







```
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <time.h>
9
10 #define NANOSECONDS_PER_SECOND_NUMBER 1000000000
11
12 #define DATA_TYPE volatile unsigned long long int
13
14 #define N1 18
15 #define N2 27
16
17 #define MIN(a,b) (((a)<(b))?(a):(b))
18 #define MAX(a,b) (((a)>(b))?(a):(b))
19
20 #define REPEAT_COUNT 1000000
21 #define REPEATOR(count, code) \
22 for (unsigned int indexIteration = (count); indexIteration--;){ code; }
23
24 double getCurrentTime(){
25     clock_t time = clock();
26     if (time != (clock_t)-1) {
27         return ((double)time / (double)CLOCKS_PER_SEC);
28     }
29     return 0.; // else
30 }
```

```
26     if (time != (clock_t)-1) {
27         return ((double)time / (double)CLOCKS_PER_SEC);
28     }
29     return 0.; // else
30 }
31
32 DATA_TYPE f1_GCD(DATA_TYPE variableN1, DATA_TYPE variableN2){
33     DATA_TYPE returnValue = 1;
34
35     for(DATA_TYPE i = 1, k = MIN(variableN1, variableN2); i <= k; i++){
36         if(!(variableN1 % i || variableN2 % i)){
37             returnValue = i;
38         }
39     }
40
41     return returnValue;
42 }
43
44 DATA_TYPE f2_GCD(DATA_TYPE a, DATA_TYPE b){
45     for(DATA_TYPE aModB;
46         aModB = a % b,
47         a = b,
48         b = aModB;
49     );
50
51     return a;
52 }
53
54 DATA_TYPE f3_GCD(DATA_TYPE a, DATA_TYPE b){
55     if(!b){
56         return a;
57     }
58
59     return f3_GCD(b, a % b); // else
60 }
```

```
59         return f3_GCD(b, a % b); // else
60     }
61
62     int main() {
63         DATA_TYPE a = MAX(N1, N2), b = MIN(N1, N2),    returnValue;
64
65         double startTime, endTime;
66
67         // f1_GCD
68         startTime = getCurrentTime();
69         REPEATOR(REPEAT_COUNT, returnValue = f1_GCD(a, b));
70         endTime = getCurrentTime();
71         printf("f1_GCD return %lld \r\nrun time: %dns\r\n\r\n",
72             returnValue,
73             (unsigned int)((endTime - startTime) * (double)(NANOSECONDS_PER_SECOND_NUMBER / REPEAT_COUNT)));
74
75         // f2_GCD
76         startTime = getCurrentTime();
77         REPEATOR(REPEAT_COUNT, returnValue = f2_GCD(a, b));
78         endTime = getCurrentTime();
79         printf("f2_GCD return %lld \r\nrun time: %dns\r\n\r\n",
80             returnValue,
81             (unsigned int)((endTime - startTime) * (double)(NANOSECONDS_PER_SECOND_NUMBER / REPEAT_COUNT)));
82
83         // f3_GCD
84         startTime = getCurrentTime();
85         REPEATOR(REPEAT_COUNT, returnValue = f3_GCD(a, b));
86         endTime = getCurrentTime();
87         printf("f3_GCD return %lld \r\nrun time: %dns\r\n\r\n",
88             returnValue,
89             (unsigned int)((endTime - startTime) * (double)(NANOSECONDS_PER_SECOND_NUMBER / REPEAT_COUNT)));
90
91         printf("Press Enter to continue . . .");
92         (void)getchar();
93
94         return 0;
95     }
```

acmlab1.cpp

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define NANoseconds_PER_SECOND_NUMBER 1000000000
6
7  #define DATA_TYPE unsigned long long int
8
9  #define N1 18
10 #define N2 27
11
12 #define MIN(a,b) (((a)<(b))?(a):(b))
13 #define MAX(a,b) (((a)>(b))?(a):(b))
14
15 #define REPEAT_COUNT 1000000
16 #define REPEATOR(count, code) \
17 for (unsigned int indexIteration = (count);
18     indexIteration--;){ code; }
19
20 double getCurrentTime() {
21     clock_t time = clock();
22     if (time != (clock_t)-1) {
23         return ((double)time / (double)CLOCKS_PER_SEC);
24     }
25     return 0.; // else
26 }
27
28 DATA_TYPE f1_GCD(DATA_TYPE variableN1, DATA_TYPE
29 variableN2) {
30     DATA_TYPE returnValue = 1;
31
32     for (DATA_TYPE i = 1, k = MIN(variableN1, variableN2);
33         i <= k; i++) {
34         if (!(variableN1 % i || variableN2 % i)) {
35             returnValue = i;
36         }
37     }
38     return returnValue;
39 }

```

Console

Shell

```

clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_7.0.1-~)
./clang-7 -pthread -lm -o main acmlab1.cpp -O0
./main
f1_GCD return 9
run time: 247ns

f2_GCD return 9
run time: 21ns

f3_GCD return 9
run time: 26ns

Press Enter to continue . . .
./clang-7 -pthread -lm -o main acmlab1.cpp -O1
./main
f1_GCD return 9
run time: 0ns

f2_GCD return 9
run time: 0ns

f3_GCD return 9
run time: 0ns

Press Enter to continue . . .

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define NANSECONDS_PER_SECOND_NUMBER 1000000000
6
7  #define DATA_TYPE volatile unsigned long long int
8
9  #define N1 18
10 #define N2 27
11
12 #define MIN(a,b) (((a)<(b))?(a):(b))
13 #define MAX(a,b) (((a)>(b))?(a):(b))
14
15 #define REPEAT_COUNT 1000000
16 #define REPEAT(count, code) \
17 for (unsigned int indexIteration = (count);
18     indexIteration--){ code; }
19
20 double getCurrentTime() {
21     clock_t time = clock();
22     if (time != (clock_t)-1) {
23         return ((double)time / (double)CLOCKS_PER_SEC);
24     }
25     return 0.; // else
26 }
27
28 DATA_TYPE f1_GCD(DATA_TYPE variableN1, DATA_TYPE
29 variableN2) {
30     DATA_TYPE returnValue = 1;
31
32     for (DATA_TYPE i = 1, k = MIN(variableN1, variableN2);
33         i <= k; i++) {
34         if (!(variableN1 % i || variableN2 % i)) {
35             returnValue = i;
36         }
37     }
38     return returnValue;

```

```

clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_7.0.0-1)
clang-7 -pthread -lm -o main acmlab1.cpp -O0
./main
f1_GCD return 9
run time: 232ns

f2_GCD return 9
run time: 22ns

f3_GCD return 9
run time: 26ns

Press Enter to continue . . .
clang-7 -pthread -lm -o main acmlab1.cpp -O1
./main
f1_GCD return 9
run time: 222ns

