

Laboratorio di Linguaggi Formali e Traduttori

LFT lab T4, a.a. 2020/2021

Docente: Luigi Di Caro

Analisi
lessicale

Analisi Lessicale

Prima però...

- ▶ Abbiamo visto un modo di implementare DFA
 - ▶ variabili state per gli stati dell'automa,
 - ▶ e ciclo per simulare il comportamento dell'automa (transizioni)
- ▶ Ora possiamo pensare di essere più flessibili

Implementazione di un DFA

- ▶ Prime due lezioni: presentato un approccio per implementare un DFA utilizzando un singolo ciclo `while`, rappresentando lo stato attuale del DFA con la variabile `state` e le transizioni tramite comandi condizionali (`switch` e `if`).
 - ▶ Vantaggio: facile capire la corrispondenza tra DFA e codice (quindi più facile ottenere un'implementazione *corretta* nel caso di un DFA *complesso*).
 - ▶ Svantaggio: codice inutilmente complesso per esempi di DFA *molto semplici*.
- ▶ Per riassumere: per automi complessi, va bene. Per situazioni semplici, no.
 - ▶ Semplice: con due stati, o più stati ma con transizioni simili (ad es. con »a« torno indietro, ecc.

Implementazione di un DFA

- ▶ Approccio alternativo per DFA semplici: evitare l'utilizzo della variabile `state` per rappresentare lo stato del DFA
 - ▶ e utilizzare cicli e comandi condizionali per rappresentare il DFA.
- ▶ Corrispondenza tra «**posizioni nel codice**» e lo **stato** del DFA.
 - ▶ Rimane la corrispondenza DFA - codice!
 - ▶ Non sarà presentato un metodologia per questo approccio, ma nelle slide successive verranno presentati due esempi che confrontano i due approcci, indicando con frecce la corrispondenza tra stato del DFA e posizioni corrispondenti nel codice.

Primo esempio

- ▶ Approccio alternativo per DFA semplici

```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

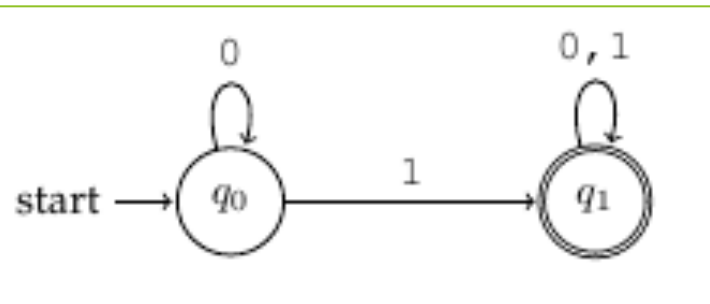
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

non usa la variabile state!



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

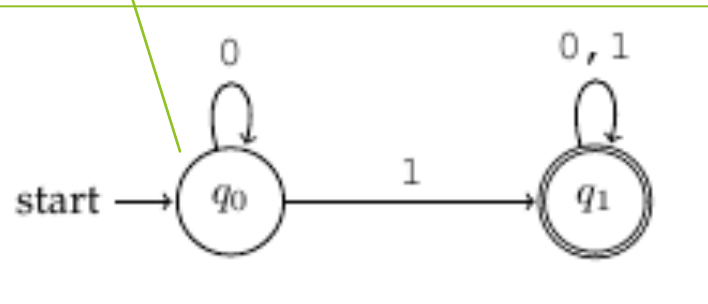
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

non usa la variabile state!



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

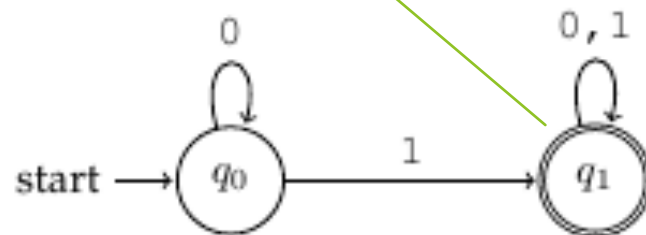
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

non usa la variabile state!




```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

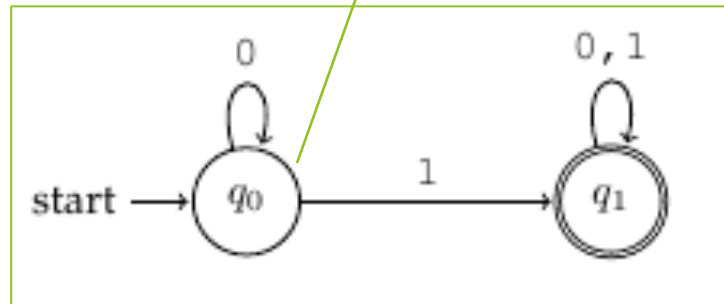
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

non usa la variabile state!



