LEXICAL ANALYSIS

LEXICAL ANALYSIS

- The task of lexical analyzer is processing data coming from the input stream and recognize strings constructed in accordance with certain principles;

- Lexical analysis is the first phase of compilation;

REGULAR LANGUAGES

- Alphabet - we call any finite set. It is usually denoted symbol Σ;
- Elements of the alphabet Σ we called symbols;
- Example of the alphabet $\Sigma=\{a,b,c,d,e,1\}$, but the alphabet is not $\Sigma=\{1,2,3,\ldots\}$.

REGULAR LANGUAGES

- **In a word** formed from the alphabet Σ call any finite, ordered arrangement of the elements of Σ;
- Otherwise, the word is any element of the Cartesian product $\Sigma^n = \Sigma \times \Sigma \times \ldots \times \Sigma$ (n-times);
- The system of 0 symbols of the alphabet Σ is called **the empty word** and is denoted Λ;
- **The word length** is the number of symbols in a word.

REGULAR LANGUAGES

- If Σ is alphabet, **the closure** of Σ is the set of all words formed from the alphabet Σ and is denote Σ*;

- **Formal language** (the language) is called any set of words from any alphabet Σ. In other words, formal language call any set L such that $L \subset \Sigma^*$;

REGULAR LANGUAGES

- Let Σ be the alphabet. If $A,B \in \Sigma^*$ are the empty words, then A **submission** from B (konkatencją A with B) is called word, AB, ie. a summary of these words.

- If $A=a_1a_2\ldots a_n$, $B=b_1b_2\ldots b_m$, then

  $AB:=a_1a_2\ldots a_nb_1b_2\ldots b_m$,

REGULAR LANGUAGES

- **A prefix** of word A we call any word $B \in \Sigma^*$ such that $A=BC$ for some word $C \in \Sigma^*$;
- **A sufix** of word A we call any word $B \in \Sigma^*$ such that $A=CB$ for some word $C \in \Sigma^*$;
- A subword of word A we call any word $B \in \Sigma^*$ such that $A=CBD$ for some words $C,D \in \Sigma^*$;

REGULAR LANGUAGES

Let Σ be the alphabet and let $L_1, L_2 \in \Sigma^*$ be arbitrary languages from Σ;

- **The sum** $L_1 + L_2$ is the set of all words belonging to the L_1 or L_2. Otherwise it is the classical sum of the sets which are those languages;

The filing $L_1 \cdot L_2$ is the set of all possible words, which are filing words from L_1 with words from L_2;

REGULAR LANGUAGES

- **Closure** L_1^* of language L_1 is a set of all words from Σ, which are finite assembiles of words from L_1 plus empty word Λ, namely:

  $L_1^* = \{A \epsilon\ \Sigma^* : A = \Lambda$ or $A = A_1 A_2 \ldots A_n$, where

  $A_i \epsilon L_1$, for $i = 1,2,\ldots,n$ i $n \geq 1\}$;

- **Padding** $L' := \Sigma^* \backslash L$, is a set of words from Σ, that does not belong to L;

REGULAR LANGUAGES

- A regular expression is any meaningful expression created with bold symbols of the alphabet Σ, operators ⋅,+,*, parenthesis and additional symbols **Λ**, **Ø;**
- The language of regular expressions over the alphabet Σ (**WR**) we call the smallest language of the symbols of the new alphabet **Σ** satisfying the following two conditions:
  - $a_1,\ldots,a_n$, **Λ, Ø** ∈ L;
  - **A,B** ϵ L ⇨ **(A+B), (A⋅B), A*, B*** ∈ L

REGULAR LANGUAGES

- The regular expression can also be defined as any word language **WR**;

- Let $\Sigma=\{a_1, a_2, \ldots, a_n\}$ be an arbitrary alphabet and **WR** language of regular expressions over the alphabet Σ. Each of the regular expression $\mathbf{A} \in \mathbf{WR}$ assign the language $L(\mathbf{A})$ over alphabet Σ using inductive definition depending on the number of symbols +, ·,* in regular expression **A**.

