Individual Exercises - Lecture 2

1. Do Sebesta Review Questions 49 on page 128 and 50 on page 129
   49. How does the typing system of PHP and JavaScript differ from that of Java?
   JavaScipt and PHP use dynamic type binding, while Java uses static type binding. Dynamic type binding is more flexible, but disallows type checking at compile time.

   50. What array structure is included in C# but not in C, C++, or Java?
   C# includes both jagged arrays (also in C, C++, C# and Java) and rectangular arrays. Rectangular arrays have a rectangular shape and entries are accessed using multiple indexes, i.e. [x, y], as opposed to [x][y] for jagged arrays.

2. Do Sebesta Programming Exercises 1,2 and 3 on page 131
   1. To understand the value of records in a programming language, write a small program in a C-based language that uses an array of structs that store student information, including name, age, GPA as a float, and grade level as a string (e.g., “freshmen,” etc.). Also, write the same program in the same language without using structs.
   Without structs, you would use an extra dimension in the array for the properties. See an example of this in the code snippet (C#) below.

```
class Program
{
   static void Main(string[] args)
   {
       /*
        * Struct version
        */
       // Easy initialization
       Student s1 = new Student("Walter", 50, 4.0f, "Senior");
       Student s2 = new Student("Jesse", 25, 1.0f, "Freshmen");

       Student[] students = new Student[2];
       students[0] = s1;
       students[1] = s2;

       for (var i = 0; i < students.Length; i++)
       {
           // Easy and readable field access
           Console.WriteLine(students[i].Name);
       }
```

```
        /*
         * Array representation
         */
        object[,] arrStudents = new object[2, 4];
        arrStudents[0, 0] = "Walter";
        arrStudents[0, 1] = 50;
        arrStudents[0, 2] = 4.0f;
        arrStudents[0, 3] = "Senior";
        arrStudents[1, 0] = "Jesse";
        arrStudents[1, 1] = 25;
        arrStudents[1, 2] = 1.0f;
        arrStudents[1, 3] = "Freshmen";

        for (var i = 0; i < arrStudents.GetLength(0); i++)
        {
            // Difficult to read field access
            Console.WriteLine(arrStudents[i,0]);
        }

    }
}

public struct Student
{
    public Student(string name, int age, float gpa, string gradeLevel)
    {
        Name = name;
        Age = age;
        GPA = gpa;
        GradeLevel = gradeLevel;
    }

    public string Name;
    public int Age;
    public float GPA;
    public string GradeLevel;

}
```

2. To understand the value of recursion in a programming language, write a program that implements quicksort, first using recursion and then without recursion.
As you can see below, the recursive version of QuickSort is significantly more readable. You can see the implementations in several different languages her: https://www.geeksforgeeks.org/quick-sort/

```java
// Java program for implementation of recursive QuickSort
import java.util.*;

class QuickSort {
    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
    smaller (smaller than pivot) to left of
    pivot and all greater elements to right
    of pivot */
    static int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = (low - 1); // index of smaller element
        for (int j = low; j <= high - 1; j++) {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot) {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }
```

```
    /* The main function that implements QuickSort()
    arr[] --> Array to be sorted,
    low --> Starting index,
    high --> Ending index */
    static void qSort(int arr[], int low, int high) {
        if (low < high) {
            /* pi is partitioning index, arr[pi] is
            now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            qSort(arr, low, pi - 1);
            qSort(arr, pi + 1, high);
        }
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 5;
        int arr[] = { 4, 2, 6, 9, 2 };

        qSort(arr, 0, n - 1);

        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Iterative version can be found on the next page.

```java
// Java program for implementation of iterative QuickSort
import java.util.*;

class QuickSort {
    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
    smaller (smaller than pivot) to left of
    pivot and all greater elements to right
    of pivot */
    static int partition(int arr[], int low, int high) {
        int pivot = arr[high];

        // index of smaller element
        int i = (low - 1);
        for (int j = low; j <= high - 1; j++) {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot) {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    /* A[] --> Array to be sorted,
   l  --> Starting index,
   h  --> Ending index */
    static void quickSortIterative(int arr[], int l, int h) {
        // Create an auxiliary stack
        int[] stack = new int[h - l + 1];

        // initialize top of stack
        int top = -1;
```

```
        // push initial values of l and h to stack
        stack[++top] = l;
        stack[++top] = h;

        // Keep popping from stack while is not empty
        while (top >= 0) {
            // Pop h and l
            h = stack[top--];
            l = stack[top--];

            // Set pivot element at its correct position
            // in sorted array
            int p = partition(arr, l, h);

            // If there are elements on left side of pivot,
            // then push left side to stack
            if (p - 1 > l) {
                stack[++top] = l;
                stack[++top] = p - 1;
            }
            // If there are elements on right side of pivot,
            // then push right side to stack
            if (p + 1 < h) {
                stack[++top] = p + 1;
                stack[++top] = h;
            }
        }
    }
    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 4, 3, 5, 2, 1, 3, 2, 3 };
        int n = 8;