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

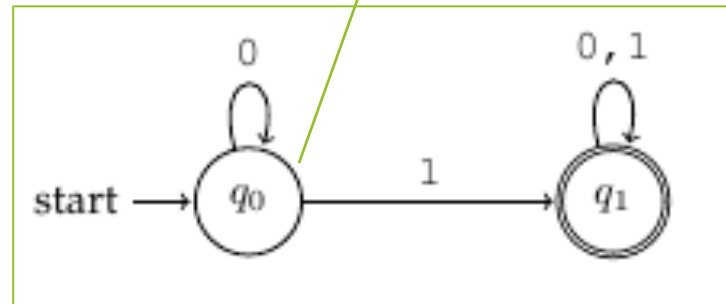
```

public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;
    while (i < s.length()) {
        char ch = s.charAt(i++);
        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

come prima

non usa la variabile state!



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

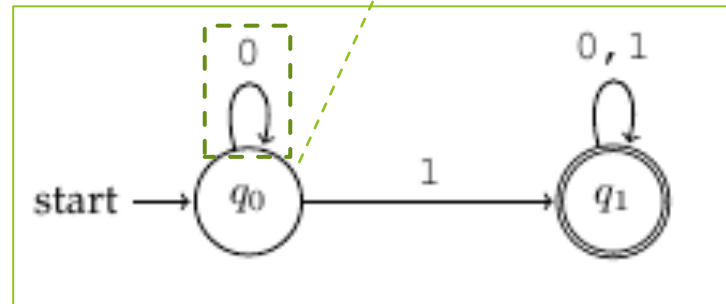
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

inizialmente, se leggiamo 0, non facciamo nulla



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

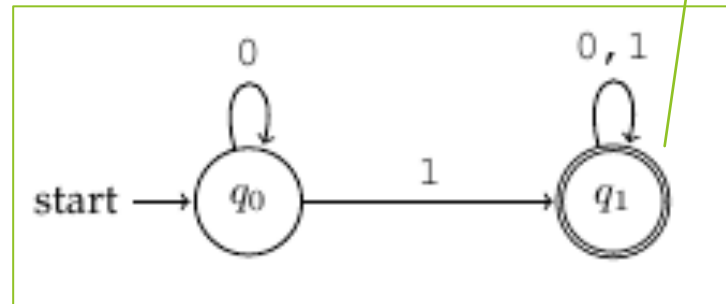
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

se leggiamo «1»,
entriamo in un
nuovo while



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

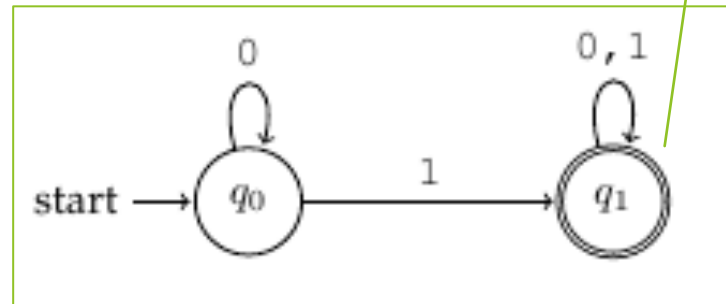
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

se leggiamo «0»
oppure «1» non
facciamo nulla



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

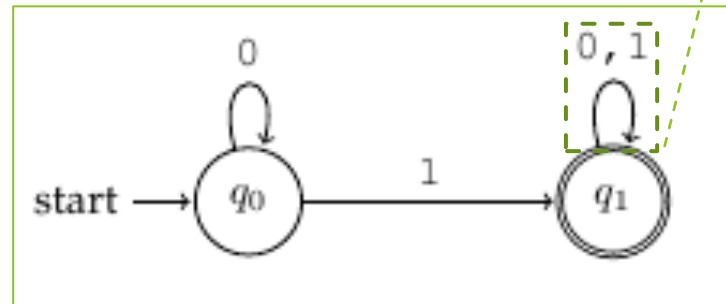
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

restituiamo vero,
se finiamo di
leggere l'input



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

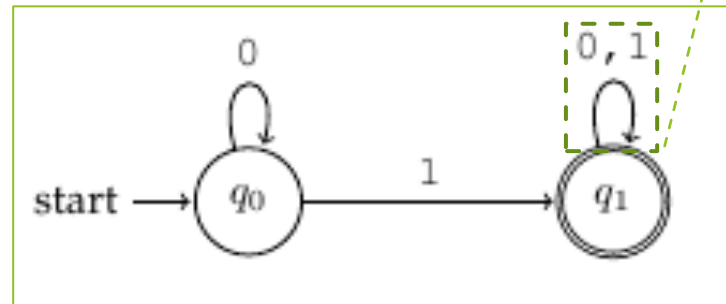
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

restituiamo falso,
se finiamo di
leggere l'input nel
primo while



```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

```

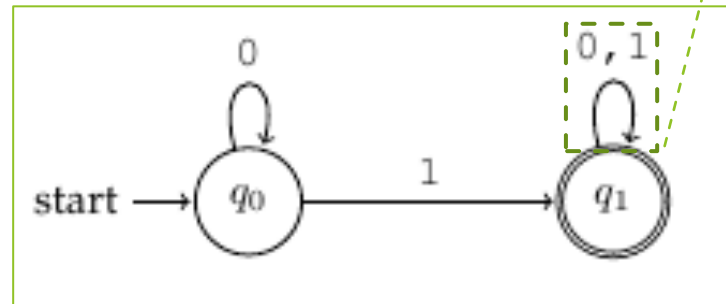
public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```

input non
valido, ad es.
«2», «b», ecc.




```

public static boolean scan_at_least_one_1_with_state(String s)
{
    int state = 0;
    int i = 0;

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == '0')
                    state = 0;
                else if (ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == '0' || ch == '1')
                    state = 1;
                else
                    state = -1;
                break;
        }
    }
    return state == 1;
}

```

Domanda senza
premi: perchè
«final» solo a
sinistra?

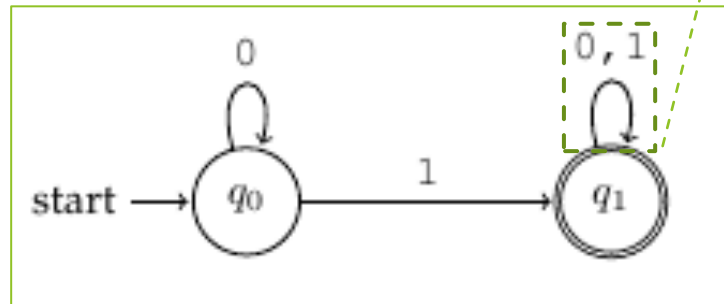
```

public static boolean scan_at_least_one_1_without_state(String s)
{
    int i = 0;

    while (i < s.length()) {
        char ch = s.charAt(i++);

        if (ch == '0')
            // do nothing
            ;
        else if (ch == '1') {
            while (i < s.length()) {
                ch = s.charAt(i++);
                if (ch == '0' || ch == '1')
                    // do nothing
                    ;
                else
                    // invalid input
                    return false;
            }
            return true;
        }
        else
            // invalid input
            return false;
    }
    return false;
}

```



Secondo esempio

- ▶ Approccio alternativo per DFA semplici

```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```

```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}

```

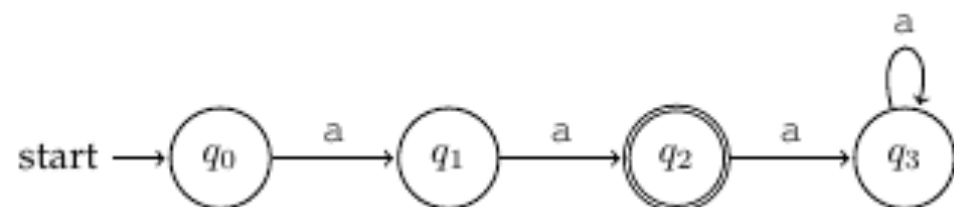
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

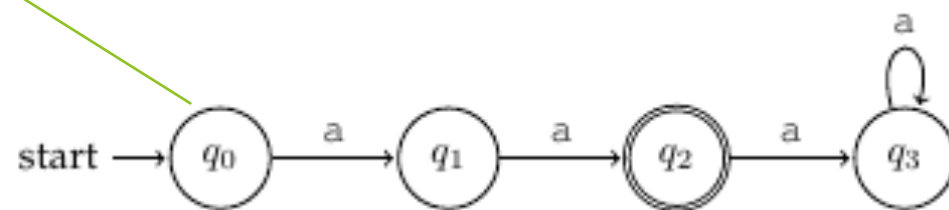
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