EXAMPLES

- Let $\Sigma=\{a,b\}$ be arbitrary alphabet. Then:

- $L(\mathbf{a})=\{a\}$;
- $L(\mathbf{a+b})=\{a,b\}$;
- $L(\mathbf{a}^*)=\{a\}^*$;
- $L((\mathbf{a+b})^*)=\{a,b\}^*$

REGULAR LANGUAGES

- Formal language over the alphabet Σ is called a regular language, when it is generated by some regular expression, ie. there exists a regular expression **A** over Σ such that L=L(**A**);

- The set of all regular languages we denote **JR**;

Example

- Let Σ={a,b} be the alphabet. We have the following regular expression

  ((ab)*b + ab*)*;

- Above regular expression generate the following language

  L(((**ab**)***b** + **ab***)***)={a,b}*;

Example

Indeed, $L((\mathbf{ab})^*\mathbf{b}+\mathbf{ab}^*) \subset L((\mathbf{a}+\mathbf{b})^*)=\{a,b\}^*$

Just to show that

$$L((\mathbf{a}+\mathbf{b})^*) \subset L((\mathbf{ab})^*\mathbf{b}+\mathbf{ab}^*)$$

- $(\Lambda b + a\Lambda)^* \in L((\mathbf{ab})^*\mathbf{b}+\mathbf{ab}^*)$
- Hence $L((\mathbf{ab})^*\mathbf{b}+\mathbf{ab}^*) = L((\mathbf{a}+\mathbf{b})^*) = \{a,b\}^*$;

A FINITE AUTOMATA

Deterministic finite state machine (automatic) we call a system $M=(Q,\Sigma,s_0,F,\delta)$ consisting of the following components:

1. $Q=\{s_0,\ldots,s_m\}$ – finite set whose elements are called states of M;
2. $\Sigma=\{a_1,\ldots,a_n\}$ – finite set called the alphabet of M;

A FINITE AUTOMATA

1. s_0 – the special state, which we call the initial state;
2. $F \subset Q$ – the special subset of states, which elements are called final states;
3. $\delta : Q \times \Sigma \rightarrow Q$ – function, called the move function of M, which the state from Q and symbol of the alphabet Σ (ie. each pair (s, a)) assigns a new state of automat M;

The set of all deterministic finite automata is denoted by **DAS.**

A FINITE AUTOMATA

The principle of operation of a finite:

- If we have a word from a alphabet and this word is read by a machine symbol after symbol;
- after reading the next symbol of the word, machine changes its state (or remains in the same state) depending on the previous state and a loaded symbol (starts loading the initial state);

A FINITE AUTOMATA

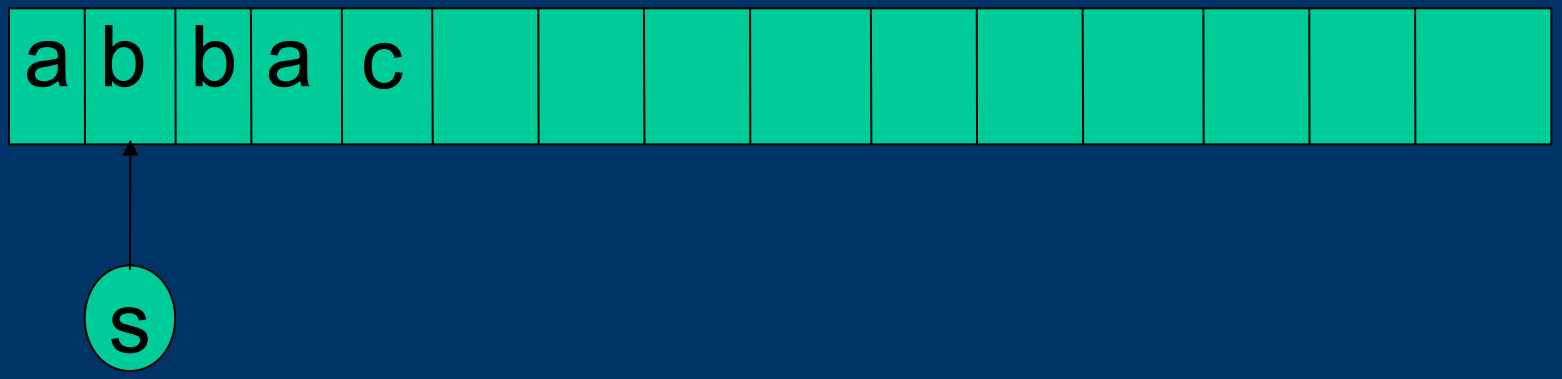
- Machine can take only a finite number of states;
- Machine accepts the word, when after reading the whole word, the machine will be in one of the final states (accept);

A FINITE AUTOMATA

Intuitive approach to the vending machine:

It is given a tape divided into cells (cages), a potentially infinite on the right side, in the cells of which will be written symbols words with the alphabet of the machine. In addition, it is given head, which reads successive symbols of input word, records on tape, and amends their state according to the function moves (move) the machine M.

