# Advanced Programming (I00032)
# A DSL for Crane Control with Type Classes

Assignment 13

In the previous assignment we made a DSL for crane control using a simple GADT version. In this assignment we make a similar DSL using type classes.

## 1 DSL

The DSL is very similar to previous assignment. The difference is that `WhileContainerBelow` is replaced by `While` with an expression as argument. The action give as second argument is repeated here. For the `Expr` definition we use semiformal function type notation inspired inspired on the GADT types.

```
:: Action
  = MoveToShip                  // move the crane to the ship
  | MoveToQuay                  // move the crane to the quay
  | MoveUp                      // moves the crane from down to up position
  | MoveDown                    // moves the crane from up to down position
  | Lock                        // locks the top container on the stack under the crane
  | Unlock                      // unlocks the container the crane is carrying, put it on the stack
  | Wait                        // do nothing
  | (:.) infixl 1 Action Action // sequence of two actions
  | While (Expr Bool) Action    // repeat action while expression yields true

:: Expr x
  = ContainersBelow :: (Expr Int) // number of containers at current position
  | Lit             :: t → Expr t | toString t
  | (<.) infix 4    :: (Expr t) (Expr t) → Expr Bool | <, toString t
  | (>.) infix 4    :: (Expr t) (Expr t) → Expr Bool | <, toString t
  | (+.) infix 4    :: (Expr Int) (Expr Int) → Expr Int
```

Define actions and expressions in a DSL based on type classes as introduced in the lecture. Design the DSL in such a way that dangerous combinations of actions (locking and unlocking when the crane is high and move when the crane is low) are rejected by the type system.

Hint: like in the previous assignment we use types

```
:: High = High
:: Low  = Low
```

There are several ways to achieve this. One solutions is using actions that produce a phantom type like `:: Step init target = Step init target`, e.g., the result of `MoveUp` will have type `Step Low High`. Another solution is to pass the position actively around and change it whenever that is needed.

The program to load all containers from the quay to the ship can look like:

```
loadShip =
    While (containersBelow  >. lit 0) (
        moveDown:.
        lock:.
        moveUp:.
        moveToShip:.
        wait:.
        moveDown:.
        wait:.
        unlock:.
        moveUp:.
        moveToQuay
    )
```

Again, we assume that the crane is initially in the high position above the quay and that all containers from the quay fit on the ship.

## 2 Show View

Define a view of the DSL that gives a representation of the program in the DSL as `[String]`. Try to ensure that the output for the program above matches the definition above textually as close as possible.

## 3 Evaluation View

Define an evaluator for programs in this DSL. It is recommended to reuse the type `State` and the initial state from the previous assignment as much as possible.

## 4 *Bonus, you do not have to make this:* Variables

To illustrate that the DSL can be extended we will add variables to the Crane-DSL. For simplicity we use a slightly simpler representation as shown in the lecture. We represent variables by their sequence number boxed in the type `Var`.

```
:: Var = Var Id
:: Id  :== Int
```

Crane-DSL has three options for variable handling:

1. Variable introduction by the keyword `int`, an expression in the Crane-DSL, and a function getting the variable in the current view as argument. The body of the function is an action as defined above. It is fine to allow only integer variables.

2. Variable occurrence in an expression is flagged by the keyword `var`.

3. Assignment is indicated by the infix-operator =.. The lefthand-side is a variable and the righthand-side is an integer expression.

The program below illustrates the definition and use of variables. Define the class `var` and instances for showing and evaluating this class. Most likely both states have to be extended with some administration for the variables defined.

The program to load the ship can now be written as:

```
loadShip2 =
    int containersBelow \n.
    While (var n >. lit 0) (
        moveDown:.
        lock:.
        moveUp:.
        moveToShip:.
        wait:.
        moveDown:.
        wait:.
        unlock:.
        moveUp:.
        moveToQuay:.
        n =. var n +. lit -1
    )
```

## 5 *Bonus, you do not have to make this:* Optimization

Write an optimizer for actions that removes all `wait` actions from a program.

A drawback of the class based DSL design is that optimizations and other views that need to inspect the structure of the DSL can be rather hard to write.

## Deadline

The deadline for this assignment is June 6, 23:59h.