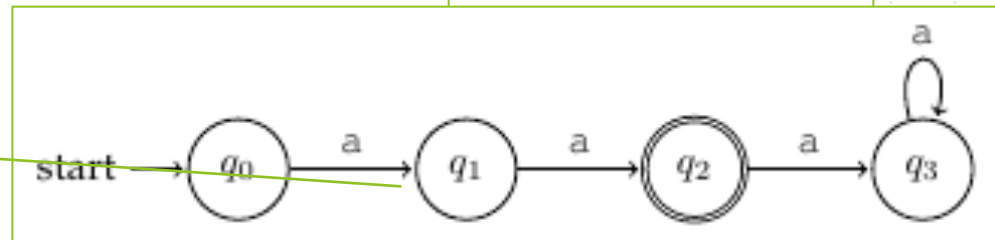
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

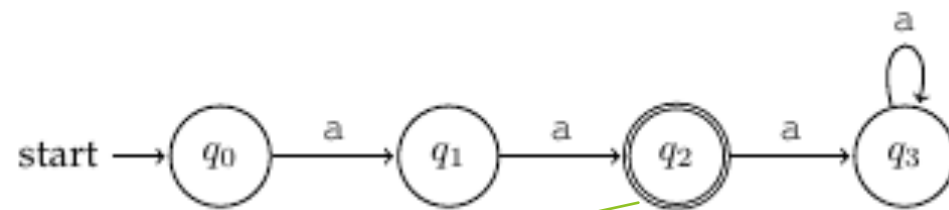
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

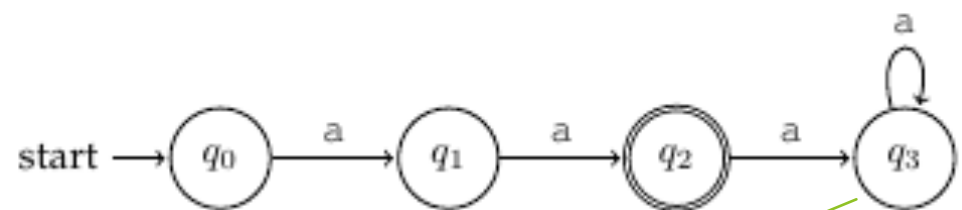
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```



```

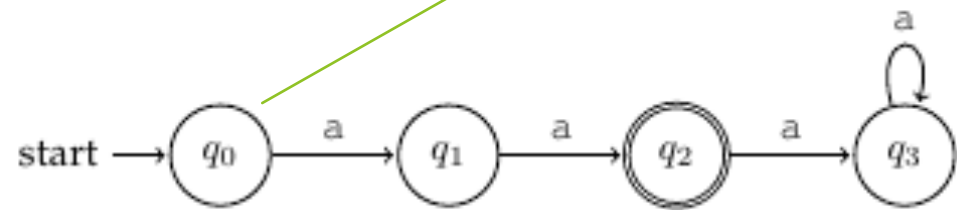
public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```

non più while
ma if. Questa
volta q0 non è
coinvolto in un
ciclo.



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

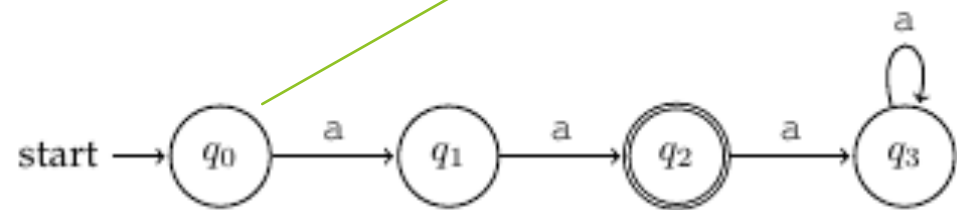
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

altrimenti
restituiamo
falso subito

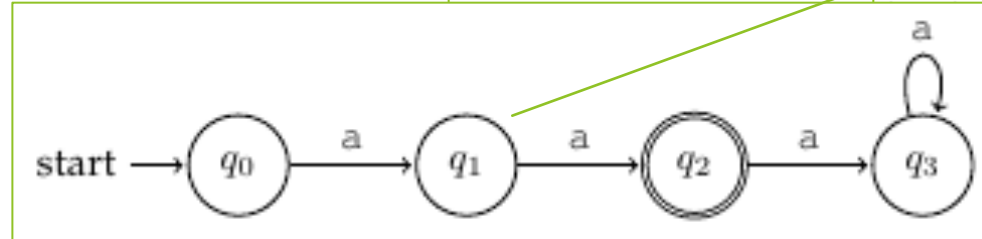
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



se leggiamo un'altra
«a»:

- se finiamo di
leggere, restituiamo
vero
- ...

```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}

```

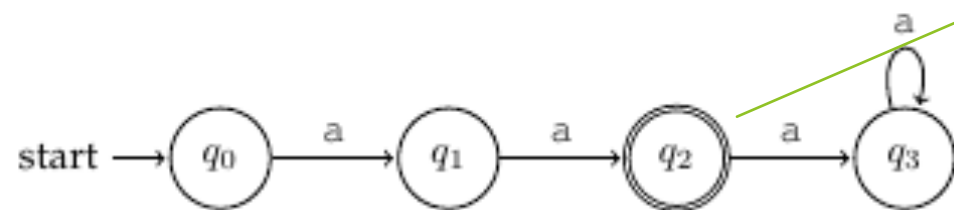
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

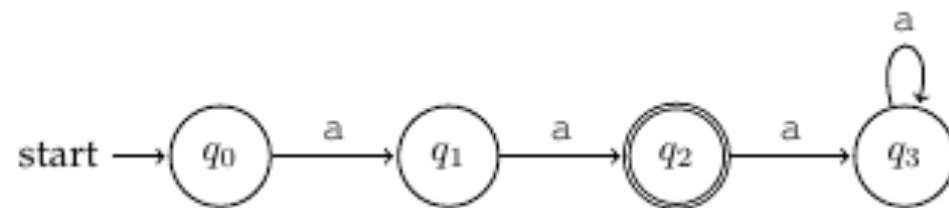
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                        ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

usiamo il
while, ma non
è obbligatorio

solo per
esplicitare
meglio
relazione con
automa

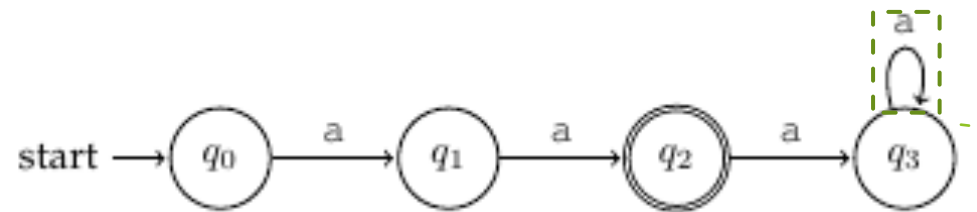
```

public static boolean scan_exactly_two_a_with_state(String s)
{
    int state = 0;
    int i = 0;

