## Kolokwium 1

**Programowanie w języku Rust** Wersja A

Wszystkie zadania wykonujemy w jednym pliku. Powinna się w nim znaleźć też funkcja `main()`, która przetestuje działanie kodu z zadań.

Proszę o podpisanie tej kartki jako listy obecności.

**Zadanie 1 (0-5)** Napisz funkcję `fn email(&str) -> bool`, która zwróci *true* wtedy, gdy parametr jest poprawnym adresem email. Przyjmujemy, że poprawny adres:

- składa się wyłącznie z liter, cyfr, kropek i dokładnie jednej małpy,
- pierwszy i ostatni znak to litery lub cyfry,
- w części za małpą znajduje się kropka.

**Zadanie 2 (0-5)** Zaprojektuj strukturę `Set`, reprezentującą skończony podzbiór liczb naturalnych. Zaimplementuj dla niej metody:

- `fn new() -> Set`, tworzącą zbiór pusty,
- `fn from_slice(&[u32]) -> Set`, tworzącą zbiór z elementów przekazanych w argumencie,
- `fn union(&Set, &Set) -> Set`, zwracającą zbiór, będący sumą mnogościową obu argumentów.

Mówimy, że zbiory A i B są w relacji $A < B$, jeśli A jest podzbiorem właściwym B. Własnoręcznie zaimplementuj cechę `PartialOrd` dla typu `Set`.

**Zadanie 3 (0-5)** Stwórz strukturę `Transaction`, reprezentującą przelew bankowy. Typ powinien zawierać informacje o kwocie przelewu oraz numerach kont nadawcy i odbiorcy, przechowywanych przy pomocy typu napisowego.

Zaprojektuj typ `BankAccount`, przechowujący numer konta oraz historię transakcji. Zaimplementuj dla niego metody:

- konstruktor,
- `fn transaction(t: Transaction)`, dopisującą transakcję do historii konta; zadbaj o to, by metoda rozpoznała czy jest to przelew przychodzący, czy wychodzący. Jeśli dane konto nie jest odbiorcą ani nadawcą, nic nie rób.
- `fn balance() -> f32`, obliczającą obecny stan konta na podstawie całej historii transakcji.
- `fn last() -> Option<Transaction>`, zwracającą ostatnią transakcję wykonaną na danym koncie.