

# Laboratorio di Linguaggi Formali e Traduttori

## LFT lab T2, a.a. 2022/2023

### Implementazione di un DFA

(riassunto e un approccio alternativo)

# Implementazione di un DFA

- Prime due lezioni: presentato un approccio per implementare un DFA utilizzando un singolo ciclo `while`, rappresentando (1) lo stato attuale del DFA con la variabile `state` e (2) le transizioni tramite comandi condizionali (`switch` e `if`).
  - Vantaggio: è facile capire la corrispondenza tra DFA e codice (quindi più facile ottenere una implementazione *corretta* nel caso di un DFA *complesso*).
  - Svantaggio: codice inutilmente complesso per esempi di DFA *molto semplici*.
- Approccio alternativo: utilizzare cicli e comandi condizionali per simulare il DFA (senza l'utilizzo della variabile `state`).
  - Fare corrispondenza tra «posizioni nel codice» e lo stato del DFA.
  - Non sarà presentato una 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.
  - Questo approccio è consigliato *solo* per i DFA *molto 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;
}

```

```

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;
}

```

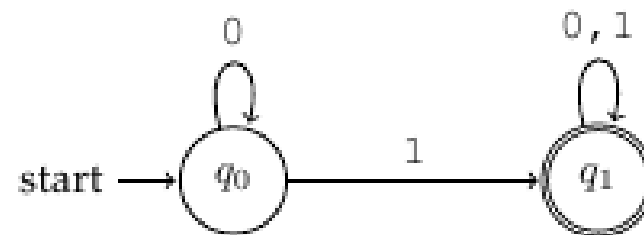
```

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;
}

```



```

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;
}

```

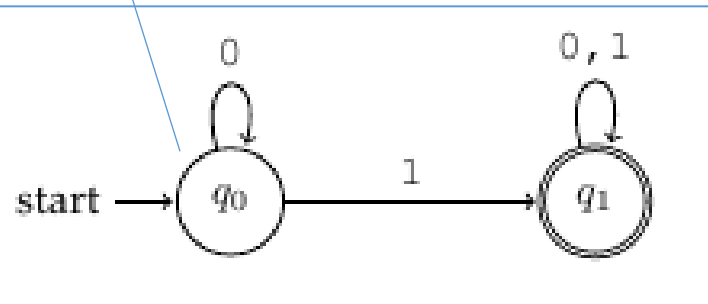
```

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;
}

```



```

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;
}

```

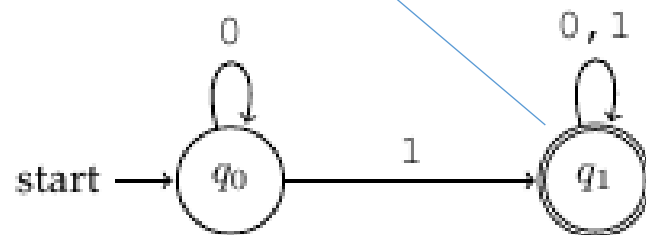
```

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;
}

```



```

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;
}

```

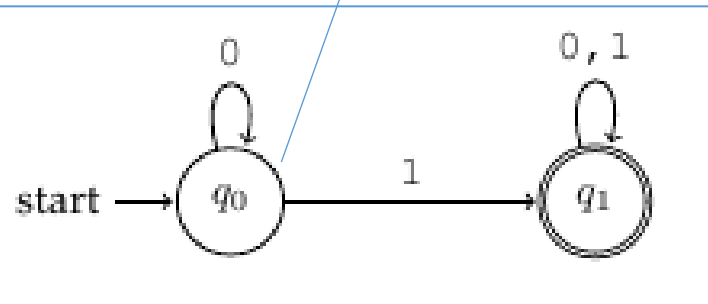
```

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;
}

```



```

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;
}

```

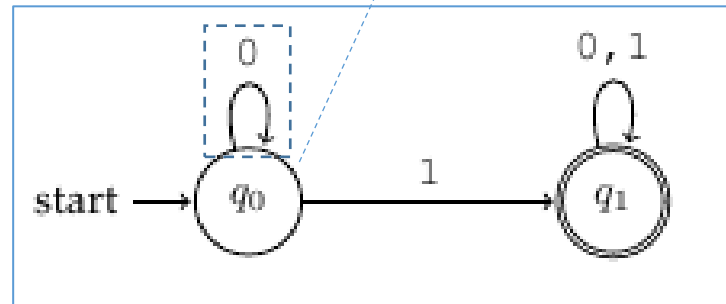
```