    System.out.println(s);

    while (state >= 0 && i < s.length()) {
        final char ch = s.charAt(i++);
        switch (state) {
            case 0:
                if (ch == 'a')
                    state = 1;
                else
                    state = -1;
                break;
            case 1:
                if (ch == 'a')
                    state = 2;
                else
                    state = -1;
                break;
            case 2:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
            case 3:
                if (ch == 'a')
                    state = 3;
                else
                    state = -1;
                break;
        }
    }
    return state == 2;
}

```



```

public static boolean scan_exactly_two_a_without_state(String s)
{
    int i = 0;
    System.out.println(s);

    char ch = s.charAt(i++);

    if (ch == 'a') {
        if (i == s.length())
            return false;
        else {
            ch = s.charAt(i++);
            if (ch == 'a') {
                if (i == s.length())
                    return true;
                else {
                    while (i < s.length()) {
                        ch = s.charAt(i++);
                        if (ch == 'a')
                            // do nothing
                            ;
                        else
                            // invalid input
                            return false;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

Considerazione

- ▶ Spesso, si implementa il lexer (fase successiva) senza pensare a questa prima parte su automi
- ▶ Spesso, in questo modo si sbaglia! **Attenzione**
- ▶ **Non dimenticate questa parte di automi nel momento in cui implementerete le prossime parti.**

Analisi lessicale

- ▶ Analisi lessicale:
 - ▶ Input: un programma scritto in un linguaggio di programmazione.
 - ▶ Operazione: *raggruppare* sequenze di caratteri dell'input in elementi atomici del linguaggio (ad esempio, parole chiave, costanti numeriche, identificatori, operatori aritmetici, operatori logici, operatori di confronto, parentesi, ecc.).
 - ▶ Output: una sequenza di *token*; ogni token corrisponde ad un elemento atomico del linguaggio.



Analisi lessicale

- ▶ Analisi lessicale:
 - ▶ Input: un programma scritto in un linguaggio di programmazione.
 - ▶ Operazione: *raggruppare* sequenze di caratteri dell'input in elementi atomici del linguaggio (ad esempio, parole chiave, costanti numeriche, identificatori, operatori aritmetici, operatori logici, operatori di confronto, parentesi, ecc.).
 - ▶ Output: una sequenza di *token*; ogni token corrisponde ad un elemento atomico del linguaggio.



ad es. «print(a)»

Analisi lessicale

- ▶ Analisi lessicale:
 - ▶ Input: un programma scritto in un linguaggio di programmazione.
 - ▶ Operazione: *raggruppare* sequenze di caratteri dell'input in elementi atomici del linguaggio (ad esempio, parole chiave, costanti numeriche, identificatori, operatori aritmetici, operatori logici, operatori di confronto, parentesi, ecc.).
 - ▶ Output: una sequenza di *token*; ogni token corrisponde ad un elemento atomico del linguaggio.



ad es. «print(a)»
sequenza di simboli

Analisi lessicale

- ▶ Analisi lessicale:
 - ▶ Input: un programma scritto in un linguaggio di programmazione.
 - ▶ Operazione: *raggruppare* sequenze di caratteri dell'input in elementi atomici del linguaggio (ad esempio, parole chiave, costanti numeriche, identificatori, operatori aritmetici, operatori logici, operatori di confronto, parentesi, ecc.).
 - ▶ Output: una sequenza di *token*; ogni token corrisponde ad un elemento atomico del linguaggio.



ad es. «print(a)»
print è una parola chiave -> elemento atomico

Analisi lessicale

- ▶ Analisi lessicale:
 - ▶ Input: un programma scritto in un linguaggio di programmazione.
 - ▶ Operazione: *raggruppare* sequenze di caratteri dell'input in elementi atomici del linguaggio (ad esempio, parole chiave, costanti numeriche, identificatori, operatori aritmetici, operatori logici, operatori di confronto, parentesi, ecc.).
 - ▶ Output: una sequenza di *token*; ogni token corrisponde ad un elemento atomico del linguaggio.

< ? > ↔ token

Programma
come sequenza
di caratteri

print(a)

Analizzatore
lessicale
(lexer)

Programma
come sequenza
di token

<266, print> <40> <257, a> <41> <-1>

266 è il nome del token, con attributo associato la stringa print

Analisi lessicale

- ▶ Esempi di elementi atomici di un linguaggio «Java-like»:
 - ▶ Parole chiave (`while`, `print`)
 - ▶ Identificatori (`i`, `f`)
 - ▶ Operatori (`:=`, `<=`, `*`, `+`)
 - ▶ Costanti (`2`, `1`)
 - ▶ Simboli di punteggiatura (`;`, `(`, `)`, `{`, `}`)
 - ▶ ...

```
i := 2;  
f := 1;  
while (i <= n) {  
    f := f * i;  
    i := i + 1  
};  
print(f)
```

Analisi lessicale: terminologia

- ▶ Unità lessicale: elemento atomico del linguaggio dell'input.
 - ▶ Esempi: la parola chiave `while`, un identificatore, un costante, ecc.
- ▶ *Token*: è un elemento che consiste di un nome, oppure una coppia che consiste di un nome e un attributo.
 - ▶ *Nome del token*: «simbolo astratto» che rappresenta un'unità lessicale.
- ▶ *Pattern*: la descrizione della forma che le sequenze di caratteri di un'unità lessicale possono avere.
 - ▶ Esempio: un identificatore è descritta da una sequenza di lettere e cifre numeriche, dove la sequenza non inizia con una cifra numerica.
- ▶ *Lessema*: sequenza di caratteri del programma sorgente che rispetta il pattern del token.