A FINITE AUTOMATA



The head after reading the next symbol word changes its state and moves to the right one cell. If after reading all the symbols of a word recorded on the tape, the head will be in one of the final states, it means that the word is accepted by the automaton M.

A FINITE AUTOMATA

- The language L made up of some alphabet Σ is called DAS - regular, if there is a deterministic finite automaton $M=(Q,\Sigma,s_0,F,\delta)$ such that

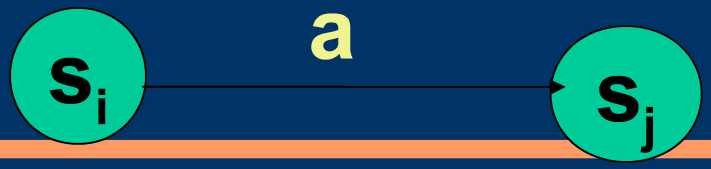
$L= \{A \in \Sigma^* :$ word A is accepted by the machine M$\}$;

We write then L = L (M) and say that language L is accepted by the automaton M. The set of all languages DAS - regular denoted by JDAS.

A FINITE AUTOMATA

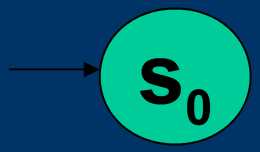
Each machine $M = (Q, \Sigma, s_0, F, \delta)$ can be presented in the form of a graph G (M). The main principles of creating such a graph ar::

- vertices of the graph G (M) are all states $s_i \in Q$ of machine M;
- if $\delta(s_i,a)=s_j$, thus vertices s_i, s_j we connect a directed edge (from s_i to s_j) and denoted by the symbol a;



A FINITE AUTOMATA

- The initial state we denote additional arrow;



- The final states we denote a double line;

Example

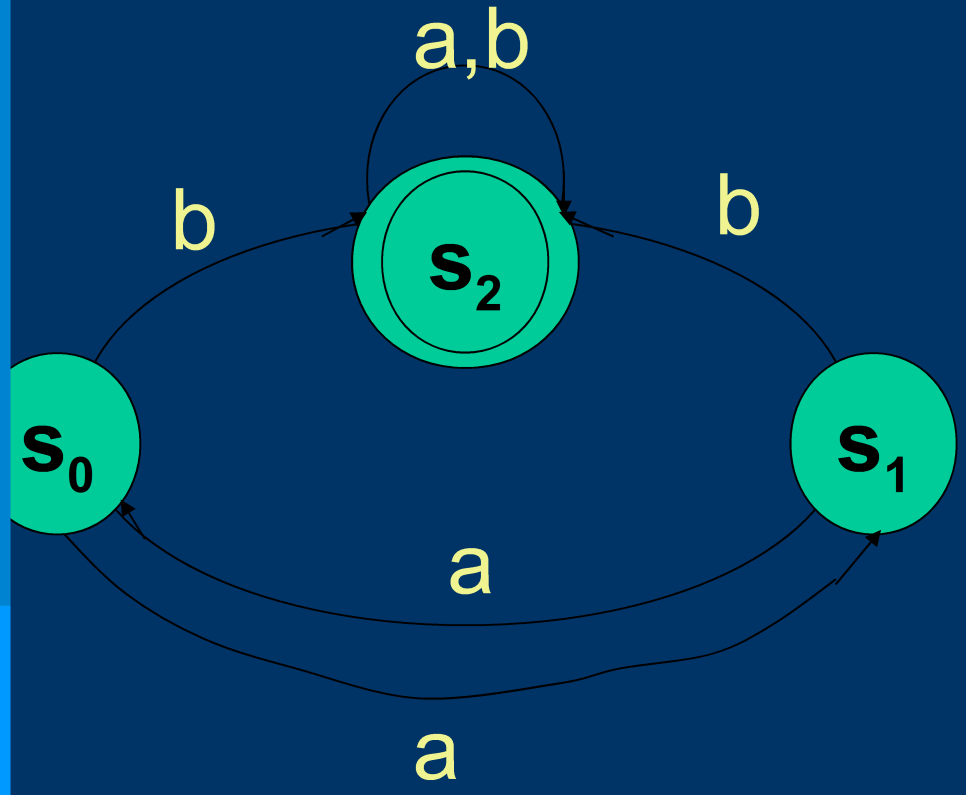
- Let $M=(Q,\Sigma,s_0,F,\delta) = (\{s_0,s_1,s_2\},\{a,b\}, s_0,\{s_2\}, \delta)$ and the move function will be following:

δ	a	b
s_0	s_1	s_2
s_1	s_0	s_2
s_2	s_2	s_2

Example

- The graph of machine:

δ	a	b
s_0	s_1	s_2
s_1	s_0	s_2
s_2	s_2	s_2



A FINITE AUTOMATA

- Non-deterministic finite state machine (automatic) called the system $M=(Q,\Sigma,s_0,F,\delta)$ consisting of the following elements:

1. $Q=\{s_0,\ldots,s_m\}$ – finite set whose elements are called states of M;
2. $\Sigma=\{a_1,\ldots,a_n\}$ – finite set called alphabet M;

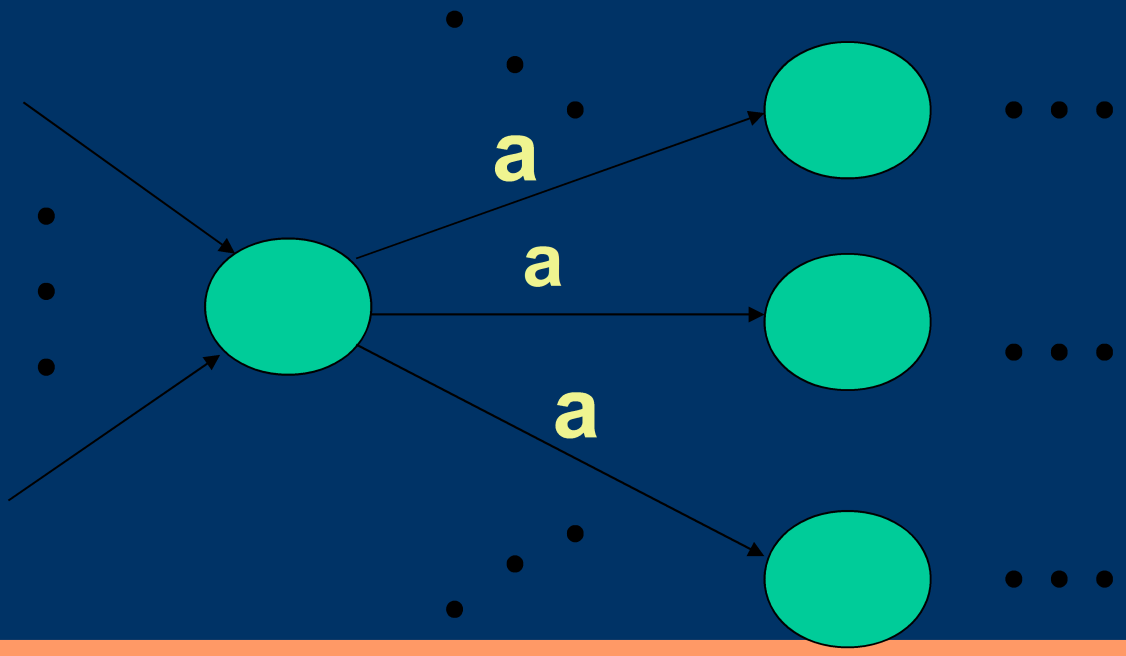
A FINITE AUTOMATA

1. s_0 – the special state, which we call the initial state;
2. $F \subset Q$ – the special subset of states, which elements are called final states;
3. $\delta : Q \times \Sigma \rightarrow \beta(Q)$ – function, called the move function of M, which the state from Q and symbol of the alphabet Σ (ie. each pair (s, a)) assigns a set of state of automat M (mabye empty set);

The set of all non-deterministic finite automata is denoted by **NAS**.