f2_GCD return 9
run time: 22ns

f3_GCD return 9
run time: 21ns

Press Enter to continue . . .

```







# Алгоритми та методи обчислень

*Лекція №2*

**Тема №1. Вступ до теорії  
алгоритмів.**

**1.2. Формалізація поняття  
алгоритму.**

# 1.2. Формалізація поняття алгоритму.

1.2.1. Введення в теорію алгоритмів.

1.2.2. Абстрактні моделі алгоритму.

1.2.3. Формальні алгоритмічні системи(ФАС).

1.2.4. Скінченний автомат.

1.2.4.1. Детермінований скінченний автомат(DFA).

1.2.4.2. Недетермінований скінченний автомат(NFA).

1.2.5. Перетворювачі(трансдуктори) на основі детермінованого скінченного автомату.

1.2.5.1. Автомат Мура(Moore machine).

1.2.5.2. Автомат Мілі(Mealy machine).

1.2.6. Магазинний автомат(PDA).

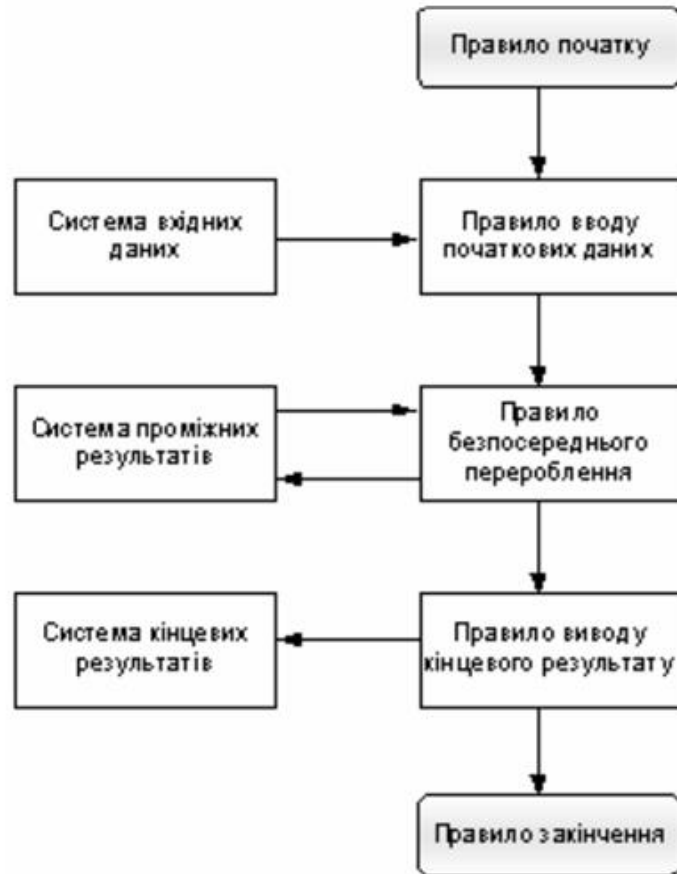
1.2.7. Машина Тюрінга та теза Черча.

1.2.8. Машина Поста.

1.2.9. Нормальні алгоритми Маркова.

# 1.2.1. Введення в теорію алгоритмів.

Черкаський М.В. / Електротехнічні та комп'ютерні системи № 04(80), 2011



Для формального опису параметричної моделі використаємо кортеж параметрів:

$$M_{par} : \langle A, Q, q_0, q_f, I, O, W \rangle,$$

де  $A$  – кінцева множина символів зовнішнього алфавіту, які позначають елементи систем вхідних даних, проміжних та кінцевих результатів;  $Q$  – кінцева множина символів внутрішнього алфавіту;  $q_0, q_f$  – початковий і кінцевий стани роботи моделі,  $q_0, q_f \in Q$ ;  $I, O$  – правила вводу і виводу;  $W$  – правило безпосереднього перероблення.

Розглянута параметрична модель, назвемо її узагальненою, може бути використана як основа для конструювання та дослідження нових і вже існуючих моделей. Ці моделі складають два класи: абстрактні алгоритми і формальні алгоритмічні системи. Вони розрізняються напрямком використання, а також повнотою та деталізацією параметрів алгоритму.

## 1.2.2. Абстрактні моделі алгоритму. (за Черкаським М.В.)

Черкаський М.В. / Електротехнічні та комп'ютерні системи № 04(80), 2011

Модель алгоритму належать до класу абстрактних, якщо правило безпосереднього перероблення не містить засобів здійснення обчислень:  $G \not\subset W$ , де  $G$  – конфігурація “апаратних” засобів.

Поширеним підкласом моделей абстрактних алгоритмів є функціональні залежності  $F$ :  $y = f(x)$ ;  $F \subset W$ . З параметричної моделі тут зафіксовані лише правило безпосереднього перероблення, системи вхідних даних та кінцевих результатів. Інші параметри ігноруються:

$$M_{af} : \langle A, F \rangle,$$

де  $F$  – алгоритм у вигляді функціональної залежності.

До іншого підкласу абстрактних алгоритмів належать блок-схеми програм і програми на мовах високого рівня. Розробка та практичне використання моделей цього підкласу перетворилося в одну з провідних індустрій сучасності. Модель програми містить всі параметри, задекларовані в параметричній моделі. Є тільки одна особливість: правило безпосереднього перероблення задано лише набором взаємозв'язаних інструкцій – програмою. Правило безпосереднього перероблення не містить засобів виконання інструкцій. Програма відокремлена від засобів її реалізації:  $P \subset W$ . Модель визначається наступним кортежем:

$$M_{pr} : \langle A, Q, q_0, q_f, I, O, P \rangle.$$

# Псевдо SH-модель. (за Черкаським М.В.)

Черкаський М.В. / Електротехнічні та комп'ютерні системи № 04(80), 2011

Ця абстрактна модель програми розглядає сукупність інструкцій  $\{B\}$  як множину об'єктів, зв'язаних між собою множиною з'єднань  $\{U\}$ . Правило безпосереднього перероблення псевдо SH-моделі задано цими двома множинами, що не перетинаються. Параметричний опис псевдо SH-моделі такий:

$$M_{ps} : \langle A, Q, q_0, q_f, I, O, G_s \rangle,$$

де  $G_s = (B, U)$  – граф, який відображає структуру блок-схеми програми,  $B$  – множина об'єктів,  $U$  – множина з'єднань між об'єктами.

У визначенні немає параметра  $P$  – програми, тому що модель  $M_{ps}$  сама є програмою. В перелік характеристик складності псевдо SH-моделі, крім часової, входять також об'єктна і структурна складності. Об'єкт це інструкція програми. Об'єктна складність дорівнює потужності множини об'єктів. Структурна складність оцінює ступінь нерегулярності зв'язків блок-схеми програми. Вона визначається логарифмічною мірою степені нерівномірності матриці інцидентів моделі. Структурна складність є інформаційною характеристикою. Одиницею структурної і об'єктної складності є біт.



# 1.2.3. Формальні алгоритмічні системи(ФАС).

## *ФАС з уявними засобами виконання інструкцій та SH-модель алгоритму*

Черкаський М.В. / Електротехнічні та комп'ютерні системи № 04(80), 2011

Клас алгоритмів, в яких правило безпосереднього перероблення задано програмою і засобами виконання інструкцій, отримав назву “Формальні алгоритмічні системи” (ФАС). Моделі ФАС можна поділити на два підкласи. До першого належать моделі ФАС з уявними засобами виконання інструкцій. Прикладами є машина Тюрінга, нормальні алгоритми Маркова, система Колмогорова та ін. Формальне визначення моделей алгоритму дається з використанням параметрів. Наприклад, машина Тюрінга визначається шістькою параметрів:

$$M_{mT} : \langle A, Q, a_0, q_f, q_0, P \rangle,$$

де  $a_0$  – позначення порожньої комірки стрічки,  $a_0 \in A$ ;  $P$  – програма [5].

До другого класу ФАС належить апаратно-програмна модель алгоритму. У визначенні цієї моделі у явній формі зафіксована наявність апаратних і програмних засобів виконання інструкцій. Правило безпосереднього перероблення задано апаратно-програмними засобами:

$$W = (G, P),$$

де  $G$  – конфігурація апаратних засобів,

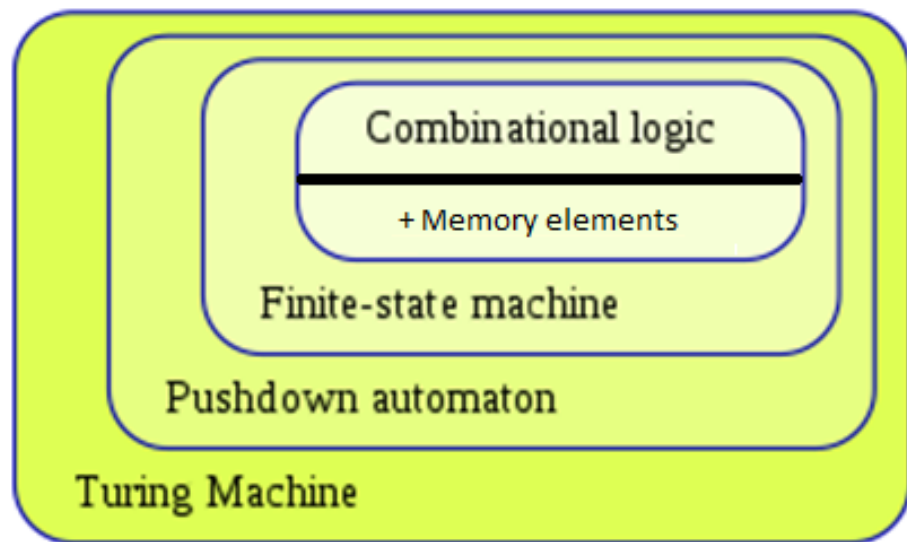
$$G = (X, U),$$

де  $X$  – множина елементарних перетворювачів,  $U$  – множина з'єднань елементарних перетворювачів;  $P$  – програма.

Модель апаратно-програмної ФАС має назву “SH-модель алгоритму” (S – Software, H – Hardware). Параметричний опис SH-моделі:

$$M_{SH} : \langle A, Q, a_0, q_f, q_0, I, O, G, P \rangle.$$



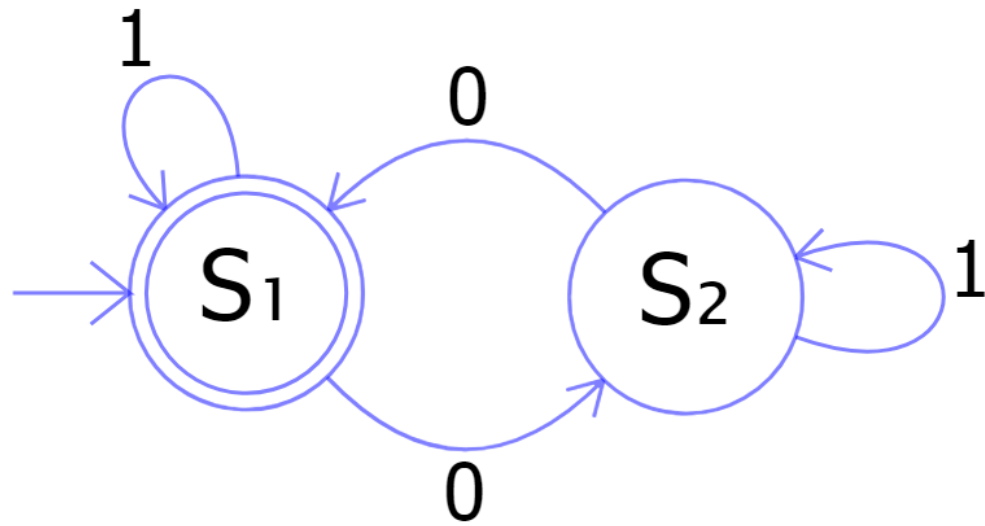


## 1.2.4. Скінченний автомат.

### 1.2.4.1. Детермінований скінченний автомат(DFA).

*детермінований скінченний автомат або детермінований скінченний автомат акцептор є  $(\Sigma, S, s_0, \delta, F)$ , де:*

- $\Sigma$  вхідна абетка (скінченний, не порожній набір символів).
- $S$  — скінченний, не порожній набір станів.
- $s_0$  — початковий стан, елемент з  $S$ .
- $\delta$  — функція переходу:  $\delta : S \times \Sigma \rightarrow S$
- $F$  набір кінцевих станів, (можливо порожня) підмножина  $S$ .



# Домашнє завдання №1

З використанням реалізації поведінки детермінованого кінцевого автомату написати на C/C++ програму(*англ. automata-based programming*) виявлення ділянки тексту у вхідній стрічці. Програма має лише відображати перехід моделі автомату у кінцевий стан без виявлення позиції входження шуканої ділянки тексту.

## Вибір варіанту

Шукана ділянка тексту це прізвище студента записане латинськими літерами.

```
#include "stdio.h"
```

```
#define DECLSTATE(NAME, ...) typedef enum {__VA_ARGS__, size##NAME} NAME;
```

```
#define GET_ENUM_SIZE(NAME) size##NAME
```

```
DECLSTATE(A,
```

```
    vA/*A*/,
```

```
    vB/*B*/,
```

```
    vC/*C*/,
```

```
    vD/*D*/,
```

```
    vE/*E*/,
```

```
    vF/*F*/,
```

```
    vG/*G*/,
```

```
    vH/*H*/,
```

```
    vI/*I*/,
```

```
    vJ/*J*/,
```

```
    vK/*K*/,
```

```
    vL/*L*/,
```

```
    vM/*M*/,
```

```
    vN/*N*/,
```

```
    vO/*O*/,
```

```
    vP/*P*/,
```

```
    vQ/*Q*/,
```

```
    vR/*R*/,
```

```
    vS/*S*/,
```

```
    vT/*T*/,
```

```
    vU/*U*/,
```

```
    vV/*V*/,
```

```
    vW/*W*/,
```

```
    vX/*X*/,
```

```
    vY/*Y*/,
```

```
    vZ/*Z*/,
```

```
    sZ/*[*/,
```

```
    sY/*\*/,
```

```
    sC/*]*/,
```

```
    sH/*^*/,
```

```
    sB/*_*/,
```

```
    sT/*`*/,
```

```
    va/*a*/,
```

```
    vb/*b*/,
```

```
    vc/*c*/,
```

```
    vd/*d*/,
```

```
    ve/*e*/,
```

```
vf/*f*/,
vg/*g*/,
vh/*h*/,
vi/*i*/,
vj/*j*/,
vk/*k*/,
vl/*l*/,
vm/*m*/,
vn/*n*/,
vo/*o*/,
vp/*p*/,
vq/*q*/,
vr/*r*/,
vs/*s*/,
vt/*t*/,
vu/*u*/,
vv/*v*/,
vw/*w*/,
vx/*x*/,
vy/*y*/,
vz/*z*/,
sF/*end mark*/
)
```

```
DECLSTATE(Q,
q0,
q1,
q2,
q3,
q4,
qf
)
```

```
typedef unsigned char INSTRUCTION;  
typedef INSTRUCTION PROGRAM[GET_ENUM_SIZE(A)][GET_ENUM_SIZE(Q)];  
  
PROGRAM program = {  
    //      q0  q1  q2  q3  q4  qf  
    /*A*/{q0, q0, q0, q0, q0, qf},  
    /*B*/{q0, q0, q0, q0, q0, qf},  
    /*C*/{q0, q0, q0, q0, q0, qf},  
    /*D*/{q0, q0, q0, q0, q0, qf},  
    /*E*/{q0, q0, q0, q0, q0, qf},  
    /*F*/{q0, q0, q0, q0, q0, qf},  
    /*G*/{q0, q0, q0, q0, q0, qf},  
    /*H*/{q0, q0, q0, q0, q0, qf},  
    /*I*/{q0, q0, q0, q0, q0, qf},  
    /*J*/{q0, q0, q0, q0, q0, qf},  
    /*K*/{q1, q0, q0, q0, q0, qf},  
    /*L*/{q0, q0, q0, q0, q0, qf},  
    /*M*/{q0, q0, q0, q0, q0, qf},  
    /*N*/{q0, q0, q0, q0, q0, qf},  
    /*O*/{q0, q0, q0, q0, q0, qf},  
    /*P*/{q0, q0, q0, q0, q0, qf},  
    /*Q*/{q0, q0, q0, q0, q0, qf},  
    /*R*/{q0, q0, q0, q0, q0, qf},  
    /*S*/{q0, q0, q0, q0, q0, qf},  
    /*T*/{q0, q0, q0, q0, q0, qf},  
    /*U*/{q0, q0, q0, q0, q0, qf},  
    /*V*/{q0, q0, q0, q0, q0, qf},  
    /*W*/{q0, q0, q0, q0, q0, qf},  
    /*X*/{q0, q0, q0, q0, q0, qf},  
    /*Y*/{q0, q0, q0, q0, q0, qf},  
    /*Z*/{q0, q0, q0, q0, q0, qf},  
    /*[*]{q0, q0, q0, q0, q0, qf},
```