Analisi lessicale: terminologia

- ▶ Unità lessicale: elemento atomico del linguaggio dell'input.
 - ▶ Esempi: la parola chiave `while`, un identificatore, un costante, ecc.
- ▶ *Token*: è un elemento che consiste di un nome, oppure una coppia che consiste di un nome e un attributo.
 - ▶ *Nome del token*: «simbolo astratto» che rappresenta un'unità lessicale.
- ▶ *Pattern*: la descrizione della forma che le sequenze di caratteri di un'unità lessicale possono avere.
 - ▶ Esempio: un identificatore è descritta da una sequenza di lettere e cifre numeriche, dove la sequenza non inizia con una cifra numerica.
- ▶ *Lessema*: sequenza di caratteri del programma sorgente che rispetta il pattern del token.

ad es. per «print» il
pattern è proprio la
sequenza di caratteri
`p r i n t`

Token del linguaggio

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Pattern: nella tabella i pattern sono descritti testualmente.

- In generale i pattern sono descritti tramite le espressioni regolari.

- «Costante numerica»:

`0|[1-9][0-9]*`

(non trattiamo numeri con sequenze di 0 iniziali).

- Identificatore come «Lettera seguita da lettere e cifre»:

`[a-zA-Z][a-zA-Z0-9]*`

Token del linguaggio

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Nomi:

- I nomi dei token sono espressi come costanti numeriche.
- Per i token che corrispondono ad un singolo simbolo: utilizziamo il codice ASCII del simbolo.
 - Eccezioni: < e > (perché c'è un token per gli operatori relazionali), costanti numeriche e identificatori che corrispondono a un singolo simbolo, ad esempio 8 oppure x.

Token del linguaggio

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Una nota: da 259 a 269

- non avremmo bisogno di un attributo,
- ma lo usiamo per semplificare l'implementazione (lo vedremo)

Esempi di generazione di token

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<, >, <=, >=, ==, <>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Input:

read(a)

Sequenza di token generata:

<267, read>

<40>

<257, a>

<41>

<-1>

Esempi di generazione di token

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Input:

```
while (> x 0) do print(x)
```

Sequenza di token generata:

<263, while>

<40>

<258, > >

<257, x>

<256, 0>

<41>

<264, do>

<266, print>

<40>

<257, x>

<41>

<-1>

Esempi di generazione di token

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Input:

&& 5 {cond

Sequenza di token generata:

<269>

<256, 5>

<123>

<259, cond>

<-1>

Per il momento non dobbiamo occuparci della struttura dell'input,

solo della generazione di token seguendo la descrizione della tabella.

Attributi dei token

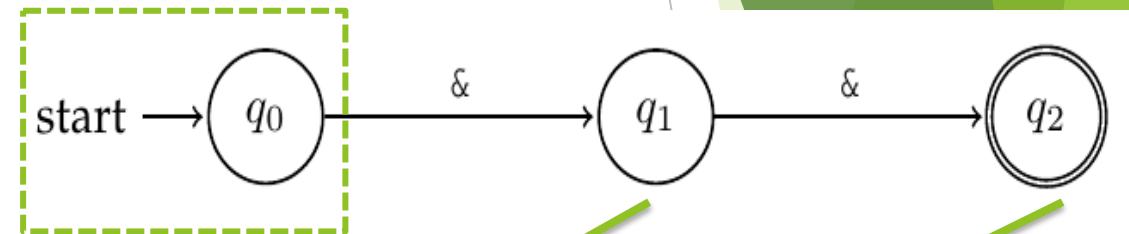
- ▶ Attributi sono importanti per i token seguenti: numeri, identificatori, operatori relazionali.
 - ▶ Esempio: per $x < 5$, genera $\langle 257, x \rangle \langle 258, < \rangle \langle 256, 5 \rangle \langle -1 \rangle$ - bisogna sapere quale identificatore (attributo “x”) è confrontato con quale costante numerica (attributo “5”), e in quale maniera (attributo “<”).
 - ▶ Questi attributi saranno **necessari** per la generazione di codice intermedio (ultimo argomento del laboratorio).
- ▶ Nell’implementazione, per comodità, anche i token che corrispondono a lessemi con più simboli (||, &&, le parole chiave) avranno attributi.

Analizzatore lessicale (lexer)

- ▶ Dato un input (programma scritto in un certo linguaggio), l'analizzatore lessicale produce una **sequenza di token** che corrisponde all'input.
- ▶ Gestisce anche «**white space**» (spazi bianchi, tabulazioni, ritorno a capo): **deve essere ignorato** (non corrisponde a nessun token).
- ▶ Deve **segnalare la presenza di caratteri illeciti** che non corrispondono ai pattern di nessun token, ad esempio # o @ per nostro linguaggio.

Utilizzo dei DFA nell'implementazione del lexer

- **Domanda:** come capire che una sotto-sequenza di simboli dell'input corrisponde ad un token?
- **Risposta:** Quando la sotto-sequenza corrisponde al pattern del token (problema di *riconoscimento* della sotto-sequenza).

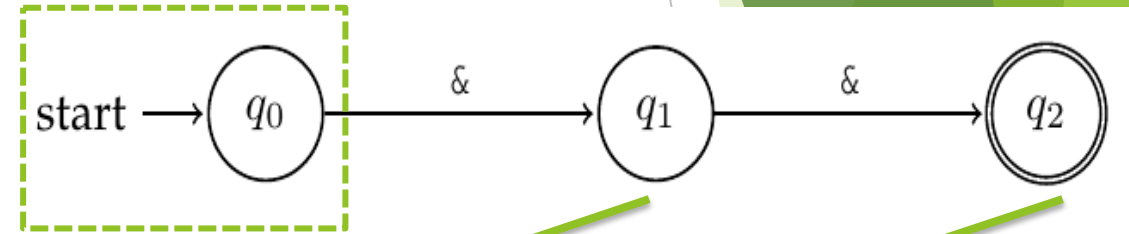


```
case '&':  
    readch(br);  
    if (peek == '&') {  
        peek = ' ';  
        return Word.and;  
    } else {  
        System.err.println("Erroneous character"  
            + " after & : " + peek );  
        return null;  
    }  
}
```

Two green arrows point from the code to the DFA diagram. The first arrow points from the character '&' in the code to the transition from q_0 to q_1 . The second arrow points from the character ' ' (space) in the code to the transition from q_1 to q_2 .

