Individual Exercises - Lecture 12

1. Do the following for Java, C, PhP and SML: (You may replace the languages with other languages you are familiar with)

   a. *Describe the set of values that data objects of type Integer (int) may contain*

      C: Ints are either 2 or 4 bytes big, in order to see the size of an int on your local compiler load the <limits.h> package and print INT_MAX

      2^16 or 2^32 signed.

      -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647

   b. *Determine the storage representation for values of Integer type*

      C: Stores integers as the binary representation in memory.

      Other languages like Java have the concept of “boxing” of primitive data types like ints and objects. See: https://stackoverflow.com/questions/27647407/why-do-we-use-autoboxing-and-unboxing-in-java

   c. *Determine the syntactic representation used for constants of that type*

      C uses the “const” keyword as part of the type qualifier. Const prohibits modifying an int beyond its initialization.

      Int const a = 1;

   d. *Determine the set of operations defined for Integers and give the signature for these operations and syntactic representation*

      Arithmetic Operators: +, -, *, /, %, ++, --

      Relational Operators: ==, !=, <=, >=, <, >

      Logical Operators: &&, ||, !

      Bitwise Operators: &, ^, |, ~, <<, >>

      Assignment Operators: =, +=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=

      Misc Operators: sizeof, &, *, ? :

      C automatically coerces types in some operations such as addition. Adding an int to a float will first convert the int to a float. Same is true for different integer sizes.

e. *Determine if any of the operation symbols defined for integers may be overloaded*

C does not support operator overloading

Consider looking at C++ operator or C# operator overloading:
https://en.cppreference.com/w/cpp/language/operators
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/operator-overloading

f. *Determine whether static or dynamic type checking is used to determine the validity of each operation.*

C utilises static type checking based on the types of the operands.

2. Do exercise 1 for floating point data type

Like for exercise 1 this information is usually available in the documentation of the language/implementation, its standard, or specified by the implementation of the language.

For instance for C#:
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/types

Most languages map integers directly to their binary representation. Floating point presents new challenges in this format. Different standards exist (as mentioned in today's chapter) for storing floats in binary. These representations are often a compromise between simplicity of the implementation, size, speed of translation/operation, and precision of the value.

3. Consider the languages C and Java. For each language determine what the language implementation does with respect to constructs that cannot be checked statically, e.g. arrays bounds check, access to pointers/references and division by 0.

a) *Does the language provide dynamic checking or leave it unchecked at runtime?*

Java provides runtime bounds checking. For array access it throws an `ArrayIndexOutOfBoundsException` when trying to access indexes outside the specified array size. Additionally, when accessing variables that have not yet been assigned a value a `NullPointerException` is thrown (not the case for primitive types as they have default values).

In C, however, there is no runtime checking for array indexes and non-initialized values.This means that it is possible to lookup data located outside of the array in

memory. Additionally, no checks are made when accessing uninitialized data structures, and they may contain garbage data. Welcome to the land of undefined behaviour.

Java provides runtime checks for division by 0 and throws an instance of ArithmeticException when it occurs. In C division by zero is undefined behaviour but does not trigger any exceptions.

a. *Are there other constructs you can think of?*
Legal values for types. Fx. both C and Java exhibit rollover behaviour when encountering an integer overflow. Neither language provides runtime checking for this. Some versions of the Ada language trigger an exception in case of overflow.

a. *What about the language you are designing?*

What data types are in your language? Do you perform any runtime or compile checks on them?

4. The language SIMSCRIPT allows data elements of different types to be stored as elements of an array. One reason for this is that arrays in SIMSCRIPT are implemented as vectors of pointers to the data elements thus each element in an array has a uniform representation. Both Java and SML use the same implementation technique, but do not allow data elements of mixed types in arrays. Can you explain why? Can you give a work-around in Java that enables you to store mixed elements in an array? Can you explain why this works in Java? What about your own language – will it allow mixed types in arrays/compound data types – why/why not?

Java can provide stronger type checking at compile time by only allowing one type of element in a collection.

To work around this, collections, e.g. `ArrayList`, can be declared with the type `Object`, which enables all types deriving from `Object(every non-primitive type)` to be added to the list regardless of their type. Additionally, the collections can be declared using interfaces or super classes(like Object), which both also enable different subtypes in the same array.

5. Do Sebesta exercise 9 on page 322.

9. Multidimensional arrays can be stored in row major order, as in C++, or in column major order, as in Fortran. Develop the access functions for both of these arrangements for three-dimensional arrays.
See the following resources:

- https://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays

- https://stackoverflow.com/questions/14015556/how-to-map-the-indexes-of-a-matrix-to-a-1-dimensional-array-c

Group Exercises - Lecture 12

1. Discuss the outcome of the individual exercises

   Did you all agree on the results?

2. Do the individual exercise 1 and 2 for your own language

   If you do not deal with the low level details of how integers and floats are represented in your language, then consider how they are represented in the language that you are compiling to.

3. Do Sebesta exercise 13 on page 323 (rather than write, discuss with your group) or more to the point discuss what was lost and what was gained by excluding pointers from the design of Java

   What was gained:
   - Simplification of the language. No need to worry about dereferencing and pointer arithmetic (which can often lead to program errors for unexperienced programmers)

   What was lost:
   - Low level control of memory. I.e no/limited control of stack/heap allocation (possible through `sun.misc.Unsafe`) and no option to pass parameters by reference(parameters are always passed by value in Java).

   See Sebesta - 6.11.7 Implementation of Pointer and Reference Types for more Information.

4. What are the arguments for and against heap management with single-sized elements and variable-sized elements?

   Single-sized cells:

   For:
   - Simple
   - Allocation is easy (just follow the linked list of pointers to available cells and take the amount you need)

   Against:
   - Not as powerful, most languages need variable-sized cells.
   - Garbage collection is difficult
     - Potential for multiple pointers to same cell (Can be solved using reference counting or the mark sweep approach)

   Variable-size cells:

   For:
   - Heap space can be utilized more effectively with objects of variable size

   Against:

- Has all the sames difficulties of managing a heap of single-size cells
- Scanning is problematic, where does the next cell start, if they have different sizes? (This can be solved by having each cell store its own size, but this uses more space)
- Finding an available (big enough) space is a complex problem

For more information see Sebesta Section 6.11.7.3 Heap Management

5. Do Sebesta exercise 16 on page 323.

16. What would you expect to be the level of use of pointers in C#? How often will they be used when it is not absolutely necessary?

Pointers are rarely used in C# as constructs are added to replace them, specifically: references, out parameters, and the Marshal Class. As such, pointers are generally only used when a managed solution adds an unacceptable overhead, and their use is discouraged by the language designers by requiring the `unsafe` keyword .

The principal argument for the more restrictive constructs used in C# is better error detection, with which better reliability is obtained. A perhaps nebulus argument for the use of pointers would be the loss of freedom when, e.g. using references. Languages such as C and C++ put a stronger ‘faith’ in the programmer, that they know what they are doing and will not make mistakes.