Individual Exercises - Lecture 14

1. Read the articles under additional references
   They can be found here:
   http://zorac.aub.aau.dk/login?url=http://link.springer.com/book/10.1007%2F978-0-85729-829-4 (chapter 4, p97-106)
   Preprint:
   http://people.cs.aau.dk/~bt/PLATEAU2016/Preprint-PLATEAU2016-KurtevChristensenThomsen.pdf
   Sample sheet: http://people.cs.aau.dk/~bt/PLATEAU2016/samplesheet.pdf
   Task sheet: http://people.cs.aau.dk/~bt/PLATEAU2016/tasksheet.pdf

2. Go through the review questions (Sebesta) 1-37, p. 436-437

   1. What are the three general characteristics of subprograms?

   There is a caller and a callee

   Callee returns control to the caller upon completion

   The subprogram has **1** entry point.

   While the callee is called, the caller is suspended, awaiting return of control from the callee.

   2. What is a subprogram call?

   Request for execution of a specific subprogram (Sebesta 391)

   3. What is a subprogram definition?

   A subprogram definition consists of a header (describes the interface of the subprogram) and a body (describing the actions of the subprogram) (Sebesta 391)

   4. What characteristic of Python subprograms sets them apart from those of other languages?

   Function definitions (keyword 'def') are executable statements that assign the function name to the function body. (Sebesta 391)

   5. How do Ruby methods differ from the subprograms of other programming languages?

   In Ruby methods can be defined outside class definitions, in which case they are considered methods of the root 'Object'(Sebesta 392)

   6. What is the feature of Lua functions?

   All Lua functions are anonymous (they don't have names) (Sebesta 392)

   7. What are function declarations called in C and C++? Where are the declarations often placed?

   Prototypes. They are often placed near the top of the program or in header files.

8. Name one pure functional programming language that does not have mutable data.

Haskell (Sebesta 393)

9. What are positional parameters?

For example a function in Java, that has the following parameters (int i, float f, String s), could not take the following arguments (“string”, 5.2, 1), even though each of the arguments’ types is represented. They have to follow the order of the parameters.

10. Give an example of a language that allows positional parameters in addition to keyword parameters.

Python, C#

11. What is the use of a default value in a formal parameter?

Providing a default value for a parameter makes it optional during invocation. If a value is not provided for the parameter it will set to its default value. Some languages, like C#, requires all non-optional parameters to precede optional ones. This is to ensure each argument is matched to the correct parameter. An alternative to this is using named parameters.

See:https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/named-and-optional-arguments

12. What is the rule of using a default parameter in C++?

Formal parameters with default values must appear after all other parameters because C++ use positional parameters

13. What is the rule for accepting variable parameters in C# methods?

Use the params keyword. All parameters must be of the same type

14. What language allows array formal parameters?

Ruby

15. What is an ellipsis?

Java ellipsis syntax (...) is used to accept a variable number of parameters

16. What are the modes, the conceptual models of transfer, the advantages, and the disadvantages of pass-by-value, pass-by-result, pass-by-value- result, and pass-by-reference parameter-passing methods?

Pass-by-value: In-mode. Often implemented by copy.

One advantage is that the callee can freely modify the passed value, and the caller can know that the passed variable will not be affected (this is complicated slightly by boxing). A disadvantage is that it can be expensive if the passed value is large.

Pass-by-result: Out-mode. Typically by copy.

Similar advantages/disadvantages of pass-by-value. Has the problem of parameter collision. See Sebesta 9.5.2.2.

Pass-by-value-result: Inout. By copy.
Has the same problem of memory/cpu cost associated with copying. Also has the problem of parameter collision.

Pass-by-reference: Inout. By access path.
Passing the parameter is effective in terms of memory and computation time. Access to the formal parameters is, however, slightly slower due to extra level of indirection. A disadvantage is that the subprogram may unexpectedly alter the variable you pass to it, or even store the reference and alter the value later. This can make debugging more challenging.

17. Describe the ways that aliases can occur with pass-by-reference parameters.
If the parameter of a function has a different name, than the argument being passed to the function. Both variables will point to the same object / place in memory, but they are referred to by two different names.
18. What is the difference between the way original C and C89 deal with an actual parameter whose type is not identical to that of the corresponding formal parameter?
In original C neither the number of parameters and their types were not checked. In C89 the type is included for the parameters in the function prototype.
19. Name some languages that support procedures.
C++, C, Python
20. Describe the problem of passing multidimensional arrays as parameters.
The problem relates to storing arrays of arrays. For instance in C, a function can only occupy a fixed size of memory and so a function must be written for each length of array that needs support, ie to pass an array with 100 columns you need the following function definition in C:

```
void fun(int matrix[][100])
```

..and to pass an array with 101 columns you need a function with definition:

```
void fun(int matrix[][101])
```

This problem can be avoided if the array is passed as a pointer, but then you will also need to pass the lengths of the array (rows and columns) and calculate the index according to row major order (See Sebesta 9.5.6).