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;
}

```





```

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;
}

```

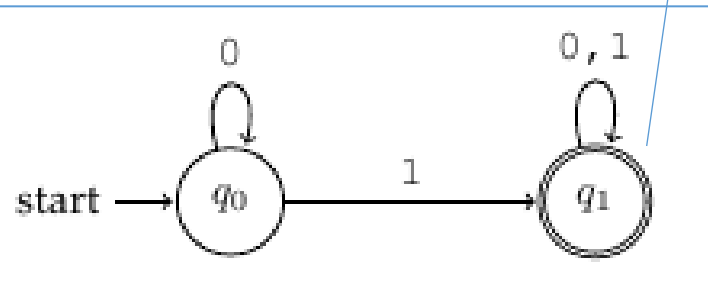
```

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;
}

```



```

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;
}

```

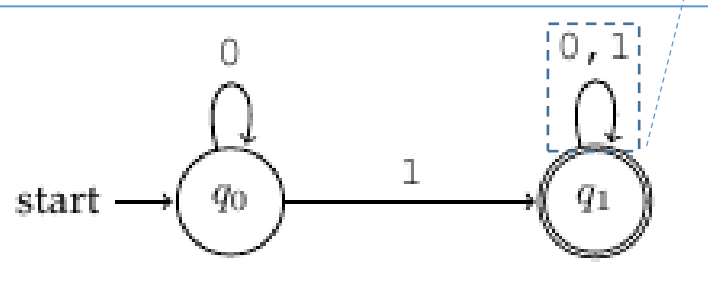
```

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;
}

```



```

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;
}

```

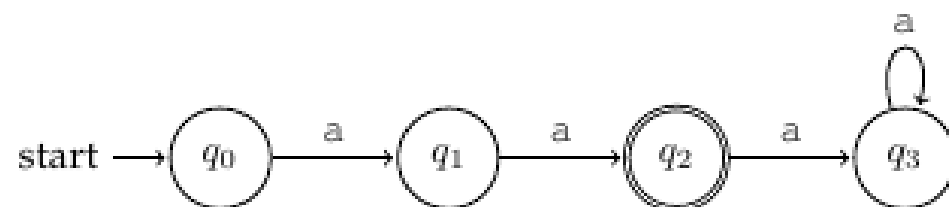
```

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;
}
}

```

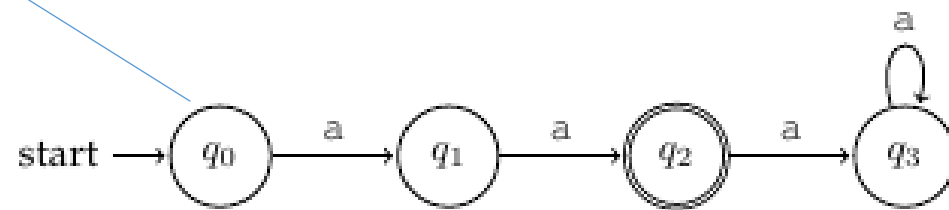
```

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;
}
}

```

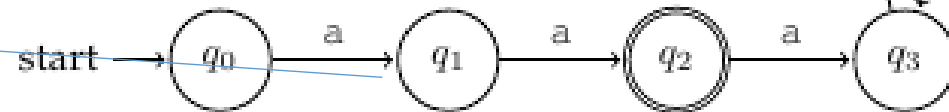
```

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;
}
}

```

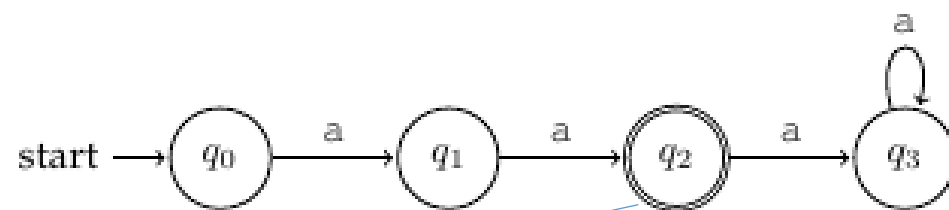
```