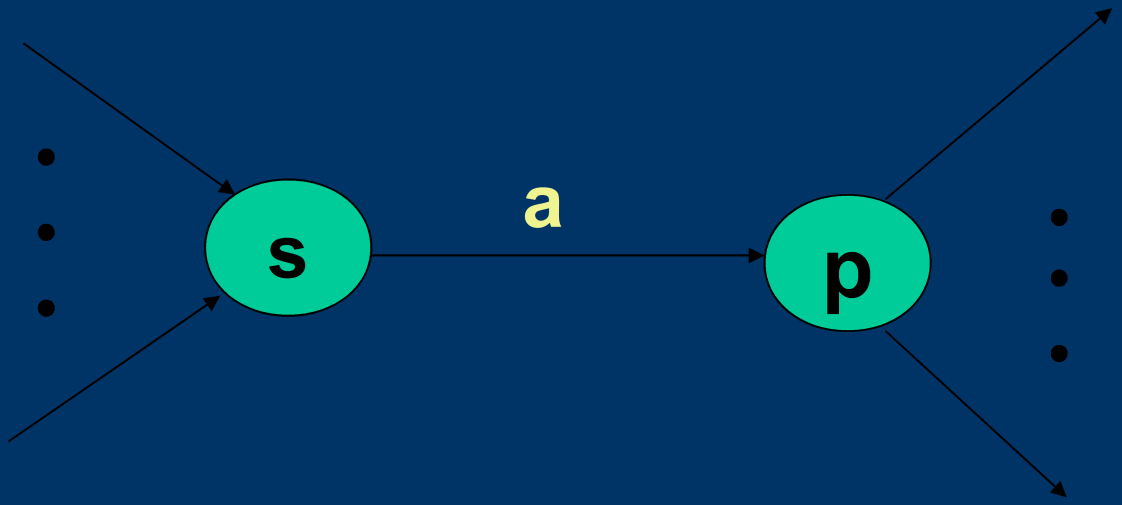
A FINITE AUTOMATA

- Non-deterministic lies in the fact that, if we have the state of the machine and loading the next symbol of word, the automaton could go into one of several possible states.



A FINITE AUTOMATA

- It may happen that a given state does not go any edge of a given symbol, for example, where $\Sigma = \{a, b\}$, it may happen that:



A FINITE AUTOMATA

- We say that the word $A \in \Sigma^*$ is accepted by a non-deterministic finite automaton $M=(Q,\Sigma,s_0,F,\delta)$, when loaded words, the machine will be in one of his final states (symbolically $\delta(s_0,A)\cap F \neq \emptyset$);
- In „the language of graph”, this means that there is a way in this graph with edges marked with the following symbols words A, leading from the initial state into one of the final states.

A FINITE AUTOMATA

The language L made up of some alphabet Σ is called NAS - regular, if there exists non-deterministic finite automaton $M=(Q,\Sigma,s_0,F,\delta)$ such that

$L= \{A \in \Sigma^* : \text{word A is accepted by the machine M}\}$
$=\{A \in \Sigma^* : \delta(s_0,A)\cap F \neq \emptyset\};$

We write then L = L (M) and say that language L is accepted by the automaton M. The set of all languages NAS - regular denoted by **JNAS**.

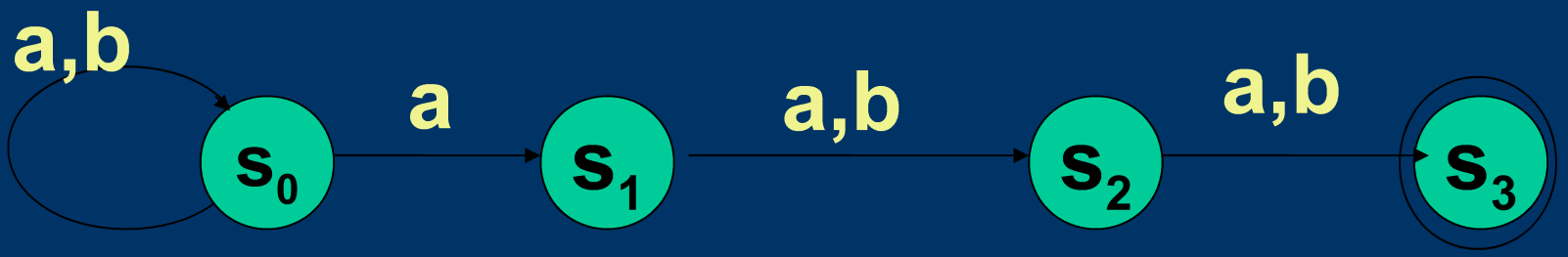
Example

Let $M=(Q,\Sigma,s_0,F,\delta) = (\{s_0,s_1,s_2, s_3\},\{a,b\}, s_0,\{s_3\},$ $\delta)$ function of move is given by the following table:

δ	a	b
s_0	$\{s_0,s_1\}$	$\{s_0\}$
s_1	$\{s_2\}$	$\{s_2\}$
s_2	$\{s_3\}$	$\{s_3\}$

Example

- We show the graph of this non-deterministic finite automaton;



Example

- Language generated by vending machine in the above example, how easily check that looks like this:

L = {A ϵ Σ* : in word A the third symbol of the end is equal to a}

= L(**(a+b)*a(a+b)²**);

A FINITE AUTOMATA

THEOREM 1.

