# CPS506 – Final Assignment

**Preamble**

Described below is the final assignment for CPS506. It is a substitute to a written exam and can be completed at home over the course of the exam period. Submission details are found at the end of this description. Your task involves implementing operations over a custom type in Haskell and Rust. Traits in Rust are directly inspired by type classes in Haskell, so it is fitting to compare the two.

**Assignment description**

You will write a program that implements a custom data type called a CauchyList. A CauchyList is a standard list of integers that has some special arithmetic operations associated with it. Specifically, if **a**, **b** are two CauchyLists you are going to overload the operators **+**, **-**, *****, etc. such that expressions like a + b, a – b, etc. can be evaluated.

The data type representing a CauchyList should contain two fields:

1. An integer which we will call **p**. All the arithmetic operations are performed mod **p**.
2. a list of integers called **content**. The elements of **content** are always kept in the range [0, p – 1].

For example, two CauchyLists **a**, **b** of length 10 and 6, respectively, with p = 31 are:

a = [17, 9, 22, 27, 28, 27, 15, 28, 24, 1]

b = [12, 4, 7, 15, 13, 4]

You must implement the following operations for the CauchyList type:

- **The + operator:**

  The sum of two CauchyLists is defined component-wise. That means if a, b are two CauchyLists then we define c = a + b by setting c.content[i] = a.content[i] + b.content[i] where the addition is done mod **p**.
  This method should be able to handle CauchyLists of different lengths. When adding two lists of length m and n, where m > n, the resulting list should be of length m and the higher elements of the shorter list is assumed to be zero. For example, for the above a, b we have a + b = [29, 13, 29, 11, 10, 0, 15, 28, 24, 1].

- **The – operator:**

  Like the **+** operator. For example, for the above a, b we have a - b = [5, 5, 15, 12, 15, 23, 15, 28, 24, 1].

- **The * operator:**

  Suppose that a, b are two CauchyLists. Then the product c = a*b is defined as:

  c[i] = a[0] * b[i] + a[1] * b[i - 1] + a[2] * b[i - 2] + · · · + a[i] * b[0]
  where we denoted a.content[k] by a[k] for simplicity.

  Remember, c[i] must be reduced mod **p** at the end. The product of two CauchyLists of lengths m and n is a CauchyList of length m + n - 1. So the index i in c[i] can be larger than the lengths of a and b. You should take care of this, again by assuming that the elements outside the bounds of a list are zero. In the example above, we have a * b = [18, 21, 16, 17, 24, 24, 2, 18, 18, 0, 27, 16, 5, 16, 4].

  If the second argument is an integer, not a CauchyList, then this method does a scalar multiplication which is much simpler. That is, if b is an integer then c[i] = a[i] * b. This allows us to be able to, for example, evaluate an expression of the form a * 65.

- **The == operator:**

  Two CauchyLists are equal if they have the same modulus value **p** and the same **content** component-wise.

- **String representation:**

  a CauchyList should have a string value of the form

  p: **p**
  length: **length**
  content: **content**
  where **length** is the length of **content**. In the example above, a is printed as
  p: 31
  length: 10
  content: [17, 9, 22, 27, 28, 27, 15, 28, 24, 1]

**Operator overloading in Haskell**

We have already seen examples of operator overloading in Haskell. Recall the **Pt2** custom data type in the lectures. To overload the operators +, -, *, we made Pt an instance of the type class **Num.** The syntax to do so is as follows:

```
Instance Num Pt2 where
…
```

To overload the operator ==, we made Pt2 an instance of the type class **Eq** using the following:

```
Instance Eq Pt2 where
```

…

Note: instances of type class Num must implement **abs**, **signum**, and **fromInteger**. For abs and signum you can perform component-wise abs and signum just like in the Pt2 type. For fromInteger, however, you need implement it in a way as to satisfy the requirement on the * operator to correctly evaluate expressions such as a * 65 where a is a CauchyList.

**Operator overloading in Rust**

Operators in Rust can be overloaded using the packages std::ops, std::cmp, etc. Recall the **Pt2D** in the lectures. We implemented the **Display** trait from std::fmt for Pt2D. Overloading the + operator can be done using the **Add** trait from std::ops. In the following, we overload the + operator for Pt2D.

```
struct Pt2D {x: f64, y: f64}

impl fmt::Display for Pt2D {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "<{}, {}>", self.x, self.y)
    }
}

impl ops::Add for Pt2D {
    type Output = Pt2D;

    fn add(self, other: Pt2D) -> Pt2D {
        return Pt2D {x: self.x + other.x, y: self.y + other.y};
    }
}
```

We can specify a type for Add by replacing ops::Add with, for example, ops::Add<Pt2D>. If we wanted to be able to add a Pt2D to an integer we could implement the trait ops::Add<i32> as well. The line “type Output = Pt2D” specifies the result of the + operation.
To overload the == operator, you need to implement the **PartialEq** trait from std::cmp.

**Implementation Details & Templates**

To get you started, templates are provided for both Haskell and Rust along with this assignment. You should start with these templates and add your code to them. In these templates, the custom types are already defined for you. In the case of Rust, you’re even given the trait and method signatures. You must fill them in.

We will be testing your code by evaluating expressions on your CauchyList implementation. Therefore, it is very important that your implementation is internally consistent. All your operations should be able to be linked together. I.e., your code should be able to evaluate expressions like the following: (a * b * b + b * c - a * 32) where a, b, c are all CauchyLists.

**Submission & Evaluation**

You will submit three files for this assignment:

1. A Haskell source file named Cauchy.hs that contains a module called Cauchy (i.e. **`module Cauchy where`** …). This module must contain all the implementation required to satisfy the requirements above, and it must load cleanly into ghci. To test your code, you should load your Cauchy module into ghci, create some CauchyLists, and perform the operations. In the Haskell slides there are many examples of doing this for the Pt2D type.
2. For Rust, you will also submit a single source file called Cauchy.rs. Within Cauchy.rs should be all the implementation required to satisfy the description above. You can test your Rust implementation in similar fashion, by writing a main() function, declaring some CauchyLists, and performing some operations on those lists. Please do **NOT** include a main() function in your final submission.
3. The third file you should submit is a text document with a few paragraphs (half a page or so) explaining which implementations is more efficient and why. Think about which is more efficient in terms of space, time, and amount of syntax. Which one was easier for you as a programmer to implement? Explain and justify your arguments.

Both source files (Cauchy.hs and Cauchy.rs) and your written answer to part 3 above should be submitted on D2L under "Final Assignment" in the same manner you've been submitting your Poker assignments. The deadline, indicated on D2L, is Saturday, April 25$^{th}$ at 11:30PM. Submissions past this date will accumulate late marks in the same manner as the assignment.