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;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

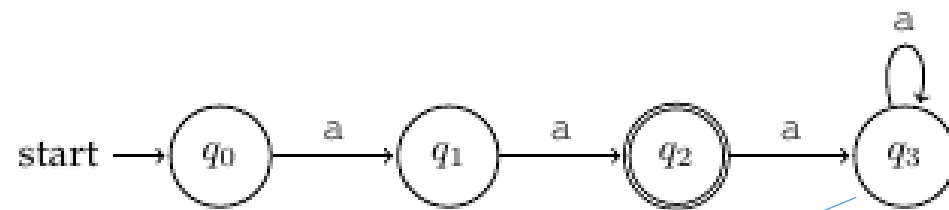
```

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;
}
}

```



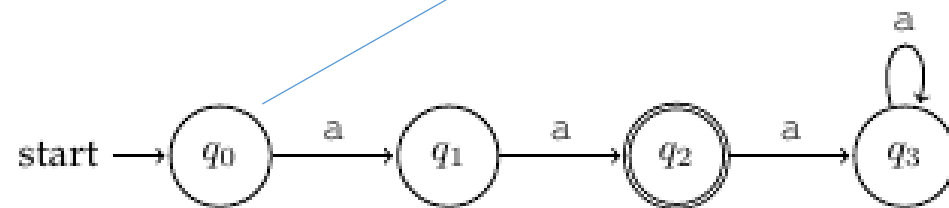
```

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;
}
}

```

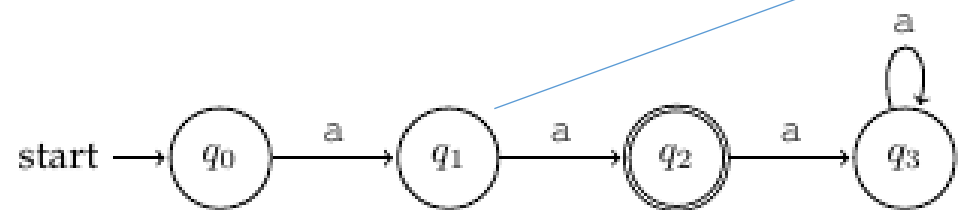
```

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;
}
}

```

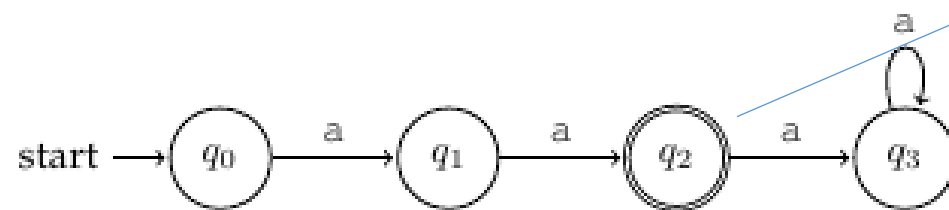
```

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;
                    }
                    return false;
                }
            }
            else
                // invalid input
                return false;
        }
    }
    else
        // invalid input
        return false;
}
}

```

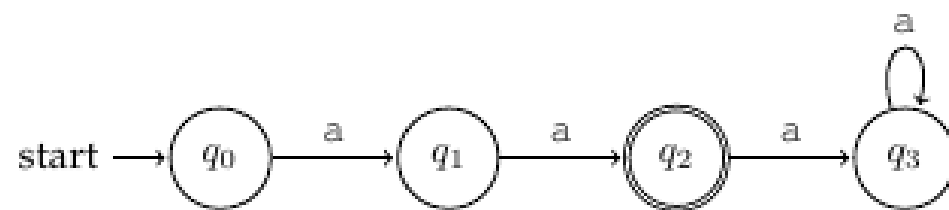
```

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;
}
}