```
/*\*/{q0, q0, q0, q0, q0, qf},
/*]*/{q0, q0, q0, q0, q0, qf},
/*^*/{q0, q0, q0, q0, q0, qf},
/*_*/{q0, q0, q0, q0, q0, qf},
/*`*/{q0, q0, q0, q0, q0, qf},
/*a*/{q0, q0, q0, q4, q0, qf},
/*b*/{q0, q0, q0, q0, q0, qf},
/*c*/{q0, q0, q0, q0, q0, qf},
/*d*/{q0, q0, q0, q0, q0, qf},
/*e*/{q0, q0, q0, q0, q0, qf},
/*f*/{q0, q0, q0, q0, q0, qf},
/*g*/{q0, q0, q0, q0, q0, qf},
/*h*/{q0, q0, q0, q0, q0, qf},
/*i*/{q0, q0, q0, q0, q0, qf},
/*j*/{q0, q0, q0, q0, q0, qf},
/*k*/{q0, q0, q0, q0, qf, qf},
/*l*/{q0, q0, q0, q0, q0, qf},
/*m*/{q0, q0, q0, q0, q0, qf},
/*n*/{q0, q0, q0, q0, q0, qf},
/*o*/{q0, q2, q0, q0, q0, qf},
/*p*/{q0, q0, q0, q0, q0, qf},
/*q*/{q0, q0, q0, q0, q0, qf},
/*r*/{q0, q0, q0, q0, q0, qf},
/*s*/{q0, q0, q0, q0, q0, qf},
/*t*/{q0, q0, q0, q0, q0, qf},
/*u*/{q0, q0, q0, q0, q0, qf},
/*v*/{q0, q0, q0, q0, q0, qf},
/*w*/{q0, q0, q0, q0, q0, qf},
/*x*/{q0, q0, q0, q0, q0, qf},
/*y*/{q0, q0, q0, q0, q0, qf},
/*z*/{q0, q0, q3, q0, q0, qf}
};
```



```

typedef struct structDFA{
    unsigned char * data;
    PROGRAM * program;
    void(*run)(struct structDFA * dfa);
    Q state;
} DFA;

void runner(DFA * dfa){
    for (; *dfa->data != sF; ++dfa->data){
        dfa->state = (*dfa->program)[*dfa->data][dfa->state];
    }
}

#define MAX_TEXT_SIZE 256
int main(){
    unsigned char data[MAX_TEXT_SIZE] = "Kozak Nazar Bohdanovych", *data_ = data;
    DFA dfa = { data, (PROGRAM*)program, runner, q0 };

    for (; *data_; *data_ -= 'A', *data_ %= GET_ENUM_SIZE(A), ++data_);
    *data_ = sF;
    dfa.run(&dfa);

    if (dfa.state == qf){
        printf("DFA: finit state\r\n");
    }
    else{
        printf("DFA: no finit state\r\n");
    }

    getchar();
    return 0;
}

```

```
#include "stdio.h"
```

```
#define DECLSTATE(NAME, ...) typedef enum {__VA_ARGS__, size##NAME} NAME;
```

```
#define GET_ENUM_SIZE(NAME) size##NAME
```

```
DECLSTATE(A,
```

```
    vA/*A*/,
```

```
    vB/*B*/,
```

```
    vC/*C*/,
```

```
    vD/*D*/,
```

```
    vE/*E*/,
```

```
    vF/*F*/,
```

```
    vG/*G*/,
```

```
    vH/*H*/,
```

```
    vI/*I*/,
```

```
    vJ/*J*/,
```

```
    vK/*K*/,
```

```
    vL/*L*/,
```

```
    vM/*M*/,
```

```
    vN/*N*/,
```

```
    vO/*O*/,
```

```
    vP/*P*/,
```

```
    vQ/*Q*/,
```

```
    vR/*R*/,
```

```
    vS/*S*/,
```

```
    vT/*T*/,
```

```
    vU/*U*/,
```

```
    vV/*V*/,
```

```
    vW/*W*/,
```

```
    vX/*X*/,
```

```
    vY/*Y*/,
```

```
    vZ/*Z*/,
```

```
    sZ/*[*/,
```

```
    sY/*\*/,
```

```
    sC/*]*/,
```

```
    sH/*^*/,
```

```
    sB/*_*/,
```

```
    sT/*`*/,
```

```
    va/*a*/,
```

```
    vb/*b*/,
```

```
    vc/*c*/,
```

```
    vd/*d*/,
```

```
    ve/*e*/,
```

```
    vf/*f*/.
```

vg/\*g\*,  
vh/\*h\*,  
vi/\*i\*,  
vj/\*j\*,  
vk/\*k\*,  
vl/\*l\*,  
vm/\*m\*,  
vn/\*n\*,  
vo/\*o\*,  
vp/\*p\*,  
vq/\*q\*,  
vr/\*r\*,  
vs/\*s\*,  
vt/\*t\*,  
vu/\*u\*,  
vv/\*v\*,  
vw/\*w\*,

```

vx/*x*/,
vy/*y*/,
vz/*z*/,
sF/*end mark*/
)

```

```

DECLSTATE(Q,
q0,
q1,
q2,
q3,
q4,
qf
)

```

```
typedef unsigned char INSTRUCTION;
```

```
#define MAX_RULE_COUNT 16
```

```
typedef struct {
    unsigned char input;
    INSTRUCTION toState;

```

```
} SimpleRule;
```

```
typedef SimpleRule PROGRAM[GET_ENUM_SIZE(Q)][MAX_RULE_COUNT];
```

```
#define DEFAULT GET_ENUM_SIZE(A)
```

```

PROGRAM program = {
    //          q0                      q1                      q2
q3            { { vk, q1 }, { DEFAULT, q0 } }, { { vo, q2 }, { DEFAULT, q0 } }, { { vz, q3 }, {
DEFAULT, q0 } }, { { va, q4 }, { DEFAULT, q0 } }, { { vk, qf }, { DEFAULT, q0 } }, { {
DEFAULT, qf } },
};

```

```

typedef struct structDFA{
    unsigned char * data;
    PROGRAM * program;
    void(*run)(struct structDFA * dfa);
    Q state;
} DFA;

```

```

void runner(DFA * dfa){
    for (; *dfa->data != sF; ++dfa->data){
        SimpleRule *prevRule, *rule;
        for (prevRule = rule = (*dfa->program)[dfa->state]; prevRule->input !=
DEFAULT; prevRule = rule, ++rule){
            if (DEFAULT == rule->input){
                dfa->state = rule->toState;
                break;
            }
            else if (*dfa->data == rule->input){
                dfa->state = rule->toState;
                break;
            }
        }
    }
}

#define MAX_TEXT_SIZE 256
int main(){
    unsigned char data[MAX_TEXT_SIZE] = "Kozak Nazar Bohdanovych", *data_ = data;
    DFA dfa = { data, (PROGRAM*)program, runner, q0 };

    for (; *data_; *data_ -= 'A', *data_ %= GET_ENUM_SIZE(A), ++data_);
    *data_ = sF;
    dfa.run(&dfa);

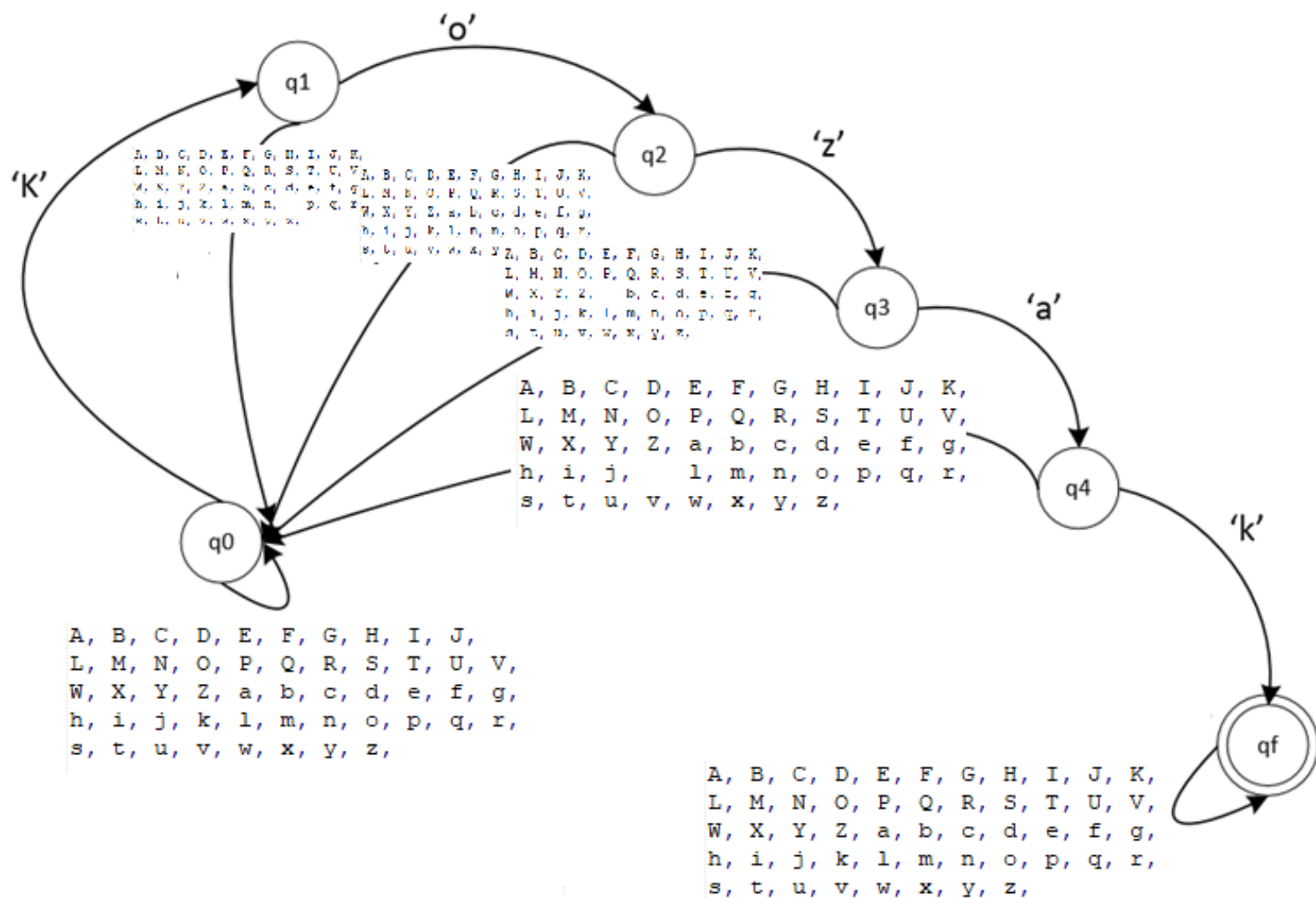
    if (dfa.state == qf){
        printf("DFA: finit state\r\n");
    }
    else{
        printf("DFA: no finit state\r\n");
    }

