

Міністерство освіти і науки, молоді та спорту України
Національний університет “Львівська політехніка”

Кафедра ЕОМ



Звіт

з домашнє завдання №27.1 з дисципліни:

“Алгоритми та моделі обчислень”

Варіант: № 24.

Виконав:

ст. групи КІ-203

Ширий Богдан Ігорович

Перевірів:

ст. викладач кафедри ЕОМ

Козак Назар Богданович

ЗАВДАННЯ:

УМОВА:

Застосовуючи елементи реактивної парадигми програмування (код має містити в явній формі Observable та Observer) за допомогою RxJS (реалізація ReactiveX для JavaScript) скласти програму (мовою JavaScript для платформи NodeJS), яка дозволяє виконувати валідацію ключа (25 шістнадцяткових цифр) ліцензії для деякого програмного продукту. Для зручності вводу програма має групувати шістнадцяткові цифри по п'ять цифр та відображати 'A', 'B', 'C', 'D', 'E', та 'F' завжди у верхньому регістрі. Окрім шістнадцяткових цифр та символів пробілу і табуляції (що трактується як одна цифра 0), програма валідації не дозволяє вводити жодних інших символів, які не належать шістнадцятковій системі числення. При цьому надається не більше ATTEMPTS_COUNT спроб вводу ключа ліцензії. Між ітераціями сусідніх спроб введені значення мають зберігатися для редагування у наступній спробі.

Ключ ліцензії «11111 22222 33333 44444 55555» має міститися в коді програми валідації у неявній формі.

ВИБІР ВАРІАНТУ:

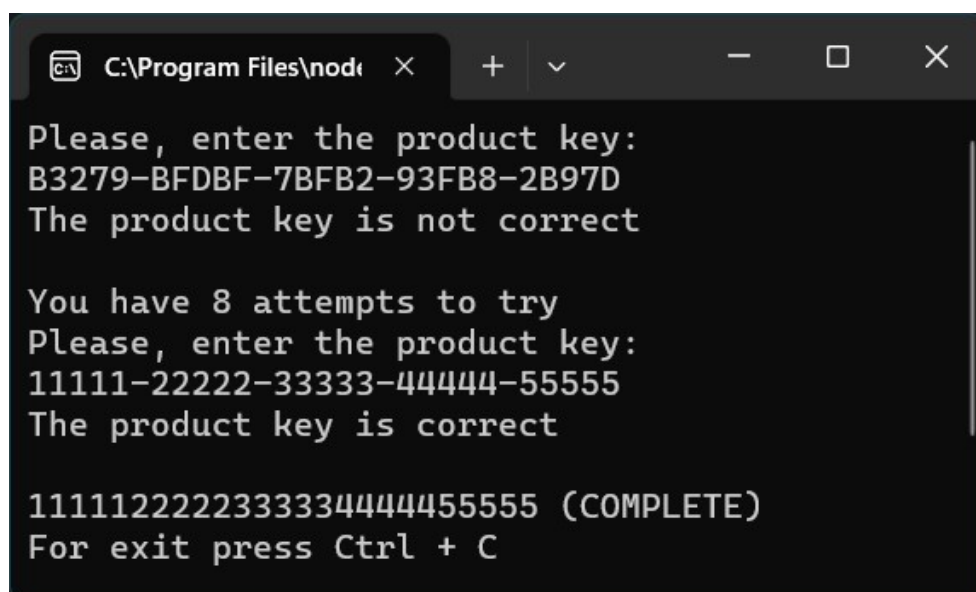
$$(N_{\text{ж}} + N_{\text{г}} + 1) \% 30 + 1 = (24 + 3 + 1) \% 10 + 1 = 28 \% 10 + 1 = 9,$$

де: $N_{\text{ж}}$ – порядковий номер студента в групі, а $N_{\text{г}}$ – номер групи.

Отже, мій шуканий варіант – це 9 можливих спроб введення ключа валідації.

ВИКОНАННЯ:

Склав Node.js консольну програму та зобразив її роботу на рисунку 1.



```
C:\Program Files\node... x + v - □ x

Please, enter the product key:
B3279-BFDBF-7BFB2-93FB8-2B97D
The product key is not correct

You have 8 attempts to try
Please, enter the product key:
11111-22222-33333-44444-55555
The product key is correct

1111122222333334444455555 (COMPLETE)
For exit press Ctrl + C
```

Рис. 1. Виконання програми написаної на Node.js.

Відповідно, у лістингу 1 навів код програми написаної на Node.js:

Лістинг 1. Код програми написаної у Node.js платформі.

```
const { from_ } = require('rxjs');
const { fromEvent } = require('rxjs');
const { map, filter } = require('rxjs/operators');

const ATTEMPTS_COUNT = 9
var attemptsDownCount = ATTEMPTS_COUNT;

const GROUPS_DIGITS_COUNT = 5;
const GROUP_DIGITS_SIZE = 5;

const PRODUCT_KEY_PART1 = [
    0xF, 0xF, 0xF, 0xF, 0xF,
    0xD, 0xD, 0xD, 0xD, 0xD,
    0x8, 0x8, 0x8, 0x8, 0x8,
    0xB, 0xB, 0xB, 0xB, 0xB,
    0xF, 0xF, 0xF, 0xF, 0xF
];

const PRODUCT_KEY_PART2 = [
    0xE, 0xE, 0xE, 0xE, 0xE,
    0xF, 0xF, 0xF, 0xF, 0xF,
    0xB, 0xB, 0xB, 0xB, 0xB,
    0xF, 0xF, 0xF, 0xF, 0xF,
    0xA, 0xA, 0xA, 0xA, 0xA
];

const DIGITS_COUNT = GROUPS_DIGITS_COUNT * GROUP_DIGITS_SIZE;

var outOfEdgeIndex = 0;
var currIndex = 0;
var data = new Array(DIGITS_COUNT).fill(0);

function integerDiv(a, b) {
    return (a - a % b) / b;
}

function checkProductKey(productKey) {
    for (var index = 0; index < DIGITS_COUNT; ++index) {
        if (productKey[index] ^ PRODUCT_KEY_PART1[index] ^ PRODUCT_KEY_PART2[index])
        {
            return false;
        }
    }

    return true
}

function toDigitPosition(currIndex) {
    let positionAddon = integerDiv(currIndex, GROUP_DIGITS_SIZE);
    positionAddon && positionAddon >= GROUPS_DIGITS_COUNT ? --positionAddon : 0;
    process.stdout.cursorTo(currIndex + positionAddon);
}

function printProductKey(productKey, outOfEdgeIndex) {
    for (var index = 0; index < DIGITS_COUNT && index < outOfEdgeIndex; ++index) {
        process.stdout.write(productKey[index].toString(16));
    }
}

function printFormattedProductKey(productKey, outOfEdgeIndex) {
    for (var index = 0; index < DIGITS_COUNT && index < outOfEdgeIndex; ++index) {
        process.stdout.write(productKey[index].toString(16));
        if (!((index + 1) % GROUP_DIGITS_SIZE) && (index + 1) < DIGITS_COUNT) {
```

```

        process.stdout.write('-');
    }
}

function inputHandler(ch, key) {
    if (!attemptsDownCount) {
        return;
    }
    if (key && key.name == 'return') {
        if (checkProductKey(data)) {
            process.stdout.write("\nThe product key is correct\n\n");
            printProductKey(data, outOfEdgeIndex);
            process.stdout.write(' (COMPLETE)');
            process.stdout.write('\nFor exit press Ctrl + C\n');
            attemptsDownCount = 0;
        }
        else {
            process.stdout.write("\nThe product key is not correct\n");
            process.stdout.write("\nYou have " +
                --attemptsDownCount + " attempts to try");
            if (attemptsDownCount) {
                process.stdout.write("\nPlease, enter the product key:\n");
                printFormattedProductKey(data, outOfEdgeIndex);
                toDigitPosition(currIndex);
            }
            else {
                process.stdout.write("\nThe product key is not entered\n");
                process.stdout.write("For exit press Ctrl + C\n");
            }
        }
    }

    if (key && key.name == 'backspace') {
        if (currIndex) {
            --currIndex;
            toDigitPosition(currIndex);
            data[currIndex] = 0;
            process.stdout.write('0');
            toDigitPosition(currIndex);
        }
    }
    else if (key && key.name == 'delete') {
        toDigitPosition(currIndex);
        data[currIndex] = 0;
        process.stdout.write('0');
        toDigitPosition(currIndex);
    }
    else if (key && key.name == 'left') {
        if (currIndex) {
            toDigitPosition(--currIndex);
        }
    }
    else if (key && key.name == 'right') {
        if (currIndex < outOfEdgeIndex) {
            toDigitPosition(++currIndex);
        }
    }
}

var hexDigitRegularExpression = /^[0-9A-Fa-f]\b/;

if (ch && hexDigitRegularExpression.test(ch) && currIndex < DIGITS_COUNT) {
    data[currIndex] = ch.toUpperCase();
    process.stdout.write(data[currIndex]);
    if (outOfEdgeIndex <= currIndex) {
        outOfEdgeIndex = currIndex + 1;
    }
    if (currIndex + 1 < DIGITS_COUNT) {

```

```

        ++currIndex;
        if (currIndex !== DIGITS_COUNT && !(currIndex % 5)) {
            process.stdout.write('-');
        }
    }
    if (currIndex + 1 === DIGITS_COUNT) {
        toDigitPosition(currIndex);
    }
}

console.clear();
var keypress = require('keypress');
keypress(process.stdin);

process.stdin.setRawMode(true);
process.stdin.setEncoding('utf8');

process.stdin.resume();

process.stdin.on('keypress', (ch, key) => {
    if (key && key.ctrl && key.name === 'c') {
        process.exit();
    }
});

const inputObservable = fromEvent(process.stdin, 'keypress');

const PrintObserver = {
    next: (key) => {
        inputHandler(key[0], key[1]);
    },
    error: (err) => {
        console.error('something wrong occurred: ' + err);
    },
    complete: () => {
        console.log('done');
    }
};

inputObservable
    .pipe(
        map(key =>
            (key && key[0] && (key[0] === ' ' || key[0] === '\t')) ?
                ['0', key[1]] : key
        )
    )
    .subscribe(PrintObserver);

console.clear();
if (attemptsDownCount) {
    process.stdout.write('Please, enter the product key:\n');
}
process.stdout.close;

```