Utilizzo dei DFA nell'implementazione del lexer

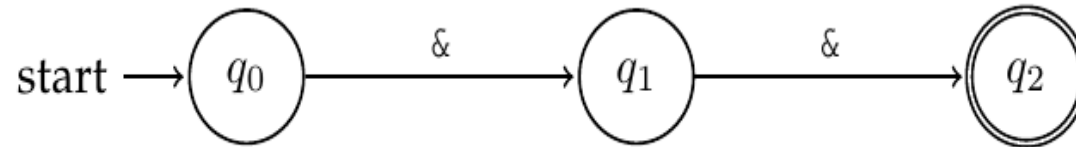
- ▶ Per ogni token, tradurre il pattern (espressione regolare) con un DFA.
 - ▶ Passi della traduzione: espressione regolare \Rightarrow ϵ -NFA \Rightarrow DFA \Rightarrow DFA minimo.
 - ▶ I pattern di nostro linguaggio sono semplici: si può progettare il DFA direttamente, senza fare la traduzione.
- ▶ Implementare il DFA come parte del lexer.
- ▶ Esempio a destra: token per &&



```
case '&':
    readch(br);
    if (peek == '&') {
        peek = ' ';
        return Word.and;
    } else {
        System.err.println("Erroneous character"
            + " after & : " + peek );
        return null;
    }
```

Utilizzo dei DFA nell'implementazione del lexer

- Implementare il DFA come parte del lexer.
 - Esempio a destra: token per &&



- DFA molto semplice!
 - come dicevamo ad inizio lezione...

```
case '&':  
    readch(br);  
    if (peek == '&') {  
        peek = ' '  
        return Word.and;  
    } else {  
        System.err.println("Erroneous character"  
            + " after & : " + peek );  
        return null;  
    }  
}
```

Utilizzo dei DFA nell'implementazione del lexer

- ▶ Ricapitolando:
 - ▶ Per ogni token
 - ▶ se complesso,
 - ▶ espressione regolare - e-NFA - DFA - DFA minimo.
 - ▶ altrimenti direttamente
 - ▶ DFA semplice
 - ▶ implementazione «alternativa»
 - ▶ DFA complesso
 - ▶ Implementazione «classica»

Riconoscimento dei token

- ▶ Dopo il riconoscimento di un token, dobbiamo identificare con cura qual è il prossimo simbolo dell'input da analizzare.
- ▶ **Esempio (1):**
 - ▶ Per l'input $x \leq 5$, dopo la lettura della sotto-sequenza \leq , siamo sicuri che quella sotto-sequenza corrisponde al token **<258, \leq >**.
 - ▶ Il prossimo simbolo, cioè 5, sarà poi letto nel contesto dell'identificazione del *prossimo* token.

$x \leq 5$

Riconoscimento dei token

- ▶ Dopo il riconoscimento di un token, dobbiamo identificare con cura qual'è il prossimo simbolo dell'input da analizzare.
- ▶ **Esempio (1):**
 - ▶ Per l'input $x \leq 5$, dopo la lettura della sotto-sequenza \leq , siamo sicuri che quella sotto-sequenza corrisponde al token $\langle 258, \leq \rangle$.
 - ▶ Il prossimo simbolo, cioè 5, sarà poi letto nel contesto dell'identificazione del *prossimo* token.

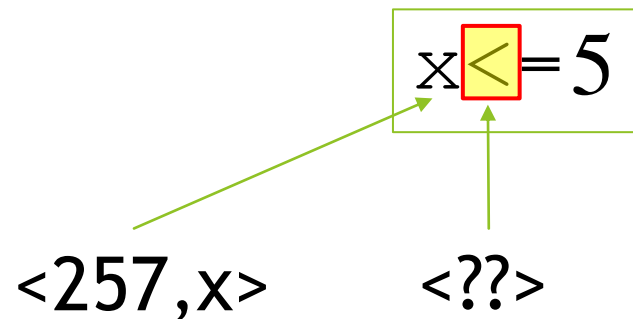
$\langle 257, x \rangle$

$x \leq 5$

Supponiamo che abbiamo già
identificato il token $\langle 257, x \rangle$

Riconoscimento dei token

- ▶ Dopo il riconoscimento di un token, dobbiamo identificare con cura qual'è il prossimo simbolo dell'input da analizzare.
- ▶ **Esempio (1):**
 - ▶ Per l'input $x \leq 5$, dopo la lettura della sotto-sequenza \leq , siamo sicuri che quella sotto-sequenza corrisponde al token $\langle 258, \leq \rangle$.
 - ▶ Il prossimo simbolo, cioè 5, sarà poi letto nel contesto dell'identificazione del *prossimo* token.

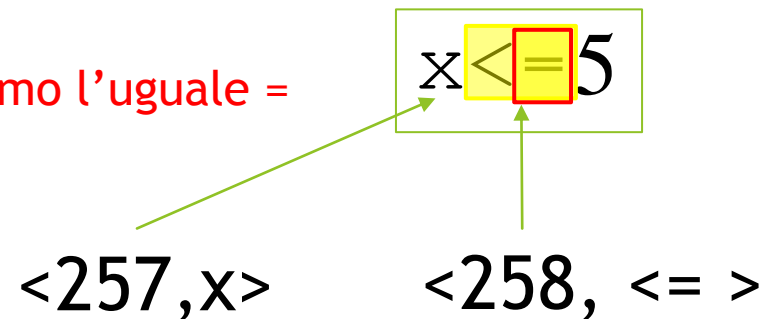


Legge il $<$ come primo passo per identificare il prossimo token

Riconoscimento dei token

- ▶ Dopo il riconoscimento di un token, dobbiamo identificare con cura qual'è il prossimo simbolo dell'input da analizzare.
- ▶ **Esempio (1):**
 - ▶ Per l'input $x \leq 5$, dopo la lettura della sotto-sequenza \leq , siamo sicuri che quella sotto-sequenza corrisponde al token $\langle 258, \leq \rangle$.
 - ▶ Il prossimo simbolo, cioè 5, sarà poi letto nel contesto dell'identificazione del *prossimo* token.

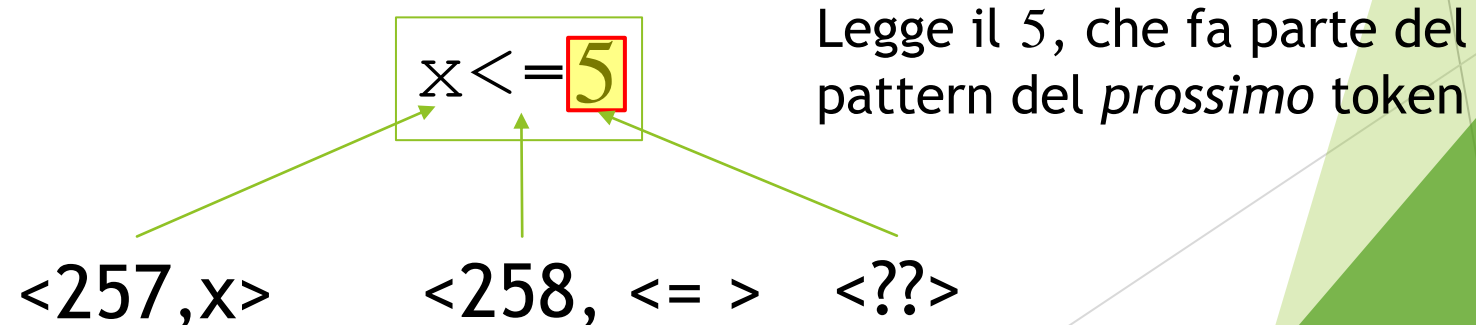
decidiamo quando leggiamo l'uguale =



Legge il =, così è sicuro che il prossimo token da generare è $\langle 258, \leq \rangle$