    getchar();
    return 0;
}

```

# Контрольне завдання №3

**Нарисувати граф, що відображає детермінований кінцевий автомат для виявлення у вхідній стрічці прізвища студента.**



## 1.2.4. Скінченний автомат.

### 1.2.4.1. Недетермінований скінченний автомат(NFA).

**Недетермінований автомат** це автомат, який при даному вхідному символі і внутрішньому стані може переходити в декілька різних внутрішніх станів.

НСА формально представляє п'ятірка,  $(Q, \Sigma, \Delta, q_0, F)$ , в якій:

- скінченна множина станів  $Q$
- скінченна множина вхідних символів  $\Sigma$
- функція переходу  $\Delta : Q \times \{\Sigma \cup \varepsilon\} \rightarrow P(Q)$ .
- початковий стан  $q_0 \in Q$
- набір станів  $F$  позначених як *допустимі* (або *кінцеві*) *стани*  $F \subseteq Q$ .

Тут,  $P(Q)$  позначає множину всіх підмножин  $Q$ . Нехай  $w = a_1 a_2 \dots a_n$  буде словом в абетці  $\Sigma$ .

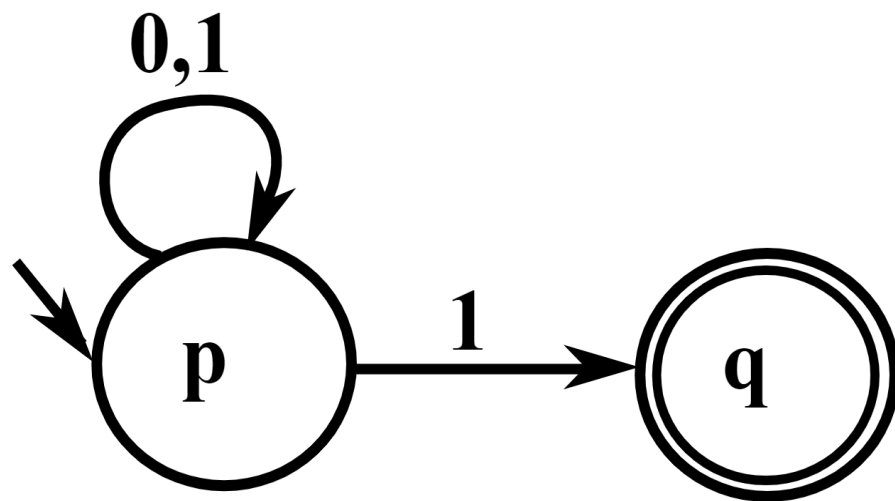
Автомат  $M$  приймає слово  $w$  якщо послідовність станів,  $r_0, r_1, \dots, r_n$ , існує в  $Q$  з такими умовами:

$$r_0 = q_0$$

$$r_{i+1} \in \Delta(r_i, a_{i+1}), \text{ for } i = 0, \dots, n-1$$

$$r_n \in F.$$



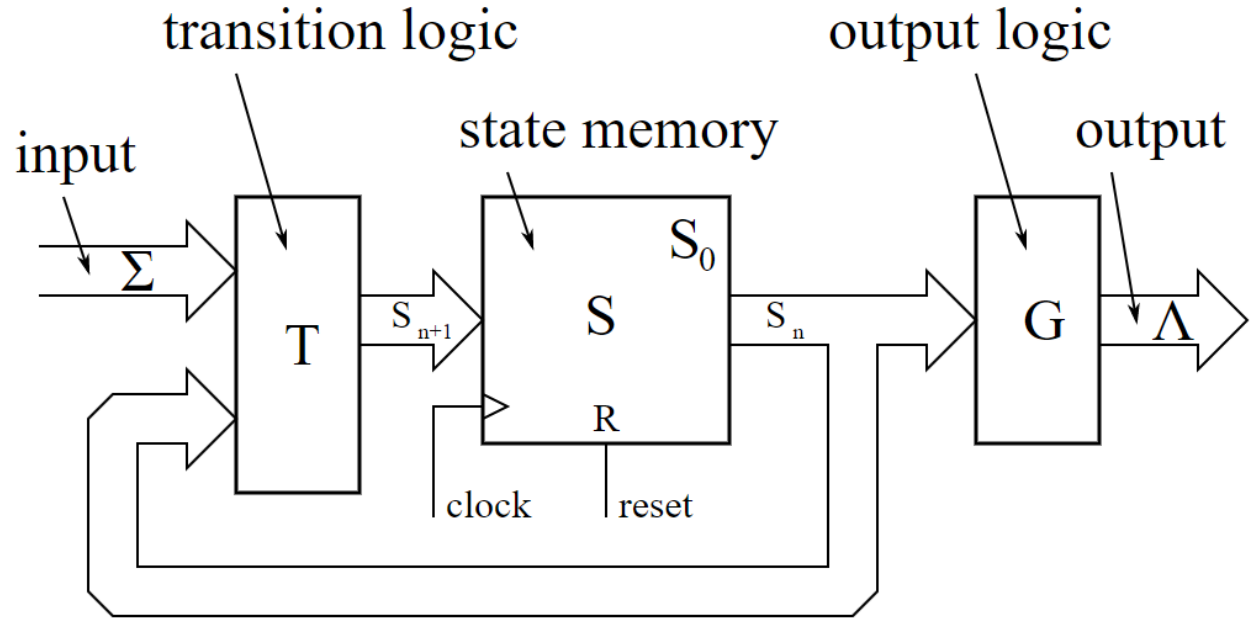


## 1.2.5. Перетворювачі(трансдуктори) на основі детермінованого скінченного автомату.

### 1.2.5.1. Автомат Мура(Moore machine).

Автомат Мура це 6 елементів ( $S$ ,  $S_0$ ,  $\Sigma$ ,  $\Lambda$ ,  $T$ ,  $G$ ):

- множина внутрішніх станів  $S$  (внутрішній алфавіт);
- початковий стан  $S_0$ , який є елементом ( $S$ );
- скінченна множина вхідних сигналів  $\Sigma$  (вхідний алфавіт);
- скінченна множина вихідних сигналів  $\Lambda$  (вихідний алфавіт);
- функція переходу ( $T: S \times \Sigma \rightarrow S$ ), яка відображає стан і вхідний алфавіт у наступний стан;
- вихідна функція ( $G: S \rightarrow \Lambda$ ), яка відображає стан у вихідний алфавіт.

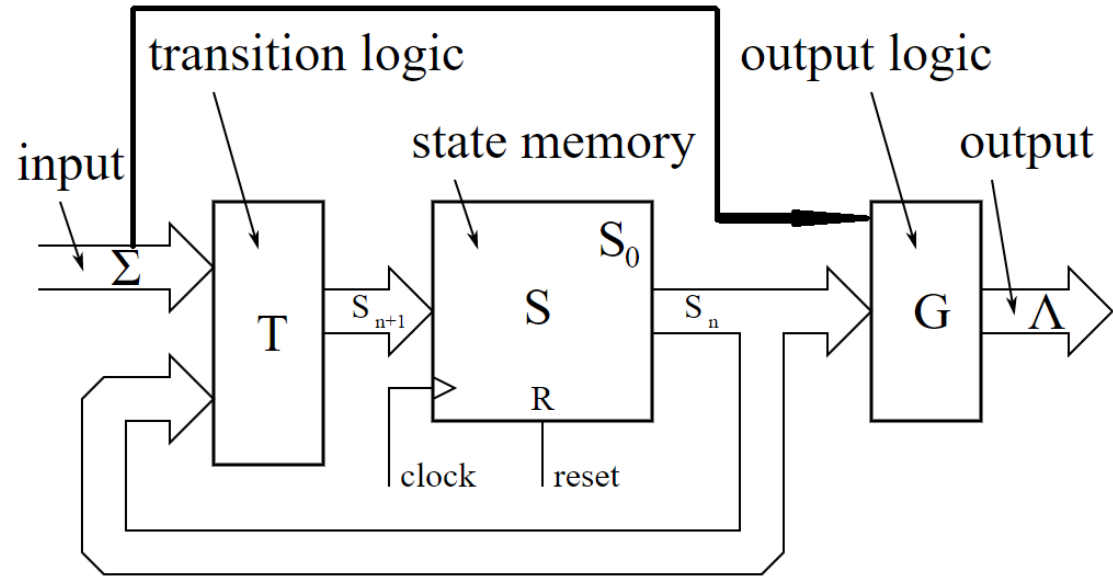


## 1.2.5. Перетворювачі(трансдуктори) на основі детермінованого скінченного автомату.

### 1.2.5.2. Автомат Мілі(Mealy machine).

Автомат Мілі це 6 елементів ( $S$ ,  $S_0$ ,  $\Sigma$ ,  $\Lambda$ ,  $T$ ,  $G$ ):

- множина внутрішніх станів  $S$  (внутрішній алфавіт);
- початковий стан  $S_0$ , який є елементом ( $S$ );
- скінченна множина входніх сигналів  $\Sigma$  (вхідний алфавіт);
- скінченна множина вихідних сигналів  $\Lambda$  (вихідний алфавіт);
- функція переходу ( $T: S \times \Sigma \rightarrow S$ ), яка відображає стан і вхідний алфавіт у наступний стан;
- вихідна функція ( $G: S \rightarrow \Lambda$ ), яка відображає стан і вхідний алфавіт у вихідний алфавіт.



# 1.2.6. Магазинний автомат(PDA).

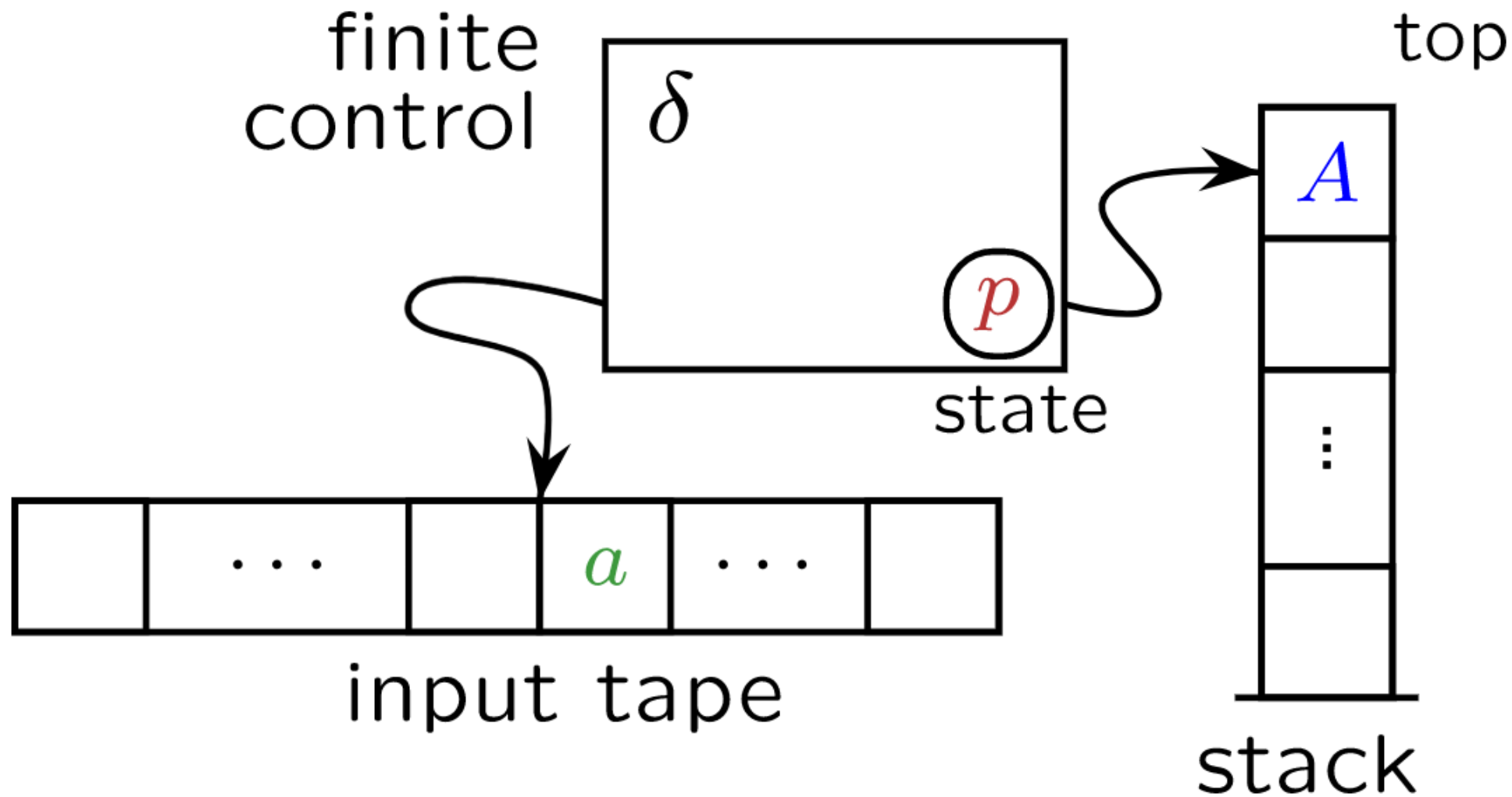
Автомат з магазинною пам'яттю (МП автомат) використовує стек для зберігання станів.

$M = (K, \Sigma, \pi, s, F, S, m)$ , де

- $K$  — скінченна множина станів автомата
- $s \in K$  — єдиний допустимий початковий стан автомата
- $F \subset K$  — множина кінцевих станів, причому допускається  $F=\emptyset$ , і  $F=K$
- $\Sigma$  — скінченна множина символів вхідного алфавіту, з якого формуються строки, що зчитуються автоматом
- $S$  — алфавіт пам'яті (магазину)
- $m \in S$  — нульовий символ пам'яті.
- $\pi$  — функція переходів:  $\pi : K \times \Sigma \times S \rightarrow K \times S$