21. What is the name of the parameter-passing method used in Ruby?

Ruby uses pass-by-assignment. (see page 410 in Sebesta)

22. What are the two issues that arise when subprogram names are parameters?

Type checking the parameters of the passed subprogram and which environment should be used to execute the passed subprogram

23. Define shallow and deep binding for referencing environments of subprograms that have been passed as parameters.

Shallow: The environment of the call statement that enacts the passed subprogram. That is, the passed subprogram runs within the environment of the subprogram that calls it.

Deep: The environment of the definition of the passed subprogram. That is, the passed subprogram runs within the environment of where it was defined.

24. What is a generic subprogram?

Functions that work regardless of the type of the argument given. This saves time, since some algorithms can now be implemented only once.

25. What is ad hoc binding?

Ad-hoc binding refers to a method for determining which environment to reference when executing nested subprograms. In ad-hoc binding we use the environment of the call statement that passed the subprogram as an actual parameter. For a more thorough explanation see Sebesta Section 9.6 pages 418-419.

26. What causes a C++ template function to be instantiated?

Naming the function in a call, or taking its address with the '&' operator

27. In what fundamental ways do the generic methods of Java 5.0 differ from those of C# 2005?

C# does not support wildcard types (indicated by '?' in Java) (Sebesta 428)

28. If a Java 5.0 method returns a generic type, what type of object is actually returned?

During execution the raw method works on objects of the Object class. Upon returning the value it is cast to the appropriate type.

29. If a Java 5.0 generic method is called with three different generic parameters, how many versions of the method will be generated by the compiler?

Only one copy of the code is built (because, as stated above, the raw method operates on objects of the Object class).

30. When does a variable have unlimited extent?

A variable has unlimited extent, if the variable's lifetime is the same as the lifetime of the entire program.

31. What is subtype polymorphism?

This is a concept from object oriented programming, where a variable of type A can be assigned a value of type A or any of the subclasses derived from A.

32. What is a multicast delegate?

A delegate class that can store a list of delegates, which can then be called in order and the value of the last delegate is returned.

For definition of a delegate see Sebesta 9.7 on page 420.

33. What is the main drawback of generic functions in F#?

F# performs type inference for generic functions. So some generic functions like adding to parameters, will have their types inferred to int and the function becomes less useful.

34. What is a coroutine?

- A normal function will run to completion before returning.
- A coroutine is a function that has the ability to pause execution, and return control to the caller for a time, before picking it up again where it left off.

35. What are the language characteristics that make closures useful?

Statically scoped, allow nested subprograms and allows subprograms as parameters

36. What languages allow the user to overload operators?

Python, C#, C++, Haskell… (see https://en.wikipedia.org/wiki/Operator_overloading for a longer list)

37. What is a symmetric unit control model?

The coroutine control mechanism (Sebesta 9.13, p. 432)

3. Do Sebesta chapter 9, p 438, exercise 4

4. Suppose you want to write a method that prints a heading on a new output page, along with a page number that is 1 in the first activation and that increases by 1 with each subsequent activation. Can this be done without parameters and without reference to nonlocal variables in Java? Can it be done in C#?

The Java program below implements a counter that does not make use of method arguments nor nonlocal references:

```java
public static void main(String[] args) {
    Runnable incrementer = createIncrementer();
    incrementer.run(); // Prints 0
    incrementer.run(); // Prints 1
    incrementer.run(); // Prints 2
}

public static Runnable createIncrementer(){
    AtomicInteger i = new AtomicInteger();
    return () -> System.out.println(i.getAndIncrement());
}
```

And here is a C# version

```
class Program {
    static void Main(string[] args) {
        var pageCounter = GetPageCounter();
        Console.WriteLine($"Page: {pageCounter()}");
        Console.WriteLine($"Page: {pageCounter()}");
        Console.WriteLine($"Page: {pageCounter()}");
    }

    static Func<int> GetPageCounter() {
        int page = 0;
        return () => page++;
    }
}
```

4. Argue for or against Java's design not to include operator overloading.

Allowing operator overloading makes operations with mathematical entities like complex number and matrices simpler (e.g. `BigDecimal` in Java uses an add method instead of +). On the other hand by allowing users to overload operators, types can be created where + and - have completely different meaning making programs much harder to understand. However, this negative aspect is similar to the problem of classes and method being poorly named. Consider for example the addition of two lists list1 + list2, does this refer to a pairwise addition of the elements, or concatenation of list1 and list2?

