პროგრამირების აბსტრაქციები

სემინარის ამოცანები #7

1 Fold ფუნქციის გამოყენება

ა) დაწერეთ ფუნქცია, რომელიც მომხმარებლისგან აიღებს საწყის მთელ რიცხვს (integer-ს), მთელი რიცხვების სიმრავლეს და ორი მთელი რიცხვის გამაერთიანებელ" ფუნქციას. შემდეგ მიყოლებით გააერთიანებს ყველა რიცხვს საწყის რიცხვთან (სანამ ბოლოს ერთი რიცხვი არ დარჩება) და მიღებულ რიცხვს დააბრუნებს. (ეს ოპერაცია ხშირად გამოიყენება ფუნქციონალური პროგრამირების პარადიგმაში. დეტალები შეგიძლიათ ვიკიპედიაზე ნახოთ)

```
#include "vector.h"

int fold(int defaultValue, Vector<int> values,
         int (combine)(int, int)) {
    int currentValue = defaultValue;
    foreach (int val in values) {
        currentValue = combine(currentValue, val);
    }
    return currentValue;
}
```

ბ) ამ ფუნქციის გამოყენებით (და მასში შესაბამისი “გამაერთიანებელი ფუნქციების” გადაცემით) დაწერეთ პროგრამა, რომელიც მომხმარებელს მოსთხოვს მთელი რიცხვების შეყვანას (სანამ მომხმარებელი ე.წ. სენტინელ მნიშვნელობა (-1)-ს არ შეიყვანს) და გამოიტანს ამ რიცხვების ჯამს, ნამრავლს და “გამომრიცხავ ან”-ს (ანუ xor-ს).

```
#include <iostream>
#inclue "console.h"
#include "simpio.h"

int add(int a, int b) {
    return a + b;
}

int mult(int a, int b) {
    return a * b;
}

int xorTwo(int a, int b) {
    return a ^ b;
}

const int sentinel = -1;

int main() {
    Vector<int> intVector;

    while (true) {
        cout << "Enter next integer (or " << sentinel << " to stop): ";
        int x = getInteger();
        if (x == sentinel)
            break;
        intVector.add(x);
    }

    cout << "Sum = " << fold(0, intVector, add) << endl;
    cout << "Product = " << fold(1, intVector, mult) << endl;
    cout << "Xor = " << fold(0, intVector, xorTwo) << endl;
    return 0;
}
```

2 გზის ძებნა BFS-ით

მოცემულია ცხრილი (Grid-ის სახით), რომელშიც მითითებულია თითოეული უჯრა დაბლოკილია თუ არა გასავლელად. ასევე გადმოგვეცემა ორი წერტილის კოორდინატები. ჩვენი მიზანია დავწეროთ ფუნქცია, რომელიც დაადგენს ამ ორ წერტილს შორის თუ არსებობს ისეთი გზა რომელიც დაბლოკილ უჯრებზე არ გაივლის. დიაგონალური სვლების გაკეთება არ შეგვიძლია. ეს ამოცანა ჯობია BFS-ით გააკეთოთ ვიდრე DFS-ით.

ა) კოორდინატების გადასაცემად დაგვჭირდება ორი კოორდინატის შემცველი სტრუქტურა. ასეთი სტრუქტურის ობიექტები სიმრავლეში უნდა ჩავყაროთ ხოლმე, რისთვისაც მოგვიწევს მათთვის შემდარებელი ფუნქციის დაწერა. ჯერ ეს სტრუქტურა და მისი შემდარებელი ფუნქცია დავწეროთ.

```
struct point {
    int x;
    int y;

    point(int xVal = 0, int yVal = 0) {
        x = xVal;
        y = yVal;
    }
};

int cmp(point p1, point p2) {
    if (p1.x < p2.x)
        return -1;
    else if (p1.x > p2.x)
        return 1;
    else if (p1.y < p2.y)
        return -1;
    else if (p1.y > p2.y)
        return 1;
    else
        return 0;
}
```

ბ) ახლა დავწეროთ ორ წერტის შორის არსებობს თუ არა გზა ამის დამდგენი ფუნქცია.

```
Vector<point> neighbours(point pt) {
    Vector<point> result;
    result.add(point(pt.x - 1, pt.y));
    result.add(point(pt.x + 1, pt.y));
    result.add(point(pt.x, pt.y - 1));
    result.add(point(pt.x, pt.y + 1));
    return result;
}

bool isValid(point pt,
             Grid<bool>& blocked,
             Set<point>& visited) {
    if (pt.x < 0 || pt.y < 0
        || pt.x >= blocked.numCols()
        || pt.y >= blocked.numRows())
        || blocked[pt.y][pt.x]
        || visited.contains(pt)
        return false;
    return true;
}

bool canBeReached(Grid<bool> blocked, point start, point end) {
    Set<point> visited(cmp);
    Queue<point> pointQueue;

    pointQueue.enqueue(start);
    visited.insert(start);

    while (!pointQueue.isEmpty()) {
        point currentPt = pointQueue.dequeue();
        if (cmp(currentPt, end) == 0) {
            return true;
        }

        foreach(point pt in neighbours(currentPt)) {
            if (isValid(pt, blocked, visited)) {
                pointQueue.enqueue(pt);
                visited.insert(pt);
            }
        }
    }

    return false;
}
```