Пам'ять працює як стек, тобто для читання доступний останній записаний в неї елемент. Таким чином, функція переходу є відображенням  $\pi : K \times \Sigma \times S \rightarrow K \times S$ . Тобто, по комбінації поточного стану, вхідного символу і символу на вершині магазину автомат вибирає наступний стан і, можливо, символ для запису в магазин. У випадку, коли у правій частині автоматного правила присутній  $\epsilon$ , в магазин нічого не додається, а елемент з вершини стирається. Якщо магазин порожній, то спрацьовують правила з  $\epsilon$  в лівій частині.

У чистому вигляді автомати з магазинною пам'яттю використовуються вкрай рідко. Зазвичай ця модель використовується для наочного подання відмінності звичайних скінченних автоматів від синтаксичних граматики. Реалізація автоматів з магазинною пам'яттю відрізняється від кінцевих автоматів тим, що поточний стан автомата сильно залежить від будь-якого попереднього.

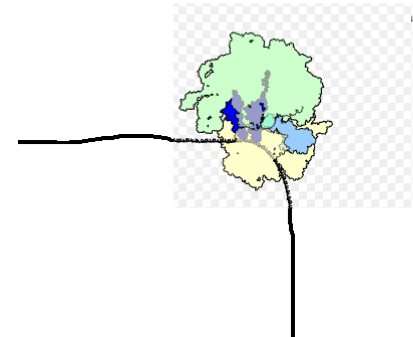
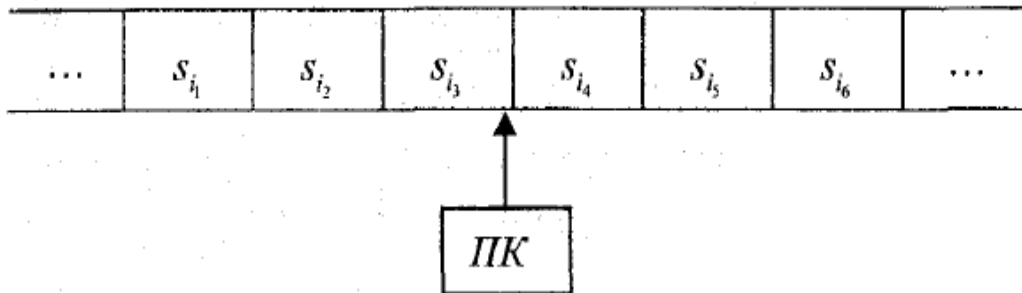


# 1.2.7. Машина Тюрінга та теза Черча.

## 1.2.7.1. Машина Тюрінга.

### Загальна концепція

- На змістовному рівні машина Тьюрінга (МТ) є деякою гіпотетичною(умовною) машиною, яка складається з трьох головних компонентів: *інформаційної стрічки, головки для зчитування і запису та пристрою керування.*
- В клітинах живих організмів є кілька механізмів, які є аналогами машини Тюрінга



# 1.2.7. Машина Тюрінга та теза Черча.

## 1.2.7.1. Машина Тюрінга.

### Формальне визначення

Модель однострічкової детермінованої МТ:

$$M = \langle A, I, Q, q_0, q_f, a_0, p \rangle,$$

де  $A$  – кінцева множина символів зовнішнього алфавіту,

$I$  – кінцева множина символів зовнішнього алфавіту на стрічці,

$Q$  – кінцева множина символів внутрішнього алфавіту,

$q_0$  – початковий стан,

$q_f$  – кінцевий стан,

$$q_0, q_f \in Q$$

$a_0$  – позначення порожньої комірки стрічки,

$p$  – така програма, яка не може мати двох команд, у яких би збігалися два перші символи:

$$\{A\} \times \{Q\} \ni \{A\}\{L, R, S\}\{Q\},$$

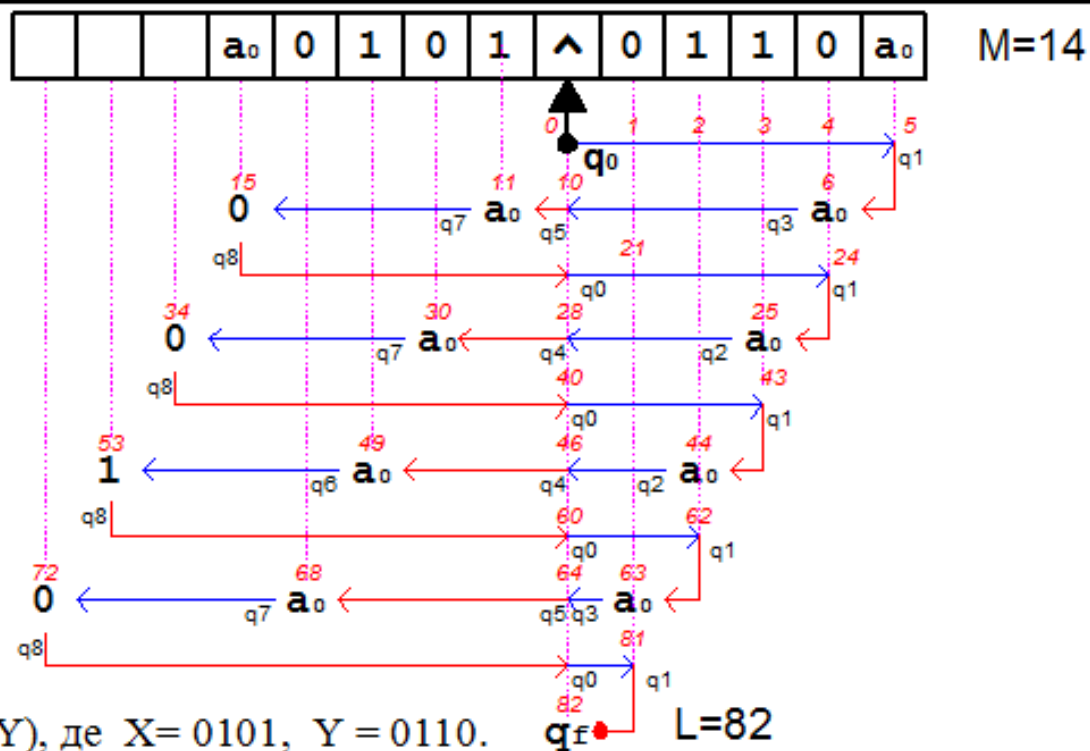
де  $L$  – зсувати головку вліво,

$R$  – зсувати головку вправо,

$S$  – головка залишається на місці.

$A^Q$	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$	$q_7$	$q_8$
<b>a</b> <sub>0</sub>	Lq <sub>1</sub>				Lq <sub>4</sub>	Lq <sub>5</sub>	1Rq <sub>8</sub>	0Rq <sub>8</sub>	Rq <sub>8</sub>
<b>0</b>	Rq <sub>0</sub>	a <sub>0</sub> Lq <sub>3</sub>	Lq <sub>2</sub>	Lq <sub>3</sub>	a <sub>0</sub> Lq <sub>7</sub>	a <sub>0</sub> Lq <sub>7</sub>	Lq <sub>6</sub>	Lq <sub>7</sub>	Rq <sub>8</sub>
<b>1</b>	Rq <sub>0</sub>	a <sub>0</sub> Lq <sub>2</sub>	Lq <sub>2</sub>	Lq <sub>3</sub>	a <sub>0</sub> Lq <sub>6</sub>	a <sub>0</sub> Lq <sub>7</sub>	Lq <sub>6</sub>	Lq <sub>7</sub>	Rq <sub>8</sub>
<b>^</b>	Rq <sub>0</sub>	<u>q<sub>f</sub></u>	Lq <sub>4</sub>	Lq <sub>5</sub>					Rq <sub>0</sub>

**P = 29**





# Домашнє завдання №2

Виконати реалізацію моделі машини Тюрінга на C/C++ для виконання обчислень згідно власного варіанту.

## Вибір варіанту

$$(N_{\text{ж}} + N_{\text{г}} + 1) \% 7 + 1$$

де:  $N_{\text{ж}}$  – порядковий номер студента в групі, а  $N_{\text{г}}$  – номер групи (1,2,3,4,5,6,7,8 або 9)

## Варіанти завдання

№	обчислення	пояснення заданого виразу	
			C-нотация
1	$X \text{ NAND } Y$	побітове І-НЕ	<code>return ~(X &amp; Y);</code>
2	$X \text{ OR } Y$	побітове АБО	<code>return X   Y;</code>
3	$X \text{ NOR } Y$	побітове АБО-НЕ	<code>return ~(X   Y);</code>
4	$\text{NOT } X$	побітове заперечення X	<code>return ~X;</code>
5	$\text{NOT } Y$	побітове заперечення Y	<code>return ~Y;</code>
6	0	константа 0 (операція завжди повертає 0 для кожного біту)	<code>return 0;</code>
7	1	константа 1 (операція завжди повертає 1 для кожного біту)	<code>return ~0;</code>