```

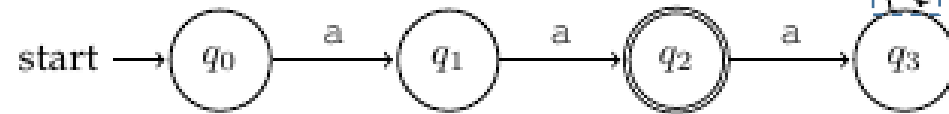
```

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;
}
}

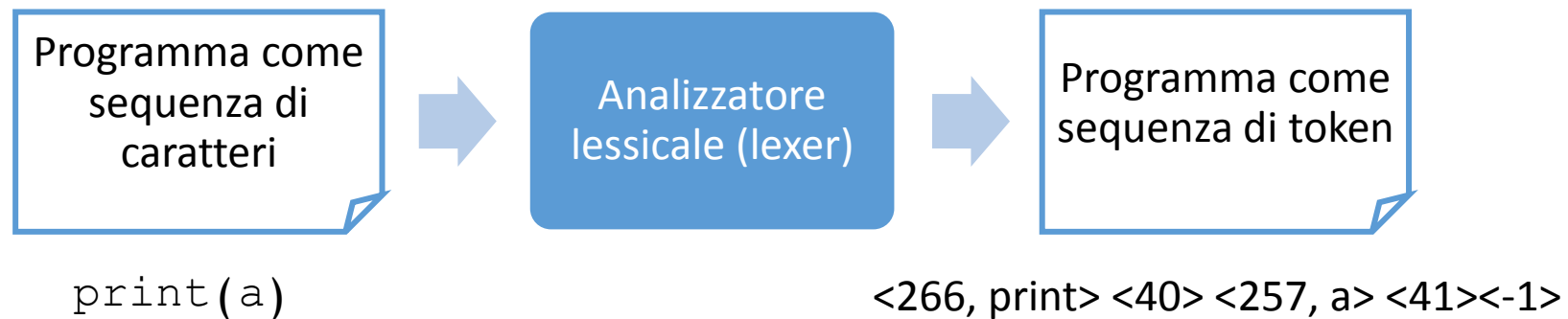
```

Laboratorio di Linguaggi Formali e Traduttori  
LFT lab T2, a.a. 2022/2023

Analisi lessicale

# 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 chiavi, 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

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

```
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 devono avere.
  - Esempio: un identificatore è descritto 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.

# Token del linguaggio

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Assegnamento	assign	259
To	to	260
Conditional	conditional	261
Option	option	262
Do	do	263
Else	else	264
While	while	265
Begin	begin	266
End	end	267
Print	print	268
Read	read	269
Disgiunzione		270
Congiunzione	&&	271
Negazione	!	33
Parentesi tonda sinistra	(	40
Parentesi tonda destra	)	41
Parentesi quadra sinistra	[	91
Parentesi quadra destra	]	93
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Punto e virgola	;	59
Virgola	,	44
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).
  - «Lettera seguita da lettere e cifre»:  
 $(a+...+z+A+...+Z)(a+...+z+A+...+Z+0+...+9)^*$

# Token del linguaggio

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Assegnamento	assign	259
To	to	260
Conditional	conditional	261
Option	option	262
Do	do	263
Else	else	264
While	while	265
Begin	begin	266
End	end	267
Print	print	268
Read	read	269
Disgiunzione		270
Congiunzione	&&	271
Negazione	!	33
Parentesi tonda sinistra	(	40
Parentesi tonda destra	)	41
Parentesi quadra sinistra	[	91
Parentesi quadra destra	]	93
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Punto e virgola	;	59
Virgola	,	44
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.

# Esempi di generazione di token

Token	Pattern	Nome
Numeri	Costante numerica	256
Identificatore	Lettera seguita da lettere e cifre	257
Relop	Operatore relazionale (<,>,<=,>=,==,<>)	258
Assegnamento	assign	259
To	to	260
Conditional	conditional	261
Option	option	262
Do	do	263
Else	else	264
While	while	265
Begin	begin	266
End	end	267
Print	print	268
Read	read	269
Disgiunzione		270
Congiunzione	&&	271
Negazione	!	33
Parentesi tonda sinistra	(	40
Parentesi tonda destra	)	41
Parentesi quadra sinistra	[	91
Parentesi quadra destra	]	93
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Punto e virgola	;	59
Virgola	,	44
EOF	Fine dell'input	-1

Input:

```
read(a)
```

Sequenza di token generata:

<269, 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
Assegnamento	assign	259
To	to	260
Conditional	conditional	261
Option	option	262
Do	do	263
Else	else	264
While	while	265
Begin	begin	266
End	end	267
Print	print	268
Read	read	269
Disgiunzione		270
Congiunzione	&&	271
Negazione	!	33
Parentesi tonda sinistra	(	40
Parentesi tonda destra	)	41
Parentesi quadra sinistra	[	91
Parentesi quadra destra	]	93
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Punto e virgola	;	59
Virgola	,	44
EOF	Fine dell'input	-1

Input:

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

Sequenza di token generata:

<265, while>

<40>

<258, > >

<257, x>

<256, 0>

<41>

<268, 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
Assegnamento	assign	259
To	to	260
Conditional	conditional	261
Option	option	262
Do	do	263
Else	else	264
While	while	265
Begin	begin	266
End	end	267
Print	print	268
Read	read	269
Disgiunzione		270
Congiunzione	&&	271
Negazione	!	33
Parentesi tonda sinistra	(	40
Parentesi tonda destra	)	41
Parentesi quadra sinistra	[	91
Parentesi quadra destra	]	93
Parentesi graffa sinistra	{	123
Parentesi graffa destra	}	125
Somma	+	43
Sottrazione	-	45
Moltiplicazione	*	42
Divisione	/	47
Punto e virgola	;	59
Virgola	,	44
EOF	Fine dell'input	-1

Input:

&& 5 {begin

Sequenza di token generata:

<269>

<256, 5>

<123>

<266, begin>

<-1>

Per il momento non dobbiamo occuparsi della struttura dell'input, solo la 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)

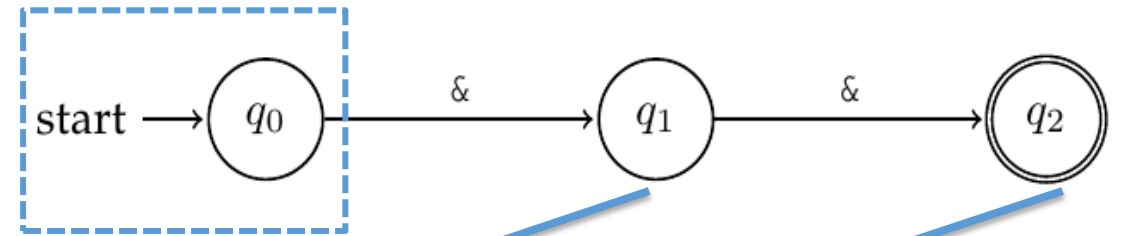
- Dato un input (programma scritto in un certo linguaggio), *l'analizzatore lessicale produce una sequenza di token che corrisponde all'input.*
- Gestisce anche «white space» (spazi bianchi, tabulazioni, ritorno a capo): deve essere ignorato (non corrisponde a nessun token).
- Deve segnalare la presenza 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 concludere 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).