        // Function calling
        quickSortIterative(arr, 0, n - 1);

        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

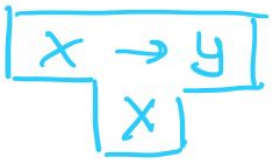
3. To understand the value of counting loops, write a program that implements matrix multiplication using counting loop constructs. Then write the same program using only logical loops—for example, while loops.
While splits definition, check, and increment into three lines, a for loop uses one. Counting loops increases readability of the code.

```
// Matrix multiplication not using counting loops
public static double[][] multiplyMatricesWhileLoops(double[][] M1,
double[][] M2){
   double[][] result = new double[M1.length][M2[0].length];

   int row = 0;
   while(row < result.length){
       int col = 0;
       while(col < result[row].length){
           double cell = 0;
           int i = 0;
           while(i < M2.length){
               cell += M1[row][i] * M2[i][col];
               i++;
           }
           result[row][col] = cell;
           col++;
       }
       row++;
   }

   return result;
}
```

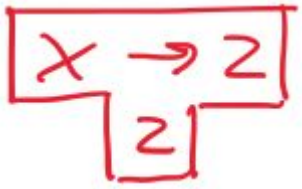
```
// Matrix multiplication using counting loops
public static double[][] multiplyMatricesCountingLoop(double[][] M1,
double[][] M2) {
   double[][] result = new double[M1.length][M2[0].length];

   for (int row = 0; row < result.length; row++) {
       for (int col = 0; col < result[row].length; col++) {
           double cell = 0;
           for (int i = 0; i < M2.length; i++) {
               cell += M1[row][i] * M2[i][col];
           }
           result[row][col] = cell;
       }
   }
   return result;
}
```

3. (Optional) Exercise 4 from Lecture 1: Do Fisher et al. exercise 3 on page 26 (exercise 5 page 55 in GE) poses some interesting questions about bootstrapping and viewing compilers as pieces of software in general. For this purpose the notion of T-diagrams, also sometimes called tombstone diagrams, has been invented. Read the above references and use T-diagrams to solve the exercise. Draw your T-diagrams using TDiagram Editor for the AtoCC toolkit.

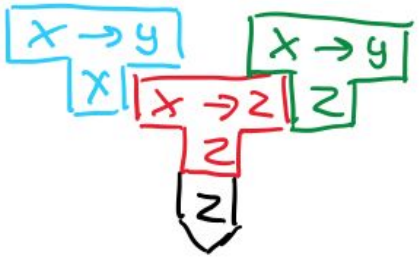
   Since I want a compiler from X to Y written in X running on system Y i start by creating this compiler.



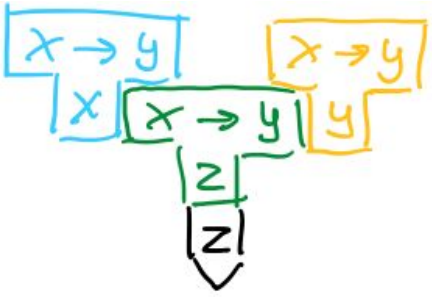
   I assume that I already have a compiler from X to Z that runs on system Z.



   I can use these two pieces to get a compiler from X to Y written in X, running on system Z. This is my cross compiler.

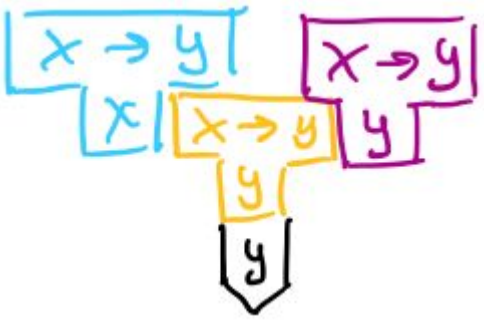


   In combination with the new cross compiler and system Z i can now create a compiler from X to Y written in Y.



   I then transfer the compiler from X to Y written in Y to system Y.

I now have a compiler for language X written in X, that runs on system Y and generates code for system Y.



4. Do Fisher et al. exercise 4 on page 27

   4. Cross-compilation assumes that a compiler for language X exists on some machine. When the first compiler for a new language is created, this assumption does not hold. In this situation, a **bootstrapping** approach can be taken. First, a subset of language X is chosen that is sufficient to implement a simple compiler. Next, a simple compiler for the X subset is written in any available language. This compiler must be correct, but it should not be any more elaborate than is necessary, since it will soon be discarded. Next, the subset compiler for X is rewritten in the X subset and then compiled using the subset compiler previously created. Finally, the X subset, and its compiler, can be enhanced until a complete compiler for X, written in X, is available.
   Assume you are bootstrapping C++ or Java (or some comparable language). Outline a suitable subset language. What language features must be in the language? What other features are desirable?