```
#include <stdio.h>

#define DECLSTATE(NAME, ...) typedef enum {__VA_ARGS__, size##NAME} NAME;

#define GET_ENUM_SIZE(NAME) size##NAME

DECLSTATE(A,
a0,
v0/* 0 */,
v1/* 1 */,
sT/* ^ */
)

DECLSTATE(Q,
q0,
q1,
q2,
q3,
q4,
q5,
q6,
q7,
q8,
qf
```

```

    )

#define NO_ACTION 0x7f

#define NO_RULE {NO_ACTION, NO_ACTION, NO_ACTION}

#define S 0
#define L 1
#define R 2

typedef unsigned char INSTRUCTION[3];
typedef INSTRUCTION PROGRAM[GET_ENUM_SIZE(A)][GET_ENUM_SIZE(Q)];

PROGRAM initProgram = { /* default pass */
    //
    q3          q4          q5          q6          q7          q8          q0          q1          q2
    /* a0 */{ { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, {
NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, {
NO_ACTION, R, q1 }, { NO_ACTION, R, q1 } },
    /* 0 */{ { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, {
NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, {
NO_ACTION, R, q1 }, { NO_ACTION, R, q1 } },
    /* 1 */{ { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, {
NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, { NO_ACTION, R, q1 }, {
NO_ACTION, R, q1 }, { NO_ACTION, R, q1 } },
    /* ^ */{ { NO_ACTION, S, q0 }, { NO_ACTION, S, q0 }, { NO_ACTION, S, q0 }, {
NO_ACTION, S, q0 }, { NO_ACTION, S, q0 }, { NO_ACTION, S, q0 }, { NO_ACTION, S, q0 }, {
NO_ACTION, S, q0 }, { NO_ACTION, S, q0 } }
};

```

```

PROGRAM program = { /* default pass */
    //                q0                q1                q2
q3                q4                q5                q6
q7                q8
    /* a0 */{ { NO_ACTION, L, q1 }, NO_RULE, NO_RULE,
NO_RULE,      { NO_ACTION, L, q4 }, { NO_ACTION, L, q5 }, { v1, R, q8 }, {
v0, R, q8 },  { NO_ACTION, R, q8 } },
    /* 0 */{ { NO_ACTION, R, q0 }, { a0, L, q3 }, { NO_ACTION, L, q2 }, {
NO_ACTION, L, q3 }, { a0, L, q7 }, { a0, L, q7 }, { NO_ACTION, L, q6 }, {
NO_ACTION, L, q7 }, { NO_ACTION, R, q8 } },
    /* 1 */{ { NO_ACTION, R, q0 }, { a0, L, q2 }, { NO_ACTION, L, q2 }, {
NO_ACTION, L, q3 }, { a0, L, q6 }, { a0, L, q7 }, { NO_ACTION, L, q6 }, {
NO_ACTION, L, q7 }, { NO_ACTION, R, q8 } },
    /* ^ */{ { NO_ACTION, R, q0 }, { NO_ACTION, S, qf }, { NO_ACTION, L, q4 }, {
NO_ACTION, L, q5 }, NO_RULE, NO_RULE, NO_RULE,
NO_RULE,      { NO_ACTION, R, q0 } }
};

```

```

typedef unsigned char tapeElementType;
#define TAPE_SIZE 256

```

```

// #define MT_BEGIN_POSITION_STATE 127

```

```

typedef struct structMT{
    tapeElementType tape[TAPE_SIZE];
    PROGRAM * initProgram;
    PROGRAM * program;
    void(*stepRun)(struct structMT * mt, PROGRAM program);
    void(*run)(struct structMT * mt);
    void(*statePrint)(struct structMT * mt);
    unsigned char * debugType;
    Q state;
    unsigned int positionState;
}

```

```
} MT;

void stepRunner(MT * mt, PROGRAM program){
    INSTRUCTION * instruction;

    if (!mt){
        return;
    }

    instruction = program[mt->tape[mt->positionState]][mt->state];

    if ((*instruction)[0] != NO_ACTION){
        mt->tape[mt->positionState] = (*instruction)[0];
    }

    if ((*instruction)[1] == L){
        --mt->positionState;
    }
    else if ((*instruction)[1] == R){
        ++mt->positionState;
    }

    mt->state = (*instruction)[2];
}
```

```

void runner(MT * mt){
    if (!mt){
        return;
    }

    // init
    mt->state = q0;
    mt->positionState = 0;
    do{
        mt->stepRun(mt, mt->initProgram);
    } while (mt->state != q0);

    printf("Begin tape state:\r\n");
    mt->statePrint(mt);
    printf("\r\n\r\n");

    printf("Tape state:\r\n");
    // run program
    while (mt->state != qf){
        mt->stepRun(mt, mt->program);
        if (mt->statePrint){
            mt->statePrint(mt);
            if (*mt->debugType == 2){
                _sleep(250);
            }
            else if (*mt->debugType == 3){
                getchar();
            }
            else{
                _sleep(25);
            }
        }
    }
}

```

```

#define MAX_PRINT_TAPE_ELEMENT_COUNT 14
void statePrinter(MT * mt){
    unsigned int index = 0;

    if (!mt){
        return;
    }

    for (; index < MAX_PRINT_TAPE_ELEMENT_COUNT; ++index){
        switch (mt->tape[index]){
            case a0:
                if (index == mt->positionState)
                    printf("[a0] ");
                else{
                    printf(" a0 ");
                }
                break;
            case v0:
                if (index == mt->positionState)
                    printf("[0] ");
                else{
                    printf(" 0 ");
                }
                break;
            case v1:
                if (index == mt->positionState)
                    printf("[1] ");
                else{
                    printf(" 1 ");
                }
                break;
            case sT:
                if (index == mt->positionState)
                    printf("[^] ");
                else{
                    printf(" ^ ");
                }
                break;
            default:
                if (index == mt->positionState)
                    printf("[?] ");
                else{
                    printf(" ? ");
                }
                break;
        }
    }

    if (mt->state == qf){
        printf("    qf");
    }
    else{
        printf("    q%d", mt->state);
    }

    printf("\n");
}

```

```
tapeElementType tape[TAPE_SIZE];

#define INIT_TAPE_DATE { a0, a0, a0, a0, v1, v0, v1, v1, sT, v1, v0, v1, v1 }

int main(){
    int ch;
    MT mt = { INIT_TAPE_DATE, initProgram, program, stepRunner, runner, statePrinter,
(char*)&ch };
    printf("mode: \r\n default - run all\r\n      '2' - live\r\n      '3' - live by press
\r\n Enter\r\n");
    ch = getchar();
    ch -= '0';

    mt.run(&mt);
    getchar();