5. Go through Sebesta chapter 14, p 652-653, review questions 1-27

1. Define exception, exception handler, raising an exception, continuation, finalization, and built-in exception.

Exception: The state of the program has somehow deteriorated to a point where the intended computation is no longer possible and instead of continuing an exception is raised as the mechanism to handle this state, via the exception handler.
Exception handler: A component responsible for deciding how to proceed in the event of a specific type of exception.
Raising an exception: An event occurs, that triggers an exception. For example, some invariant no longer holds.
Continuation: The question of whether to terminate the program or resume exection.
Finalization: Code that is always executed, regardless of whether an exception is thorwn/raised. For example the finally block in java.
Build-in exception: The exception types built into the language (as opposed to programmer-defined exceptions)

2. How did the early programming languages support the exception-handling mechanism?
In early programming languages the occurrence of errors simply caused the program to be terminated and control to be transferred to the operating system. Alternatively, some languages, e.g. C, use integer values to represent at exit status of a program. This integer value may represent some exception/error.
3. What are the advantages of having support for exception handling built into a language?
It allows for programmers to create code that to some extent attempts to handle exceptions that could occur as a construction that isn't directly written along with the rest of the program's statements.

It makes it possible to handle exceptions externally by raising errors up though the call stack to an eventual point that has a solution.
4. What is the reason that some languages do not include exception handling?
One reason some languages do not include exception handling is the complexity it adds to the language. Some methods of exception handling could be considered a special case of the goto, a jump to another point in the program.
5. Give an example of an error that is not detectable by hardware but can be detected by the code generated by the compiler in Java.
Checking that arrays are indexed within their bounds.
6. What is the use of label parameters in Fortran?
This allows the callee to return to a different point in the caller if an exception is thrown.
7. How does exception propagation work in Java?
An exception is propagated through the call-stack until an appropriate `try/catch` catches the exception. If the exception is not caught before the main method, the program terminates.
8. How does a throw statement without an operand work in C++?
An empty `throw` either rethrows the exception if used inside a handler or terminates the program. For more information see (2):
https://en.cppreference.com/w/cpp/language/throw
9. How can an exception handler be written in C++ so that it handles any exception?
C++ has a special construct for catching all exceptions:

    catch(...)

10. What constraint will a function overriding a function that has a throws clause face?
The overriding function cannot have a more general exception in its own throws clause.
11. Does Java include built-in exceptions?
Yes. Here is a list
(https://www.geeksforgeeks.org/types-of-exception-in-java-with-examples/)
12. Which library functions in C++ support the overflow_error exception?
Some third party math libraries like boost.math

13. What are the two system-defined direct descendants of the Throwable class in Java?
Error and Exception.
14. What package in Java contains the Array Index Out Of Bounds Exception?
Java.lang.Exception
15. How can an exception handler be written in Java so that it handles any Exception?
catch(Exception e) will catch all exceptions.
catch(Exception e), however, catches exceptions and serious runtime errors that most programs should not catch (e.g., VirtualMachineError - the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.).
16. What is the difference between C++ exception handlers and Java exception handlers?
C++ can throw anything as an exception, Java can only throw 'throwables'
C++ has a catch all exception
17. What is the difference between Java exception handlers and Python exception handlers?
Python's handlers allow for an `else` clause to be defined which is called if no exception is raised.
18. How can a method deal with exceptions it receives from another method that lists a particular checked exception in its throw clause?
It can call the other method inside a try statement, with a catch clause, that handles the case, where the exception is thrown. In Java, methods with a checked exception can only be called inside a try-clause with a catch-clause catching the checked exception.
19. In Java, when does a try clause only have a finally clause and no exception handlers?
This was a result of the explicit lock interface introduced in Java 5.0. Now a section that acquired a lock will be implemented in a try clause, with the lock being released in a finally clause.
20. In Java, what clause should a method include to propagate any checked exception?
A `throws` clause must be added to the methods definition.
21. Explain what an else block in Python does.
This is a block of code that only executes in the absence of an exception thrown in the try clause.
22. What advantages do assert statements have?
They can help in debugging by telling exactly where in the program a function started having unexpected values. These assertions can also be disabled for the deployment build. This saves time for later program maintenance.
23. What happens when an assert statement evaluates to false?
In Java the AssertionError exception is thrown.
24. What does the message method of Ruby's StandardError class do?
Returns the result of invoking `exception.to_s`. Normally this returns the exception's human-readable error message or its name, as `to_s` by default returns the exception's message or the name of the exception if no message is set.