Each deterministic finite automaton is non-deterministic finite automaton, i.e. **DAS** $\subset$ **NAS.**

THEOREM 2.

The family of **DAS** – regular languages is equal to the family of **NAS** – regular languages, i.e. **JDAS=JNAS**.

A FINITE AUTOMATA

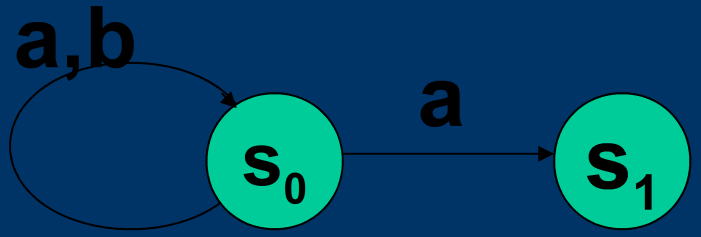
Remark 1.

From Theorem 2 we have that for each NAS there exists an equivalent DAS.

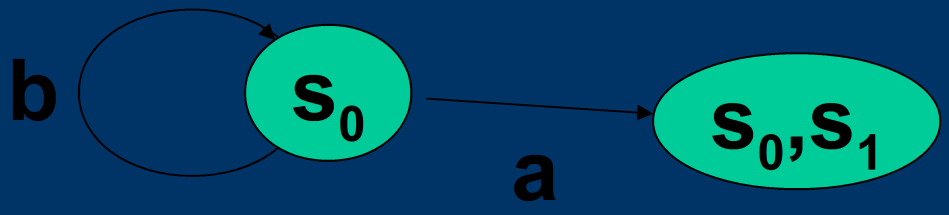
Indeed, when we consider the earlier example of the NAS, it's easy to give him the equivalent of the DAS machine. In the newly constructed automaton (DAS), the states will be sets of states of NAS.

A FINITE AUTOMATA

- non-deterministic automaton graph:



- deterministic automaton graph:



IMPLEMENTATIONS OF DAS

- Let $M=(Q,\Sigma,s_0,F,\delta) = (\{s_0,s_1,s_2\},\{a,b\}, s_0,\{s_2\}, \delta)$ and the move function will be given by the table:...

δ	a	b
s_0	s_1	s_2
s_1	s_0	s_2
s_2	s_2	s_2

CACHING

A buffer is a storage area for storing data for communication between two systems. Buffers allow asynchronous communications between systems.

Example: The PHP functions that only work before sending anything to the recipient. If you need a function call after sending information to the help comes buffer. During compilation collects all the information sent and stores them, and then at the very end of the script displays the collected information.

CACHING

- **Caching** is a technology involving the use of buffers (software or hardware), used mostly to compensate for differences in the speed of data transfer between different devices.

CACHING

One technique for cache explain an example. Consider the function of sign, which we mentioned. Recall:

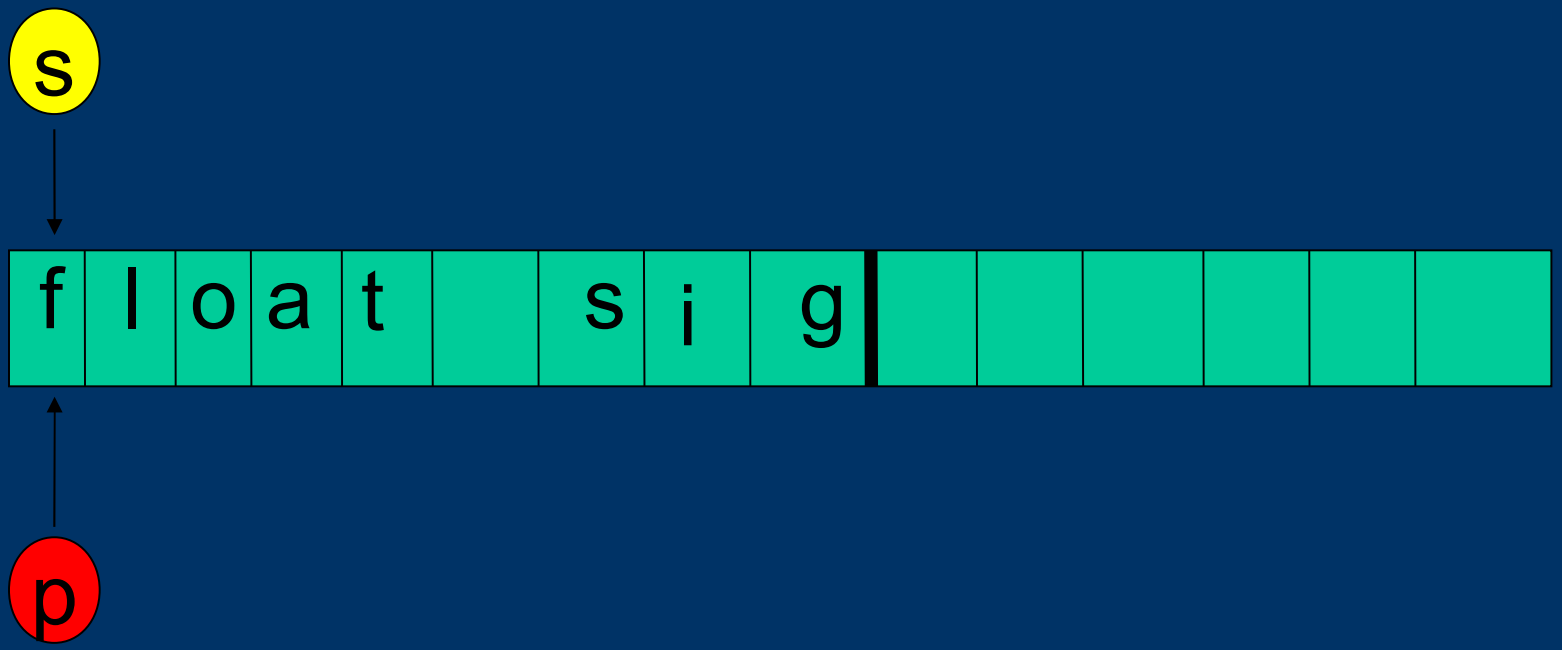
$$sig(x) = \begin{cases} -1 & \text{gdy } x < 0 \\ 0 & \text{gdy } x = 0 \\ 1 & \text{gdy } x > 0 \end{cases}$$

CACHING

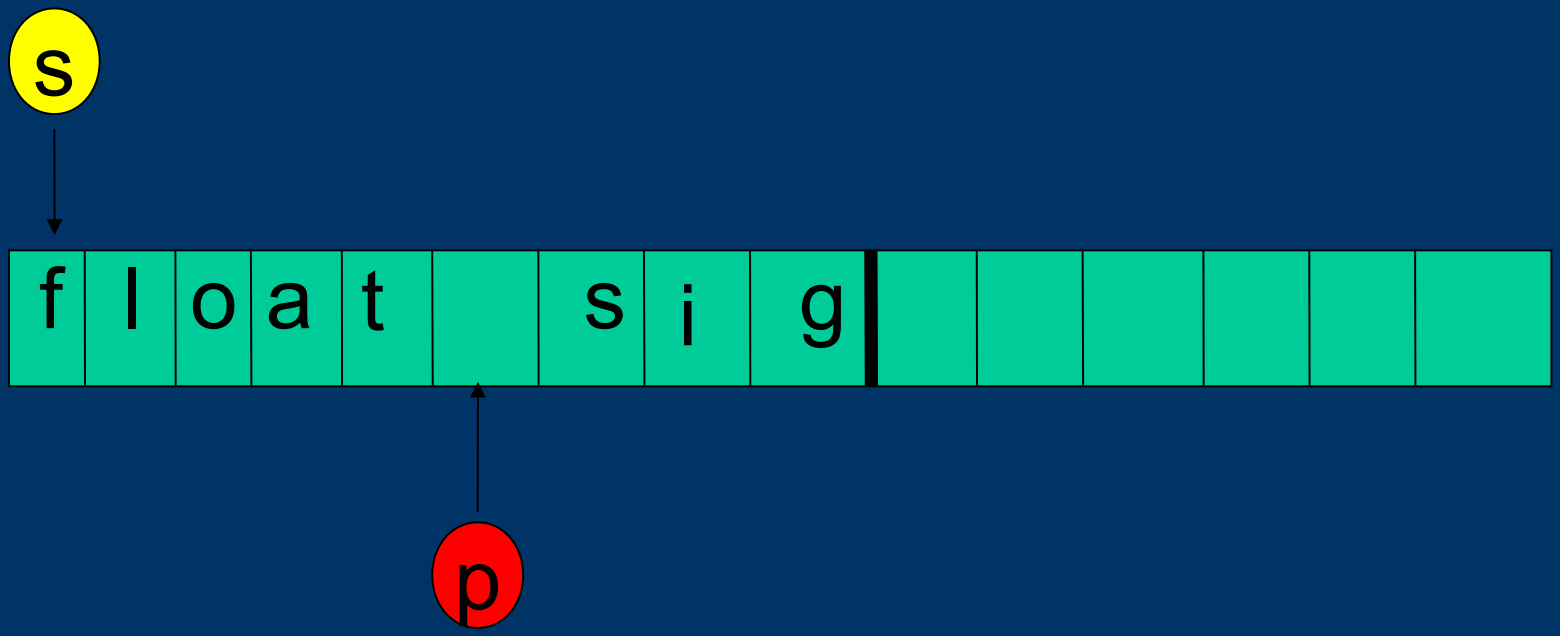
- Recall also the implementation in C ++:

```
float sign (int a)
    {
     if a<0 return -1;
     else if (a==0) return 0;
     else return 1;
    }
```