Riconoscimento dei token

- ▶ Dopo il riconoscimento di un token, dobbiamo identificare con cura qual'è il prossimo simbolo dell'input da analizzare.
- ▶ **Esempio (1):**
 - ▶ Per l'input $x \leq 5$, dopo la lettura della sotto-sequenza \leq , siamo sicuri che quella sotto-sequenza corrisponde al token $\langle 258, \leq \rangle$.
 - ▶ Il prossimo simbolo, cioè 5, sarà poi letto nel contesto dell'identificazione del *prossimo* token.



Riconoscimento dei token

► Esempio (2):

- Per l'input $x < 5$, dopo la lettura del $<$, non è ancora chiaro se il token corrisponde a $<258, <= >$, $<258, <> >$ oppure $<258, < >$.
- Quindi il simbolo dopo $<$, cioè 5, dev'essere letto.
- A quel punto, è chiaro che il token per l'operatore relazionale è $<258, < >$.
- Procediamo con l'identificazione del prossimo token, prendendo in considerazione il fatto che abbiamo già letto il 5 (che sarà il primo simbolo del lessema del prossimo token da generare).

$<257, x>$

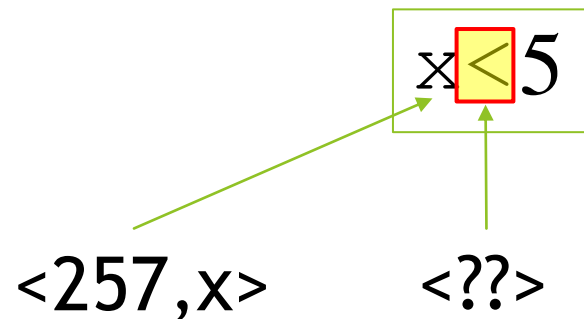
$x < 5$

Supponiamo che abbiamo già identificato il token $<257, x>$

Riconoscimento dei token

► Esempio (2):

- Per l'input $x < 5$, dopo la lettura del $<$, non è ancora chiaro se il token corrisponde a $<258, <= >$, $<258, < >$ oppure $<258, < >$.
- Quindi il simbolo dopo $<$, cioè 5, dev'essere letto.
- A quel punto, è chiaro che il token per l'operatore relazionale è $<258, < >$.
- Procediamo con l'identificazione del prossimo token, prendendo in considerazione il fatto che abbiamo già letto il 5 (che sarà il primo simbolo del lessema del prossimo token da generare).

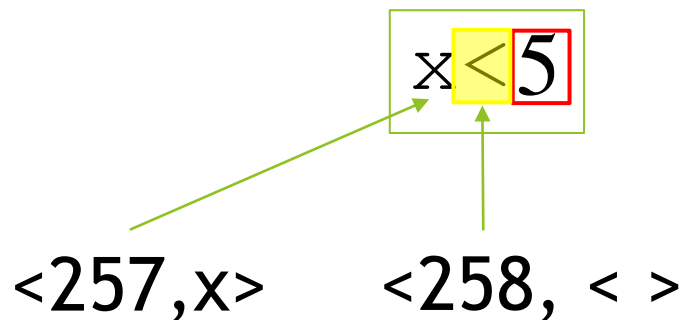


Legge il $<$ come primo passo per identificare il prossimo token

Riconoscimento dei token

► Esempio (2):

- Per l'input $x < 5$, dopo la lettura del $<$, non è ancora chiaro se il token corrisponde a $<258, <= >$, $<258, < >$ oppure $<258, < >$.
- Quindi il simbolo dopo $<$, cioè 5, dev'essere letto.
- A quel punto, è chiaro che il token per l'operatore relazionale è $<258, < >$.
- Procediamo con l'identificazione del prossimo token, prendendo in considerazione il fatto che abbiamo già letto il 5 (che sarà il primo simbolo del lessema del prossimo token da generare).



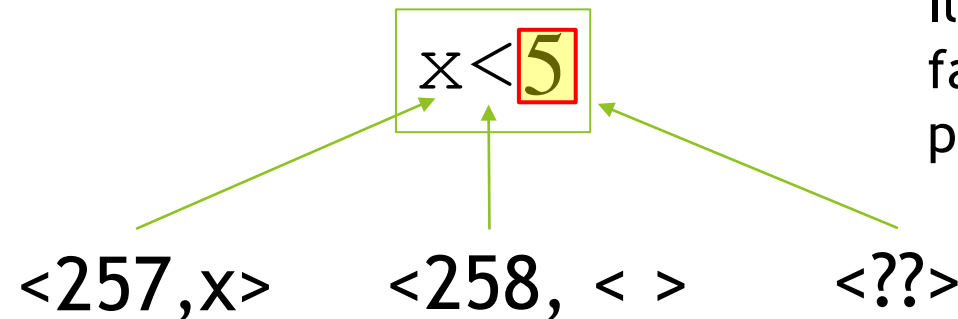
capiamo quando leggiamo il 5

**dato che in nessun pattern
possiamo avere la sequenza <5
all'inizio, è sicuro che $<$
corrisponde al token $<258, < >$**

Riconoscimento dei token

► Esempio (2):

- Per l'input $x < 5$, dopo la lettura del $<$, non è ancora chiaro se il token corrisponde a $<258, <= >$, $<258, < >$ oppure $<258, < >$.
- Quindi il simbolo dopo $<$, cioè 5, dev'essere letto.
- A quel punto, è chiaro che il token per l'operatore relazionale è $<258, < >$.
- Procediamo con l'identificazione del prossimo token, prendendo in considerazione il fatto che abbiamo già letto il 5 (che sarà il primo simbolo del lessema del prossimo token da generare).



Il 5 letto a passo precedente
fa parte del lessema del
prossimo token

Riconoscimento dei token

- ▶ Come facciamo a distinguere
 - ▶ identificatori
- ▶ da
 - ▶ operazioni come print, while, ecc.?