25. What exactly happens when a raise statement with a string parameter is Executed?
In Ruby this causes an `RuntimeError` to be raised with the string as its message.
26. What does the Python _debug_ flag do?
A constant that can be disabled by running the python program with the -o flag. Means that unnecessary calls, such as messages written to the console during design time can be removed elegantly.
27. In what ways are exception handling and event handling related?
In both cases a pre-registered handler is called when something occurs (an exception or an event). However, while exceptions are either explicitly raised by user code or implicitly raised by the runtime environment, events are created by external actions such as the user clicking a button in a gui.

For more information see Sebesta Section 14.5 - 14.7

6. (optional but recommended) In exercise 2 for Lecture 11 you created an AST for the little language defined in Figure 7.14. Design and implement a recursive interpreter for the language based on the Visitor Pattern (clue: look at the Visitor for pretty printing you implemented in exercise 4 in Lecture 11)

```
 1  Start  → Stmt $
 2  Stmt   → id assign E
 3         | if lparen E rparen Stmt else Stmt fi
 4         | if lparen E rparen Stmt fi
 5         | while lparen E rparen do Stmt od
 6         | begin Stmts end
 7  Stmts  → Stmts semi Stmt
 8         | Stmt
 9  E      → E plus T
10         | T
11  T      → id
12         | num
```

Figure 7.14: Grammar for a simple language.

```java
// Visitor for IfExpression in the main interpreter class
public void visit(ifExprNode ifNode) {
      boolean predicateResult;
      predicateResult = ifNode.predicateNode.accept(new BooleanVisitor());
      if (predicateResult) {
             // Execute statements inside the if statement body
            ifNode.Body.accept(this);
      } else if (ifNode.hasElseClause()) {
            // Execute statements inside the else body
            ifNode.elseBody.accept(this);
      }
}
```

This language does not feature boolean expressions/operators, so we are assuming the c-style approach to boolean values, where 0 represents *false* and all other numbers represent *true*. The ExpressionVisitor recursively calls down depth first into the expression until literal values or ids are found. If an id is found, the value of said variable is looked up in the symbol table, and the expression is then evaluated using the values from the variables and the literals values of the expression when returning up the tree.

```
public boolean visit(PlusExpression plusNode) {
      int leftValue = plusNode.leftChild.accept(new ExpressionVisitor());
      int rightValue = plusNode.rightChild.accept(new ExpressionVisitor());
      int sum = leftValue + rightValue;
      return sum != 0;
}
```

Group Exercises - Lecture 14

1. Discuss the outcome of the individual exercises
   Did you all agree on the answers?

2. Do Sebesta chapter 9, p 438, exercise 5
   5. Consider the following program written in C syntax:

```
void swap(int a, int b){
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void main(){
    int value = 2, list[5] = {1, 3, 5, 7, 9};
    swap(value, list[0]);
    swap(list[0], list[1]);
    swap(value, list[value]);
}
```

For each of the following parameter-passing methods, what are all of the values of the variables value and list after each of the three calls to swap?

A) Passed by value,
swap(value, list[0]) => value: 2 and list: 1, 3, 5, 7, 9
swap(list[0], list[1]) => value: 2 and list: 1, 3, 5, 7, 9
swap(value, list[value]) => value: 2 and list: 1, 3, 5, 7, 9

B) Passed by reference
swap(value, list[0]) => Value: 1 and List: 2 3 5 7 9
swap(list[0], list[1]) => Value: 1 and List: 3 2 5 7 9
swap(value, list[value]) => Value: 2 and List: 3 1 5 7 9

C) Passed by value-result
As stated in Sebesta Section 9.5.2.2 when using variables that are passed-by-result for indexing arrays the effect depends on when the parameter is bound to an address. If the address is bound at the time of the call the result will match B, otherwise the result is:

swap(value, list[0]) => Value: 1 and List: 2 3 5 7 9
swap(list[0], list[1]) => Value: 1 and List: 3 2 5 7 9
swap(value, list[value]) => Value: 2 and List: 3 2 1 7 9

For more information about parameter passing methods see Sebesta Section 9.5.2

3. What mechanism can be used for detecting errors in programming languages without exception handling?

Examples include error codes (e.g., C, C++, Go), returning types that can be either the result or an empty value like Option in Rust (e.g., Rust, OCaml, Haskell), or jumping to different labels depending on if the operation fails or succeeds (e.g. Fortran see Sebesta Section 14.1).

4. What is the difference between exception handling in C# and Java?

Java supports both checked and unchecked exceptions, C# only has unchecked exceptions.

A checked exception is an exception the caller is forced to handle if the callee throws it.

5. (optional) Write a recursive interpreter for (a subset) of the language you are designing in your project

Only do a very small subset of your language.