does not contain \"CSE681\"", s);
    }
    Console.Write("\n\n");
  }
}
```

3. What are the goals for writing an Operational Concept Document (OCD) and what are its principal parts?

   Answer:

   The goals for writing an OCD are:
   - Clearly describe the concept for a project's development, e.g.:
     - Who are the users and how will they use the code developed to implement the concept? Note that users may be other software entities as well as people.
     - What are the main packages for this development and what are their responsibilities and interactions.
     - What issues may effect the usebility of the code and affect how easily it can be implemented.
     - Ensure that the concept is coherent, understandable, useful, and identify the most important parts of the development, all before beginning to write code.
   - Provide an invariant description of what the team is trying to accomplish, independent of reasonable changes in requirements, and changes in the implementing team.

   The main parts are:
   - Executive Summary
   - Description of the systems uses and users.
   - Description of its most important parts, e.g., package and their responsibilities, activities, and context, using text descriptions augmented with UML diagrams, and possibly some small code prototypes.
   - Discussion of Critical Issues, e.g., things that affect system usability, performance, ease of implementation, vulnerability to threats, and maintainability.

Some exams claimed that the document is for users. It is not. It provides a vehicle for the Architect, Program Manager, and team leads to develop their ideas about what the project should be and how it will function, e.g., the services it provides users and how those are deployed in major packages. It also helps these individuals assess risk and convey to their stakeholders. Users will probably never see the document.

Other exams claimed that it helps to manage requirements. It does not. An OCD is a Concept document, not a requirements document.

**4.** Write a WCF Service Contract that is used to receive messages that have the following parts:
To address, from address, a command enumeration, and a list of strings.
Show how you could dispatch a received message to a lambda for processing, based on its command enumeration.

Answer:

```
[ServiceContract(Namespace = "MessagePassingComm")]
public interface IMessagePassingComm
{
  [OperationContract(IsOneWay = true)]
  void postMessage(CommMessage msg);

  CommMessage getMessage();
}

[DataContract]
public class CommMessage
{
  public enum MessageType
  {
    [EnumMember] connect,
    [EnumMember] request,
    [EnumMember] reply,
  }

  public CommMessage(MessageType mt)
  {
    type = mt;
  }

  [DataMember]
  public MessageType type { get; set; } = MessageType.connect;

  [DataMember]
  public string to { get; set; }

  [DataMember]
  public string from { get; set; }

  [DataMember]
  public Command command { get; set; }

  [DataMember]
  public List<Argument> arguments { get; set; } = new List<Argument>();
}

Dictionary<string, Func<CommMessage, CommMessage>> messageDispatcher =
  new Dictionary<string, Func<CommMessage, CommMessage>>();

CommMessage msg = server.comm.getMessage();
CommMessage reply = server.messageDispatcher[msg.command](msg);
server.comm.postMessage(reply);
```

5. Write code to create a delegate type that can bind to a function that accepts a string and returns a bool. Bind that to a lambda that checks to see if the string contains the substring "CSE681". How do you invoke the delegate.

Answer:

```
class MT1Q5
{
  delegate bool DelType(string s);
  DelType delInst;

  bool hasSpecificSubstring(string s)
  {
    return s.Contains("CSE681");
  }

  static void Main(string[] args)
  {
    MT1Q5 q5 = new MT1Q5();
    string s = "hello CSE681 world";
    q5.delInst = (string arg) => { return q5.hasSpecificSubstring(arg); };

    bool result = q5.delInst.Invoke(s);

    if (result)
    {
      Console.Write("\n  {0} contains \"CSE681\"", s);
    }
    else
    {
      Console.Write("\n  {0} does not contain \"CSE681\"", s);
    }

    Console.Write("\n\n");
  }
}
```

6. Write code to show how you could use the Lexer (Toker and Semi) to find all the using statements for namespaces. Please create instances of the Toker and Semi and show how you would use them to process a file "Test.CS", in the current directory, to detect those using statements.

Answer:

```
class MT1Q6
{
  string path = "../../MT1Q6.cs";  // imagine MT1Q6.cs is Test.cs
  static void Main(string[] args)
  {
    Console.Write("\n  MT1Q6 - find using statements with Lexer");
    Console.Write("\n ========================================");
    MT1Q6 mt1q6 = new MT1Q6();
    mt1q6.path = System.IO.Path.GetFullPath(mt1q6.path);
    Console.Write("\n  fileSpec: {0}", mt1q6.path);

    ////////////////////////////////////////////////
    // Beginning of answer
    Toker toker = new Toker();
    toker.open(mt1q6.path);
    Semi semi = new Semi();
    semi.toker = toker;
    while(!semi.isDone())
    {
      semi.get();
      if (semi[0] == "using")
        semi.show();
    }
    // End of answer
    ////////////////////////////////////////////////

    Console.Write("\n\n");
  }
}
```

7. Draw an activity diagram for Tokenizer you built for Project #2.

   Answer:

   Note: Deadlock if you remove Ready process.