    return 0;
}
```





# Контрольне завдання №4


Показати «слід» машини Тюрінга при виконанні нею алгоритму для обчислення  $(X \wedge Y)$ , де  $\wedge$  – операції(логічна операція «І»). Визначити часову, програмну та місткісну складність алгоритму


$$X = (Nж \% 5),$$

$$Y = (Nж \% 10).$$

$Nж$  – порядковий номер студента в журналі

## Лекція №2

 Контрольне завдання №1

 Контрольне завдання №2

## Лекція №3

 Контрольне завдання №3

 Контрольне завдання №4 (виконати протягом тижня)

 Зразок контрольного завдання №4

# 1.2.7. Машина Тюрінга та теза Черча.

## 1.2.7.1. Машина Тюрінга.

### *Варіанти машини Тюрінга*

- Варіанти машини Тюрінга:
  - Змінна машина Тюрінга
  - Нейронна машина Тюрінга
  - Недетермінована машина Тюрінга
  - Квантова машина Тюрінга
  - Машина Поста-Тюрінга
  - Імовірнісна машина Тюрінга
  - Незмінна машина Тюрінга
  - Незмінна машина Тюрінга з рухом праворуч
  - Багатострічкова машина Тюрінга
  - Симетрична машина Тюрінга
  - Повна машина Тюрінга
  - Однозначна машина Тюрінга
  - Універсальна машина Тюрінга
  - Машина Зенона

## 1.2.7. Машина Тюрінга та теза Черча.

### 1.2.7.2. Теза Черча.

Клас алгоритмічно-обчислюваних функцій збігається з класом частково-рекурсивних функцій, функцій обчислюваних за Тюрінгом та іншими формальними уточненнями інтуїтивного поняття алгоритму.

З цього випливає, що якщо функція належить до класу певної формалізації алгоритмічно-обчислюваної функції, то вона є алгоритмічно-обчислювана.

Теза не доводиться. А еквівалентність класів формалізмів – доведено.

# 1.2.8. Машина Поста.

- Машина Поста має нескінченну інформаційну стрічку, на яку записують інформацію в двійковому алфавіті  $\{0,1\}$ . Алгоритм задають як скінченну впорядковану послідовність пронумерованих правил — команд Поста.
- Є шість типів команд Поста:
  - 1) відмітити активну комірку стрічки, тобто записати в неї 1 і перейти до виконання  $i$ -ї команди;
  - 2) стерти відмітку активної комірки, тобто записати в неї 0 і перейти до виконання  $i$ -ї команди;
  - 3) змістити активну комірку на одну позицію вправо і перейти до виконання  $i$ -ї команди;
  - 4) змістити активну комірку на одну позицію вліво і перейти до виконання  $i$ -ї команди;
  - 5) якщо активна комірка відмічена, то перейти до виконання  $i$ -ї команди, інакше - до виконання  $j$ -ї команди;
  - 6) зупинка, закінчення роботи алгоритму.

# 1.2.9. Нормальні алгоритми Маркова.

Для формалізації поняття алгоритму А. Марков 1954 р. розробив систему нормальних алгоритмів. Алгоритми Маркова — це формальна математична система. Вони були основою для першої мови обробки рядів COMIT. Крім того, є подібність між моделлю Маркова і мовою СНОБОЛ, яка з'явилась після COMIT.

**Простою продукцією** (формулою підстановки) називають запис вигляду:

$$u \longrightarrow w$$

, де  $u, w$  - рядки в алфавіті  $V$ . У цьому разі  $V$  не містить символів ' $\longrightarrow$ ' та ' $\cdot$ '. Величину  $u$  називають ангицедентом, а  $w$  - консеквентом.

Вважають, що формула  $u \longrightarrow w$  може бути застосована до рядка  $Z \in F$ , якщо є хоча б одне входження  $u$  в  $Z$ . В іншому випадку ця формула не застосовна до рядка  $Z$ . Якщо формула може бути застосована, то канонічне (перше ліворуч) входження  $u$  в  $Z$  замінюється на  $w$ . Наприклад, якщо формулу ' $ba$ '  $\longrightarrow$  ' $c$ ' застосувати до вхідного рядка ' $ababab$ ', то в підсумку отримаємо рядок ' $acbab$ '. Проте формула ' $baa$ '  $\longrightarrow$  ' $c$ ' до рядка ' $ababab$ ' не може бути застосована.

**Заключною продукцією** (заключною підстановкою) називають запис вигляду:

$$u \longrightarrow -w$$

Упорядковану множину продукцій  $P_1, P_2, \dots, P_n$  називають нормальним алгоритмом, чи алгоритмом Маркова.

# 1.2.9. Нормальні алгоритми Маркова.

## П р и к л а д.

Нехай над словами з алфавіту  $V = \{a, b, c\}$  задано алгоритм з такими формулами підстановки

$P1 : 'ab' \rightarrow 'b'$ ,

$P2 : 'ac' \rightarrow 'c'$ ,

$P3 : 'aa' \rightarrow 'a'$ .

Цей алгоритм вилучає всі входження символу 'a' з рядка, за винятком випадку, коли 'a' є в кінці рядка.

Простежимо роботу алгоритму, якщо вхідний рядок має вигляд 'bacaaba'.  
Нехай символ  $\Rightarrow$  означає результат перетворення, а підрядок, який підлягає заміні, будемо підкреслювати:

$'baca\underline{a}baa' \Rightarrow 'bac\underline{a}baa' \Rightarrow 'ba\underline{c}baa' \Rightarrow 'bcba\underline{a}' \Rightarrow 'bcba'$ .

$P1 \qquad \qquad P1 \qquad \qquad P2 \qquad \qquad P3$

Оскільки далі жодна з формул не може бути застосована, то на цьому робота алгоритму завершується.





# Алгоритми та методи обчислень

*Лекція №3*

**Тема №1. Вступ до теорії  
алгоритмів.**

**1.3. Формальні граматики та  
формальні мови.**

Говорять, що скінченний автомат  $M = (S, I, f, s_0, F)$  **допускає** (**приймає**) ланцюжок  $\alpha$ , якщо він переводить початковий стан  $s_0$  в заключний стан; це означає, що стан  $f(s_0, \alpha)$  – елемент множини  $F$ .

Мова  $L(M)$ , яку **розпізнає автомат**  $M$ , – це множина всіх ланцюжків, які допускає автомат  $M$ . Два автомати називають **еквівалентними**, якщо вони розпізнають одну й ту саму мову.

Якщо визначити **слово** як стрічку символів, що створена через конкатенацію (з'єднання), а **мову** як множину слів(може бути нескінченною множиною), то говорять, що мова  $L$  читається (приймається) автоматом  $M$ , якщо по закінченню обробки він переходить в один з кінцевих станів  $F$ :

$$L = \{w \in \Sigma^* \mid \hat{\delta}(S_0, w) \in F\}$$

## Формальні мови. Засоби опису формальних мов

- Формальна граматика або просто граматика в теорії формальних мов — спосіб опису формальної мови, тобто виділення деякої підмножини з множини всіх слів деякого скінченного алфавіту. Розрізняють породжувальні і аналітичні граматики — перші ставлять правила, за допомогою яких можна побудувати будь-яке слово мови, а другі дозволяють по даному слову визначити, входить воно в мову чи ні. Формальні граматики були введені американським вченим, математиком та філософом, Н. Хомським у 50-тих роках ХХ сторіччя.
- **Алфавіт** — скінченна множина символів.  $\varepsilon$  — порожній ланцюжок. Нехай  $T$  — алфавіт, тоді:
  - $T^+$  — множина усіх можливих послідовностей, що складені з елементів цього алфавіту крім порожньої послідовності  $\varepsilon$ .
  - $T^*$  — множина усіх можливих послідовностей, що складені з елементів цього алфавіту, будь-якої довжини.
  - $T^k$  — множина усіх можливих послідовностей, що складені з елементів цього алфавіту, довжини не більше  $k$ .
- **Мова** в алфавіті  $T$  це множина ланцюжків скінченної довжини в цьому алфавіті. Зрозуміло, що кожна мова в алфавіті  $T$  є підмножиною множини  $T^*$ .

# Формальна граматика

Формальна граматика - це четвірка  $G = \{N, T, P, S\}$ . Де:

- $T$  — алфавіт термінальних символів, терміналів (від англ. terminate - завершитись). Термінальні символи є алфавітом мови.
- $N$  — алфавіт нетермінальних символів, нетерміналів.  $T \cap N = \emptyset$ ; Нетермінали не входять в алфавіт мови.
- $S$  — аксіома, спеціально виділений нетермінальний символ з якого починається опис граматики.

$$S \in N$$

- $P$  — скінченна підмножина множини  $(T \cup N)^+ \times (T \cup N)^*$ . Інколи визначають так:  $\alpha \in (T \cup N)^* \times N \times (T \cup N)^*, \beta \in (T \cup N)^*$ .

Елемент  $(\alpha, \beta)$  з множини  $P$  називається *правилом виводу* і записується у вигляді  $\alpha \rightarrow \beta$ .

Таким чином, ліва частина правила не може бути порожньою. Правила з однаковою лівою частиною записують:

$\alpha \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n, \beta_i, i = 1, 2, \dots, n$  - називаються альтернативами правила виводу з ланцюжка  $\alpha$ .

# Формальні мови та породжуючі граматики

**Формальні мови** породжена граматикою  $G$  - це множина термінальних рядків, що виводяться з аксіоми, тобто множина усіх правильних рядків.

$$L(G) = \{\alpha \in T^* \mid S \Rightarrow^* \alpha\}$$

З іншого боку, множина термінальних рядків, таких, що вони можуть бути виведені в граматиці  $G$ , називається мовою, що може бути розпізнана в даній граматиці, або допускається даною граматикою.

- Граматика  $G_1$  та  $G_2$  називаються **еквівалентними**, якщо  $L(G_1) = L(G_2)$ .
- Граматика  $G_1$  та  $G_2$  називаються **майже еквівалентними**, якщо  $L(G_1) \cup \{\varepsilon\} = L(G_2) \cup \{\varepsilon\}$ , тобто мови, ними породжувані, відрізняються не більш ніж на  $\varepsilon$ .

Граматика може бути **породжуючою** та **розпізнавальною**, це залежить від "напрямку" застосування правил. Для породжуючих граматик виведення починається з аксіоми і закінчується термінальним рядком (рядком, що складається тільки з терміналів). А розпізнавальні аналізують вхідний термінальний рядок на правильність, чи можна такий рядок вивести в цій граматиці. Коротко кажучи - породжуючі це "згори вниз", а розпізнавальні - "знизу вгору". Остання задача є практичнішою і гострою.

# Ієрархія Чомскі

	Граматика	Правила	Мови
<b>Тип-0</b>	Довільна формальна граматика	$\begin{matrix} \alpha \xrightarrow{\beta} \gamma \\ \alpha \in V^*, \beta \in V^*, \gamma \in V^*, \end{matrix}$	рекурсивно зліченна
Автомат	Машина Тюринга		
Абр.	KSV*		
<b>Тип-1</b>	Контекстно-залежна граматика	$\begin{matrix} \alpha A \beta \rightarrow \alpha' \gamma \beta \\ A \in N, \alpha, \beta \in V^*, \gamma \in V^+, \\ S \xrightarrow{\epsilon} \epsilon \text{ дозволене, коли є серед правил } P \\ \text{відсутнє } \alpha \rightarrow \beta S \gamma. \end{matrix}$	контекстно-залежна
Автомати	Лінійний обмежений автомат		
Абр.	КЗ		
<b>Тип-2</b>	Контекстно-вільна граматика	$\begin{matrix} A \xrightarrow{\gamma} \gamma \\ A \in N, \gamma \in V^* \end{matrix}$	контекстно-вільна
Автомати	недетермінований автомат з магазинною пам'яттю		
Абр.	КВ		
<b>Тип-3</b>	регулярна граматика	$\begin{matrix} S \xrightarrow{\epsilon} aB \text{ (праволінійна)} \\ \text{або } A \xrightarrow{\epsilon} Ba \text{ (ліволінійна)} \\ A \xrightarrow{\epsilon} a \\ A, B \in N, a \in \Sigma \end{matrix}$	регулярна
Автомати	Скінченний автомат (детермінований і недетермінований)		
Абр.	A		
Позначення множин – $\Sigma, N, V=\Sigma \cup N$ Операції – $\cup, \cap, \setminus, \subseteq, \supseteq, \subset, \supset$			

# Ієрархія Чомскі

	Граматика	Правила	Мови
Тип-0	Довільна формальна граматики	$\alpha \rightarrow \beta$ $\alpha \in V^* N V^*$ , $\beta \in V^*$	рекурсивно зліченна
Автомат	Машина Тюринга		
Абр.	KSV*		
Тип-1	Контекстно-залежна граматики	$\alpha A \beta \rightarrow \alpha' \gamma' \beta$ $A \in N, \alpha, \beta \in V^*, \gamma \in V^+$ $S \rightarrow \epsilon$ дозволено, коли серед правил $P$ відсутнє $\alpha \rightarrow \beta S \gamma$	контекстно-залежна
Автомати	Лінійний обмежений автомат		
Абр.	КЗ		
Тип-2	Контекстно-вільна граматики	$A \rightarrow \gamma$ $A \in N$ , $\gamma \in V^*$	контекстно-вільна
		$S \rightarrow \epsilon$ $A \rightarrow aB$ (праволінійна) $\text{або } A \rightarrow Ba$ (ліволінійна) $A \rightarrow a$ $A \rightarrow \epsilon$ $A, B \in N, a \in \Sigma$	регулярна

## Позначення множин

$\Sigma$ : множина термінальних символів

$N$ : множина нетермінальних символів

$V = \Sigma \cup N$ : словник граматики (множина всіх термінальних та нетермінальних символів)

## Операції

$C$  = Доповнення множин

$K$  = Конкатенація формальних мов

$S$  = Перетин множин

$V$  = Об'єднання множин

$*$  = Зірочка Кліні



## Довільна формальна граматика та рекурсивно зліченні мови

- Це найзагальніший клас граматик. На правила не накладаються ніяких обмежень. Вони еквівалентні Машині Тюринга. Доведено існування мов, які породжуються граматиками типу 0, але не граматиками типу 1, але невідомі прості приклади таких мов.

# Контекстно-залежна граматика та породжувані ними мови

- Нескорочуючі граматики. Всі правила мають вигляд

$$\alpha \rightarrow \beta$$

$$\alpha \in (T \cup N)^+, \beta \in (T \cup N)^*$$

$$|\alpha| \leq |\beta|$$

де  $|\alpha|$  означає кількість символів у рядку  $\alpha$ .

- Контекстно-залежні граматики. Всі правила мають вигляд:

$$\alpha \rightarrow \beta$$

$$\alpha = \xi_1 A \xi_2, \beta = \xi_1 \gamma \xi_2 : A \in N, \gamma \in (T \cup N)^+, \xi_1, \xi_2 \in (T \cup N)^*$$

- Контекстно-залежні граматики породжують контекстно-залежні мови. Тобто, кожна КЗ граматика породжує КЗ мову, і для кожної КЗ мови існує КЗ граматика, що її породжує.
- **Контекстно-залежні мови можна розпізнати недетермінованою лінійно-обмеженою машиною Тюринга (недетермінована машина Тюринга зі стрічкою обмеженої довжини).**
- Визначення приналежності слова мові ( $x \in L$ ) для КЗ-мов розв'язується.

# Контекстно-вільні граматики та породжувані ними мови

Всі правила мають вигляд

$$\alpha \rightarrow \beta$$

$$\alpha \in N, \beta \in (T \cup N)^*$$

- В усіх правилах зліва стоїть тільки один нетермінал, тобто на використання правила для даного нетерміналу не впливають символи, що оточують його. Ці символи називають **контекстом**.

# Контекстно-вільна граматики та породжувальні ними мови

Контекстно-вільну мову  $L$ , породжену КВ граматикою  $G$  позначають  $L(G)$ , де:

$$L(G) = \{w | w \in T^*, S \rightarrow_G^* w\}$$

Символом  $\rightarrow_G^*$  позначають послідовність правил виводу граматики  $G$ , в результаті застосування якої отримують слово  $w$  мови  $L(G)$ . Також  $L(G) \subset T^*$ .

- Контекстно-вільні мови можна розпізнати недетермінованим автоматом з магазинною пам'ятю. За умови існування детермінованого автомату, здатного розпізнати мову, її називають детермінованою КВ мовою. Ця підмножина КВ мов утворює теоретичну основу для синтаксиса багатьох мов програмування.

Контекстно-вільні мови можуть містити порожнє слово, наприклад, через правило виводу  $(S \rightarrow \varepsilon)$ .

# Регулярні граматики. А-граматики

- Граматиками типу 3 можна визначити як праволінійну або ліволінійну граматику, або як змішану. Також ці мови називають скінченно-автоматними, бо вони еквівалентні скінченним автоматам, тобто **клас мов, що породжуються граматиками типу 3, збігається з класом мов, які розпізнаються скінченими автоматами**. Також ці граматики є основою регулярних виразів. Доведено, що праволінійні та ліволінійні граматики еквівалентні, і існує алгоритм для переведення правил граматики одного типу в інший тип.

- Праволінійна граматика. Всі правила мають вигляд:

$$\alpha \rightarrow \beta$$

$$\alpha \rightarrow \omega\beta, \beta \in N, \omega \in T^*$$

або

$$\alpha \rightarrow \omega, \omega \in T^*$$

- Ліволінійна граматика. Всі правила мають вигляд:

$$\alpha \rightarrow \beta$$

$$\alpha \rightarrow \beta\omega, \beta \in N, \omega \in T^*$$

або

$$\alpha \rightarrow \omega, \omega \in T^*$$

# Регулярні вирази

- Окрім автоматів і граматики для опису регулярних мов зручно використовувати регулярні вирази.
- **Регулярний вираз є послідовністю, що описує множину рядків. Ці послідовності використовують для точного описання множини без перелічення всіх її елементів.** Наприклад, множина, що складається із слів «грати» та «ґрати» може бути описана регулярним виразом «[гґ]рати». В більшості формалізмів, якщо існує регулярний вираз, що описує задану множину, тоді існує нескінченна кількість варіантів, які описують цю множину

# Метасимволи регулярних виразів

- **.** (крапка) – один довільний символ (крім символу переходу на новий рядок)
- **\t** – табуляція
- **\n** – новий рядок
- **\\** – власне \ (backslash)
- **\s** – один довільний пробільний символ (пробіл, табуляція, новий рядок)
- **\S** – один довільний символ, який не входить у перелічені для \s
- **\d** – одна довільна цифра (digit)
- **\D** – один довільний символ, який не може бути цифрою

# Метасимволи регулярних виразів

- **\w** – одна довільна літера або цифра або знак підкреслювання (word character), те саме що [A-Za-z0-9\_]
- **\W** – один довільний символ, який не входить у перелічені для \w
- **-** – позначає діапазон(у класі символів) або власне цей символ, якщо це перший символ у класі ([-abc])
- **\$** – кінець рядка
- **^** – початок рядка або заперечення, якщо це перший символ у класі символів ([^abc])
- **^\$** – пустий рядок
- **|** - логічне АБО (використовується у групі символів)



# Метасимволи регулярних виразів

- `\<` – початок слова
- `\>` – кінець слова
- `\b` – межа слова (початок або кінець) або символ `backspace`, якщо він знаходиться у класі символів
- `\B` – позиція, що не є межею слова
- `\.` – власне крапка
- `\$` – власне символ `$`
- `\[, \]` – власне квадратні дужки
- `\(, \)` – власне круглі дужки
- `\{, \}` – власне фігурні дужки

# Метасимволи регулярних виразів

## *Групування та повторення:*

- **\*** – нуль або більше разів повторений попередній символ (або група символів)
- **+** – один або більше разів повторений попередній символ (або група символів)
- **?** – нуль або один раз повторений попередній символ (або група символів)
- **()** – групують символи (всі, що присутні у дужках, можливе застосування логічного АБО (символ |))
- **[]** – визначають клас (або множину) символів – неупорядковану групу символів, з якої для відповідного регулярного виразу обирається один довільний
- **{n}** – рівно n разів повторений попередній символ (або група символів)
- **{n, m}** – попередній символ (або група символів) повторений від n до m разів
- **{n, }** – n або більше разів повторений попередній символ (або група символів)

# Регулярні вирази

## Приклади

- **[A-Za-z]** – один довільний символ латинської абетки, незалежно від регістру (тут не можна писати [A-z], бо між літерами у великому та малому регістрах знаходяться інші символи)
- **[0-9]** або **(0|1|2|3|4|5|6|7|8|9)** або **\d** – одна довільна цифра
- **\(\d{3}\)\ \d{3}-\d{4}** – номер телефону у форматі (044) 123-4567
- **#[0-9a-fA-F]{6}** – шістнадцятковий код кольору (наприклад, #12CCAA)
- **([0-9]|[1-9][0-9]|1[1-9][0-9]|2[0-4][0-9]|25[0-5])** – довільне число з діапазону 0-255 (цей вираз можна скоротити)

## Домашнє завдання №3

З використанням `std::regex` розробити програму, яка шукає в тексті лексеми заданого формату.

### Вибір варіанту

$$(N_{\text{ж}} + N_{\text{г}} + 1) \% 30 + 1$$

де:  $N_{\text{ж}}$  – порядковий номер студента в групі, а  $N_{\text{г}}$  – номер групи (1,2,3,4,5,6,7,8 або 9)

№	Формат лексем
1	Up-Low2, перший символ Up
2	Low-Up2, перший символ Low
3	Up2
4	Low2
5	Up-Low4, перший символ Up
6	Low-Up4, перший символ Low
7	Up4
8	Low4
9	Up-Low6, перший символ Up
10	Low-Up6, перший символ Low
11	Up6
12	Low6
13	Up-Low8, перший символ Up
14	Low-Up8, перший символ Low
15	Up8
16	Low8
17	Up-Low2, перший символ _
18	Low-Up2, перший символ _
19	Up4, перший символ _
20	Low4, перший символ _
21	Up-Low4, перший символ _
22	Low-Up4, перший символ _
23	Up6, перший символ _
24	Low6, перший символ _
25	Up-Low6, перший символ _
26	Low-Up6, перший символ _
27	Up8, перший символ _
28	Low8, перший символ _
29	Up-Low8, перший символ _
30	Low-Up8, перший символ _

# Приклад коду

*Лістинг*