- Per ogni token, passare dal pattern (espressione regolare) ad un DFA equivalente.
  - Catena di costruzioni:
    - espressione regolare
      - ⇒  $\epsilon$ -NFA
      - ⇒ DFA
      - ⇒ DFA minimo
  - I pattern di nostro linguaggio sono semplici: si può progettare il DFA direttamente, senza fare tutti i passi della catena di costruzioni.
- Implementare il DFA come parte del lexer.

Esempio: token per &&



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

# Riconoscimento dei token

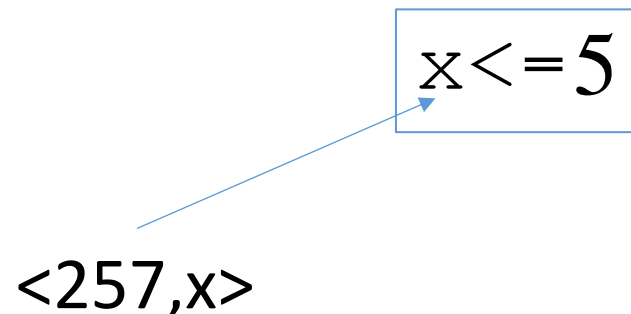
- Dopo il riconoscimento di un token, dobbiamo identificare con cura qual'è il prossimo simbolo dell'input da analizzare.
- Esempio (1):
  - Per l'input  $x \leq 5$ , dopo la lettura della sotto-sequenza  $\leq$ , siamo sicuri che quella sotto-sequenza corrisponde al token  $<258, \leq >$ .
  - Il prossimo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

$\langle 257, x \rangle$

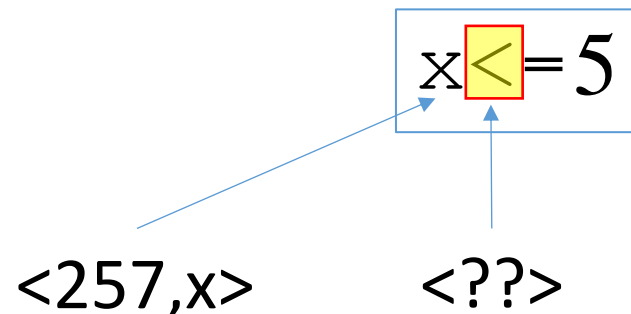


The diagram illustrates the state of a lexical analyzer. At the bottom left, the text  $\langle 257, x \rangle$  represents the current token. A blue arrow points from this token to a blue-bordered box at the top right containing the input  $x \leq 5$ . This visualizes the process of identifying the next symbol to be analyzed after the current token is recognized.

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

# Riconoscimento dei token

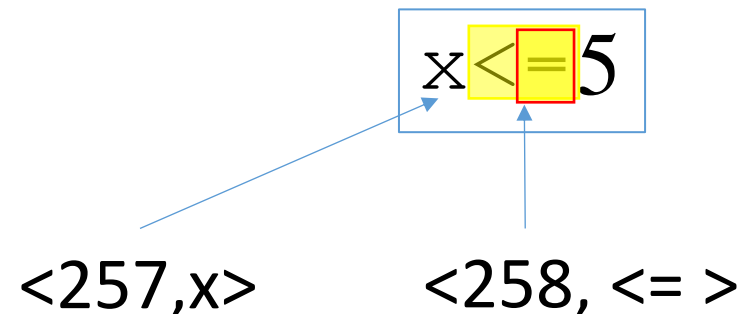
- Dopo il riconoscimento di un token, dobbiamo identificare con cura qual'è il prossimo simbolo dell'input da analizzare.
- Esempio (1):
  - Per l'input  $x \leq 5$ , dopo la lettura della sotto-sequenza  $\leq$ , siamo sicuri che quella sotto-sequenza corrisponde al token  $\langle 258, \leq \rangle$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

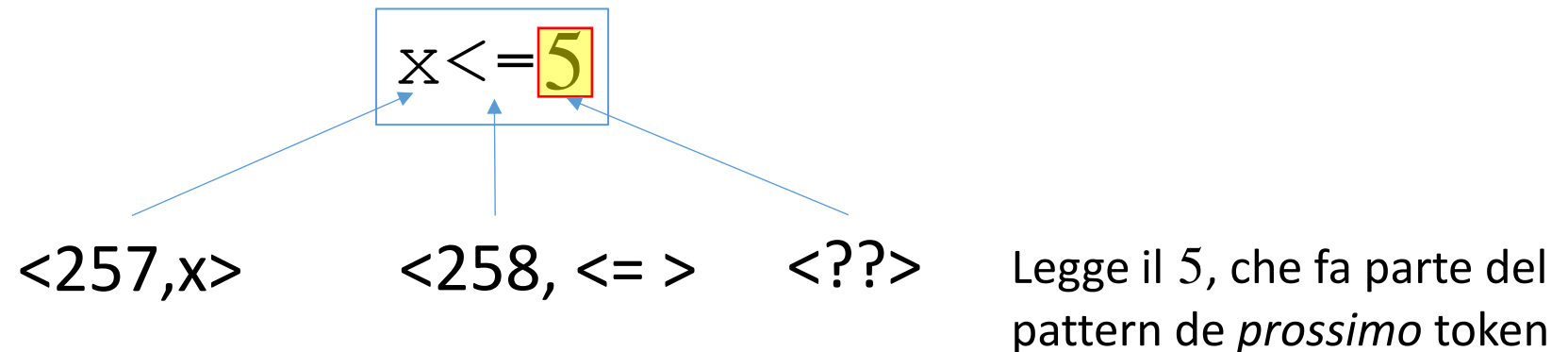