Riconoscimento dei token

- ▶ Come facciamo a distinguere
 - ▶ identificatori
- ▶ da
 - ▶ operazioni come print, while, ecc.?

identificatori: lettere seguite eventualmente da numeri, non considerando le parole chiave

Riconoscimento dei token

- ▶ Come facciamo a distinguere
 - ▶ identificatori
- ▶ da
 - ▶ operazioni come print, while, ecc.?

identificatori: lettere seguite eventualmente da numeri, non considerando le parole chiave

> prima si procede con pattern per identificatori, dopodiché si controlla di non aver letto una parola chiave e si assegna il token id in caso negativo

Gestione identificatori/parole chiave

- ▶ Dopo la lettura di un lessema di un identificatore: utilizzare il lessema per l'attributo del token (ad esempio, per l'identificatore `temp`, otteniamo il token `<257, temp>`).
- ▶ Parole chiave: corrispondono al pattern degli identificatori («lettera seguita da lettere e cifre», cioè `[a-zA-Z][a-zA-Z0-9]*`).
- ▶ Consiglio per l'implementazione:
 - ▶ Prima identificare una sotto-sequenza che corrisponde al pattern degli identificatori («lettera seguita da lettere e cifre»), memorizzando la sotto-sequenza come una `string`.
 - ▶ Poi confrontare la `string` con tutte le parole chiave; se la `string` non corrisponde a una delle parole chiave, è per forza un identificatore.

Andiamo sul codice!

- ▶ Dalla pagina Moodle del corso, scaricate la cartella «**Codice analisi lessicale**», che include:
 - ▶ *Tag.java*
 - ▶ *Token.java*
 - ▶ *Word.java*
 - ▶ *NumberTok.java*
 - ▶ *Lexer.java*

Classe Tag

```
public class Tag {  
    public final static int  
        EOF = -1, NUM = 256, ID = 257, RELOP = 258,  
        COND = 259, WHEN = 260, THEN = 261, ELSE = 262,  
        WHILE = 263, DO = 264, SEQ = 265, PRINT = 266, READ = 267,  
        OR = 268, AND = 269;  
}
```

All'interno del codice, avrete accesso al nome/id dei tag usando la classe Tag.

Ad es., con Tag.NUM potrò avere:

...

return Tag.NUM;

...

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Classe Token

```
public class Token {
    public final int tag;
    public Token(int t) { tag = t; }
    public String toString() {return "<" + tag + ">";}
    public static final Token
        not = new Token('!'),
        lpt = new Token('('),
        rpt = new Token(')'),
        lpg = new Token('{'),
        rpg = new Token('}'),
        plus = new Token('+'),
        minus = new Token('-'),
        mult = new Token('*'),
        div = new Token('/'),
        assign = new Token('='),
        semicolon = new Token(';');
}
```

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Classe Word

```
public class Word extends Token_I {  
    public String lexeme = "";  
    public Word(int tag, String s) { super(tag); lexeme=s; }  
    public String toString() { return "<" + tag + ", " + lexeme + ">"; }  
    public static final Word  
        cond = new Word(Tag.COND, "cond"),  
        when = new Word(Tag.WHEN, "when"),  
        then = new Word(Tag.THEN, "then"),  
        elsetok = new Word(Tag.ELSE, "else"),  
        whiletok = new Word(Tag.WHILE, "while"),  
        dotok = new Word(Tag.DO, "do"),  
        seq = new Word(Tag.SEQ, "seq"),  
        print = new Word(Tag.PRINT, "print"),  
    ...  
}
```

inseriamo anche il
lessema nella stampa

Classe NumberTok

```
public class NumberTok extends Token {  
    // ... completare ...  
}
```

suggerimenti:

- attenzione ad usare il copia-incolla
- non usare stringhe come in Word.java → un numero non è una stringa, usate un intero
- create un costruttore, un metodo *toString*, ...
- non elencate possibilità come in Word.java → per identificatori e numeri esistono infinite possibilità!

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Condizionale	cond	259
When	when	260
Then	then	261
Else	else	262
While	while	263
Do	do	264
Sequence	seq	265
Print	print	266
Read	read	267
Disgiunzione		268
Congiunzione	&&	269
Negazione	!	33
Parentesi tonda sinistra	(40
Parentesi tonda destra)	41
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Assegnamento	=	61
Punto e virgola	;	59
EOF	Fine dell'input	-1

Classe Lexer

```
import java.io.*;
import java.util.*;

public class Lexer {

    public static int line = 1;
    private char peek = ' ';

    private void readch(BufferedReader br) {
        try {
            peek = (char) br.read();
        } catch (IOException exc) {
            peek = (char) -1; // ERROR
        }
    }

    public Token lexical_scan(BufferedReader br) {
        while (peek == ' ' || peek == '\t' || peek == '\n' || peek == '\r') {
            if (peek == '\n') line++;
            readch(br);
        }
    }
}
```

...

Classe Lexer

```
switch (peek) {  
    case '!':  
        peek = ' ';  
        return Token.not;  
  
    // ... gestire i casi di (, ), {, }, +, -, *, /, ; ... //  
  
    case '&':  
        readch(br);  
        if (peek == '&') {  
            peek = ' ';  
            return Word.and;  
        } else {  
            System.err.println("Erroneous character"  
                               + " after & : " + peek );  
            return null;  
        }  
  
    // ... gestire i casi di ||, <, >, <=, >=, ==, <>, = ... //  
  
    case (char)-1:  
        return new Token(Tag.EOF);  
    ...  
}
```

questo permette di entrare nel
while della slide precedente!

questo permette di entrare nel
while della slide precedente!

Classe Lexer

```
switch (peek) {  
    case '!':  
        peek = ' ';  
        return Token.not;  
  
    // ... gestire i casi di (, ), {, }, +, -, *, /, ; ... //  
  
    case '&':  
        readch(br);  
        if (peek == '&') {  
            peek = ' ';  
            return Word.and;  
        } else {  
            System.err.println("Erroneous character"  
                               + " after & : " + peek );  
            return null;  
        }  
  
    // ... gestire i casi di ||, <, >, <=, >=, ==, <>, = ... //  
  
    case (char)-1:  
        return new Token(Tag.EOF);  
    ...  
}
```

tocca a voi!

Main!

```
public static void main(String[] args) {  
    Lexer lex = new Lexer();  
    String path = "...path..."; // il percorso del file da leggere  
    try {  
        BufferedReader br = new BufferedReader(new FileReader(path));  
        Token tok;  
        do {  
            tok = lex.lexical_scan(br);  
            System.out.println("Scan: " + tok);  
        } while (tok.tag != Tag.EOF);  
        br.close();  
    } catch (IOException e) {e.printStackTrace();}  
}
```

It's your turn now...

- Definire **NumberTok**
- Completare **Lexer**