```
#include <fstream>
#include <iostream>
#include <algorithm>
#include <iterator>
#include <regex>

int main()
{
    std::string text =
        "Sir, in my heart there was a kind of fighting "
        "That would not let me sleep. Methought I lay "
        "Worse than the mutines in the bilboes. Rashly- "
        "And prais'd be rashness for it-let us know "
        "Our indiscretion sometimes serves us well ... "
        ; // - Hamlet, Act 5, Scene 2, 4-8

    std::regex token_re("\\b[s][a-z']+\\b", std::regex::icase);
    std::copy(std::sregex_token_iterator(text.begin(), text.end(), token_re, 0),
        std::sregex_token_iterator(),
        std::ostream_iterator<std::string>(std::cout, "\\n"));

    return 0;
}
```

# Нотація Бекуса-Наура

- Нотація Бе́куса—Нау́ра (англ. Backus-Naur form, BNF) — це спосіб запису правил контекстно-вільної граматики, тобто форма опису формальної мови.
- **БНФ визначає скінченну кількість символів (нетерміналів). Крім того, вона визначає правила заміни символу на якусь послідовність букв (терміналів) і символів.** Процес отримання ланцюжка букв можна визначити поетапно: спочатку є один символ (символи зазвичай знаходяться у кутових дужках, а їх назва не несе жодної інформації). Потім цей символ замінюється на деяку послідовність букв і символів, відповідно до одного з правил. Потім процес повторюється (на кожному кроці один із символів замінюється на послідовність, згідно з правилом). Зрештою, виходить ланцюжок, що складається з букв і не містить символів. Це означає, що отриманий ланцюжок може бути виведений з початкового символу.

# Нотація Бекуса-Наура

Нотація БНФ є набором «продукцій», кожна з яких відповідає зразку:

**<символ> ::= <вираз, що містить символи>**

**<вираз, що містить символи>** це послідовність символів або послідовності символів, розділених вертикальною рисою |, що повністю перелічують можливий вибір символ з лівої частини формули.

Наступні чотири символи є символами мета-мови, вони не визначені у мові, котру описують:

- < — лівий обмежувач виразу
- > — правий обмежувач виразу
- ::= — визначене як
- | — або

Інші символи належать до «абетки» описуваної мови.



# Нотація Бекуса-Наура

*Приклад 1.* БНФ для поштової адреси:

```
<поштова-адреса> ::= <поштове-відділення> <вулична-адреса> <особа>  
<поштове-відділення> ::= <індекс> ", " <місце> <EOL>  
<місце> ::= <село> | <місто>  
<вулична-адреса> ::= <вулиця> ", " <будинок> <EOL>  
<особа> ::= <прізвище> <ім'я> <EOL> | <прізвище> <ім'я> <по батькові> <EOL>
```

*Приклад 2.* Один зі способів означення натуральних чисел за допомогою БНФ:

```
<нуль> ::= 0  
<ненульова цифра> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
<цифра> ::= <нуль> | <ненульова цифра>  
<послідовність цифр> ::= <нуль> | <ненульова цифра> | <цифра> <послідовність цифр>  
<натуральне число> ::= <цифра> | <ненульова цифра> <послідовність цифр>
```

# Розширена нотація Бекуса-Наура

- Розширена форма Бекуса — Наура (англ. extended Backus–Naur form, EBNF) була розроблена Ніклавсом Віртом, яка сьогодні існує в багатьох варіантах, перед усім — ISO-14977. РБНФ відрізняється від БНФ більш «ємкими» конструкціями, що дозволяють при тій же виразності спростити і скоротити в обсязі опис.

# Розширена нотація Бекуса-Наура

При використанні розширеної форми Бекуса-Наура (EBNF):

- не термінальні символи записуються як окремі слова
- термінальні символи записуються в лапках
- вертикальна риска |, як і в БНФ, використовується для визначення альтернатив
- круглі дужки використовуються для групування
- квадратні дужки використовуються для визначення можливого входження символу або групи символів
- фігурні дужки використовуються для визначення можливого повторення символу або групи символів
- символ рівності використовується замість символу  $:: =$
- конкатенація позначається комою
- символ крапки використовується для позначення кінця правила
- коментарі записуються між символами (\* ... \*)

# Розширена нотація Бекуса-Наура

*Приклад.* Один зі способів означення цілих чисел за допомогою РБНФ:

---

Integer = Sign UnsignedInteger.

UnsignedInteger = digit, {digit}.

Sign = [ "+" | "-" ].

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

---

\* для компіляції коду рекомендується використати <http://cpp.sh>

## Домашнє завдання №4

Розробити програму, яка за допомогою `boost::spirit` реалізовує заданий відповідно до варіанту синтаксис для запису виразу.

### Вибір варіанту

$$(N_{\text{ж}} + N_{\text{г}} + 6) \% 10 + 1$$

де:  $N_{\text{ж}}$  – порядковий номер студента в групі, а  $N_{\text{г}}$  – номер групи (1,2,3,4,5,6,7,8 або 9)

## Варіанти завдання

[illegible]

[illegible]

# Приклад коду

Наведена програма (лістинг 2) перевіряє коректність виразу, синтаксис якого заданий РБНФ (лістинг 1):

Лістинг 1

```
<expression> = <term_a>, { ("+" | "-"), <term_a> }*.  
<term_a>      = <term_m>, { ("*" | "/"), <term_m> }*.  
<term_m>      = <value> | "+" <term_m> | "-" <term_m> | "(", <expression>, ")".
```

Лістинг 2

```
#include <iostream>  
#include <string>  
#include <boost/spirit/include/qi.hpp>  
  
namespace qi = boost::spirit::qi;  
  
template <typename Iterator>  
struct calculator_simple : qi::grammar<Iterator>{  
    calculator_simple() : calculator_simple::base_type(expression){  
        expression = term    >> *( '+' >> term | '-' >> term );  
        term        = factor >> *( '*' >> factor | '/' >> factor );  
  
        factor      =  
            qi::uint_  
            | '(' >> expression >> ')'   
            | '+' >> factor  
            | '-' >> factor;  
    }  
  
    qi::rule<Iterator> expression, term, factor;  
};
```



```
int main(){
    std::cout << "Welcome to the expression parser!\n\n";
    std::cout << "Type an expression or [q or Q] to quit\n\n";

    typedef std::string      str_t;
    typedef str_t::iterator str_t_it;

    str_t expression;

    calculator_simple<str_t_it> calc;

    while(true){
        std::getline(std::cin, expression);
        if(expression == "q" || expression == "Q") break;
        str_t_it begin = expression.begin(), end = expression.end();

        bool success = qi::parse(begin, end, calc);

        std::cout << "-----\n";
        if(success && begin == end)
            std::cout << "Parsing succeeded\n";
        else
            std::cout << "Parsing failed\nstopped at: \""
                << str_t(begin, end) << "\"\n";
        std::cout << "-----\n";
    }
}
```