- Dopo il riconoscimento di un token, dobbiamo identificare con cura qual'è il prossimo simbolo dell'input da analizzare.
- Esempio (1):
  - Per l'input  $x \leq 5$ , dopo la lettura della sotto-sequenza  $\leq$ , siamo sicuri che quella sotto-sequenza corrisponde al token  $\langle 258, \leq \rangle$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



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

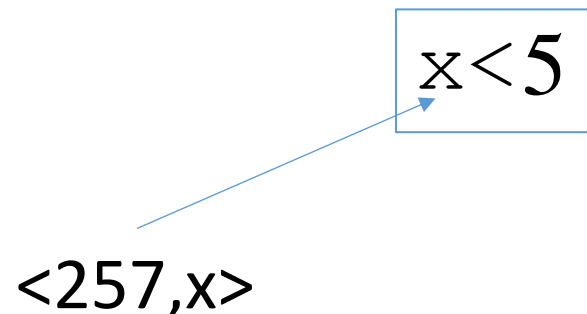
# Riconoscimento dei token

- 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

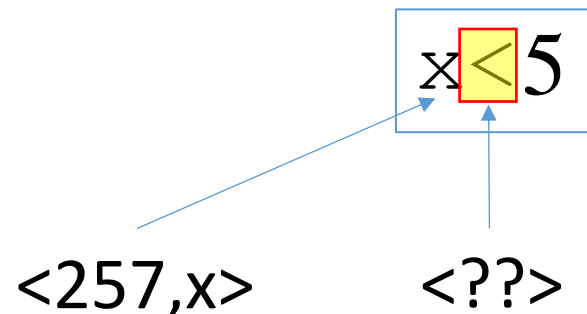
- Esempio (2):
  - Per l'input  $x < 5$ , dopo la lettura del  $<$ , non è ancora chiaro se il token corrisponde a  $<258, <= >$ ,  $<258, < >$  oppure  $<258, < >$ .
  - Quindi il simbolo dopo  $<$ , cioè 5, deve essere letto.
  - A quel punto, è chiaro che il token per l'operatore relazionale è  $<258, < >$ .
  - 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).



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, deve 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).

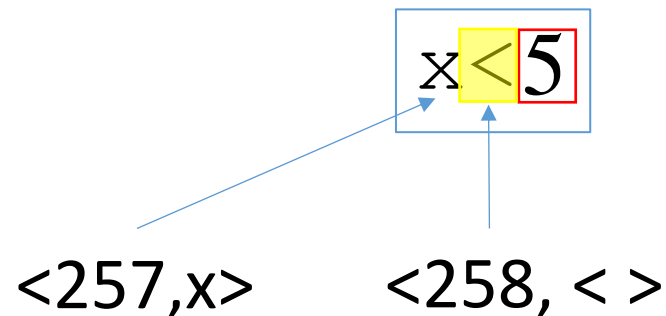


Legge il  $<$  come primo passo per identificare il prossimo token



# Riconoscimento dei token

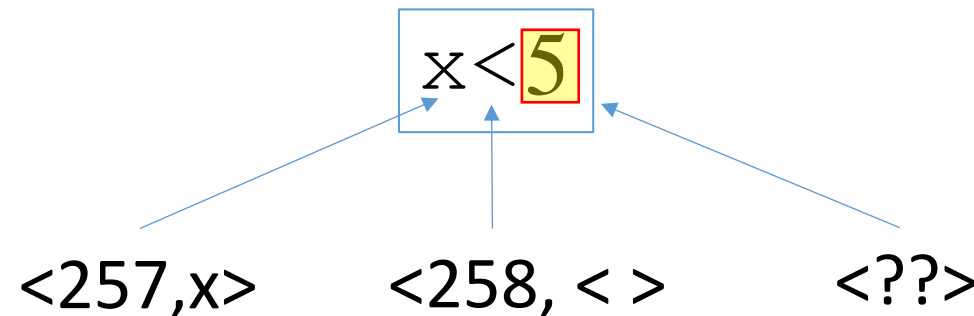
- Esempio (2):
  - Per l'input  $x < 5$ , dopo la lettura del  $<$ , non è ancora chiaro se il token corrisponde a  $<258, <= >$ ,  $<258, < >$  oppure  $<258, < >$ .
  - Quindi il simbolo dopo  $<$ , cioè 5, deve essere letto.
  - A quel punto, è chiaro che il token per l'operatore relazionale è  $<258, < >$ .
  - 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 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, deve 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

# 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 stringa generata da  $(a+...+z+A+...+Z)(a+...+z+A+...+Z+0+...+9)^*$ ).
- 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.