   Required: I/O (File / Memory), Branches (control structure), Iteration, Dynamic Memory

   Deseriable: Subroutines, Strings, Boolean

5. Familiarize yourself with the companion web site to Crafting a Compiler http://www.cs.wustl.edu/~cytron/cacweb/e.g. read the FAQ and follow the instructions Installing Eclipse.
   For this a virtual machine will be provided with the necessary tools pre installed.

Group Exercises - Lecture 2

1. Do Sebesta exercise 6, 7, 14, 16, 17, 18, 21, 22, 24 on page 129-130

   6. Make an educated guess as to the most common syntax error in Lisp programs.
   Too many, too few, or incorrectly positioned parentheses.

   7. Lisp began as a pure functional language but gradually acquired more and more imperative features. Why?
   To increase performance, but some problems are also hard to express using a pure functional programming language.

   14. What are the arguments both for and against the idea of a typeless language?
   For: More freedom to create abstractions and less typing. If everything is well thought out errors should not occur.
   Against: Increased amounts of bugs that are difficult to find, since in reality most programs are not perfectly written.

   16. What is your opinion of the argument that languages that are too complex are too dangerous to use, and we should therefore keep all languages small and simple?
   Complexity makes it easier to introduce subtle bugs without knowing. Users might only use parts of the language, while other users use different parts of the language. This might lead to each sub-group not understanding each other's language subset. Last, as languages die the programs written in it might not, this could introduce legacy code that is very hard for developers to debug and maintain.

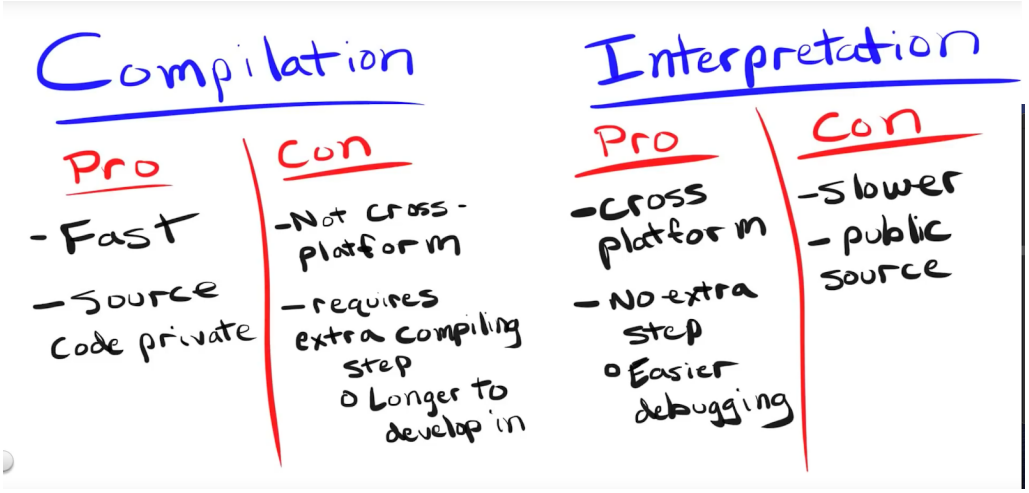
   17. Do you think language design by committee is a good idea? Support your opinion.
   A committee can help create a standard accepted by multiple parties (e.g. C++ committee has multiple companies), but also makes the process slower. However, if the members of the committee do not have shared goals, the resulting language might contain different features with the same functionality and be very complex.
   Open-question: is open-source language design the same as design by committee (Python PEP, Rust RFC, etc)?

   18. Languages continually evolve. What sort of restrictions do you think are appropriate for changes in programming languages? Compare your answers with the evolution of Fortran.
   Backwards compatibility might be very important. Or at least a very good compiler, compiling the old code to working new code. It may, however, be difficult to do while keeping the semantics of the program.
   Fortran is well known for having a high level of backwards compatibility.

21. In recent years data structures have evolved in scripting languages to replace traditional arrays. Explain the chronological sequence of these developments.

It all started as direct memory management with just a block of memory. Then we got static size arrays and linked lists. By now most programming languages contain some kind of arrays.

22. Explain two reasons why pure interpretation is an acceptable implementation method for several recent scripting languages.
One situation in which pure interpretation is acceptable for scripting languages is when the amount of computation is small, for which the processing time will be negligible. Another situation is when the amount of computation is relatively small and it is done in an interactive environment, where the processor is often idle because of the slow speed of human interactions.



24. Give a brief general description of a markup-programming hybrid language.
A markup-programming hybrid language is a markup language in which some of the elements can specify programming actions, such as control flow and computation. The concept of literate programming is close, here a markdown language embeds blocks of code written in one or more programming languages (R, Emacs Org-mode, Jupyter Notebook).