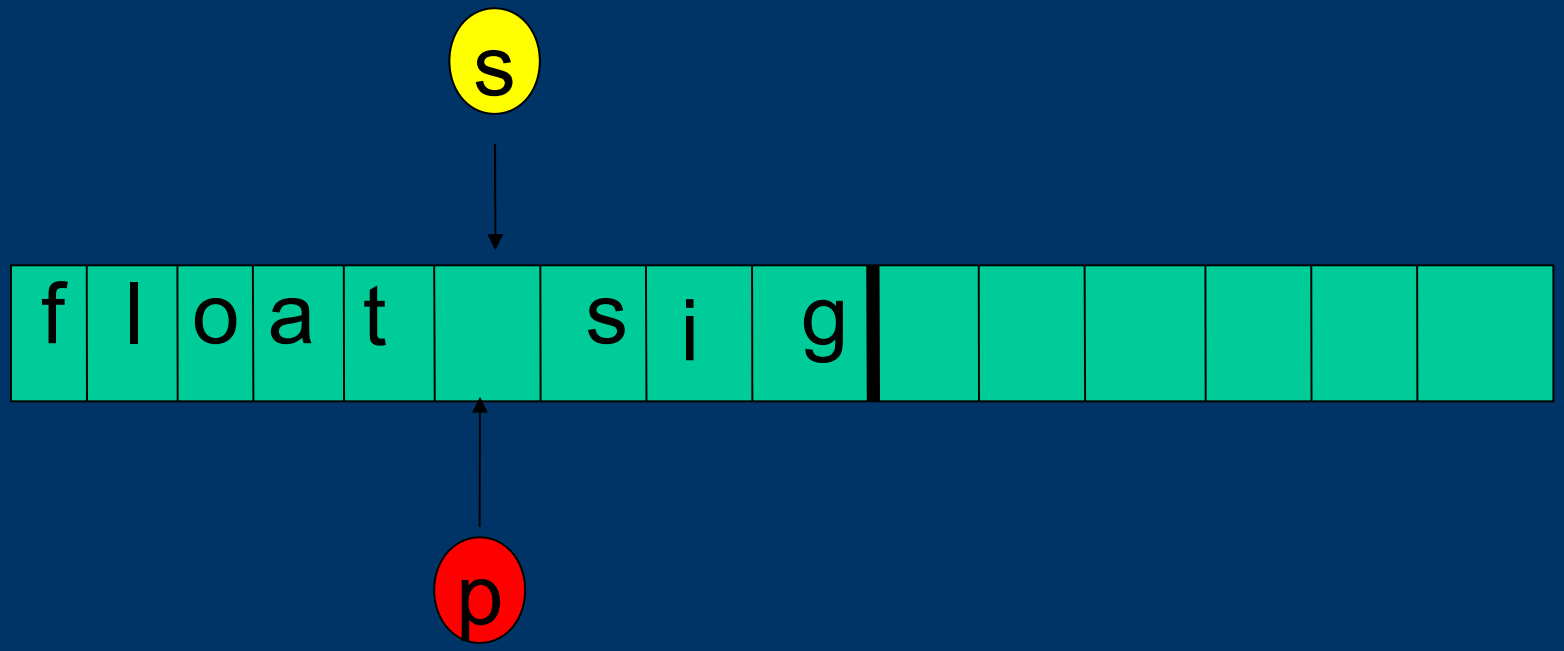
CACHING



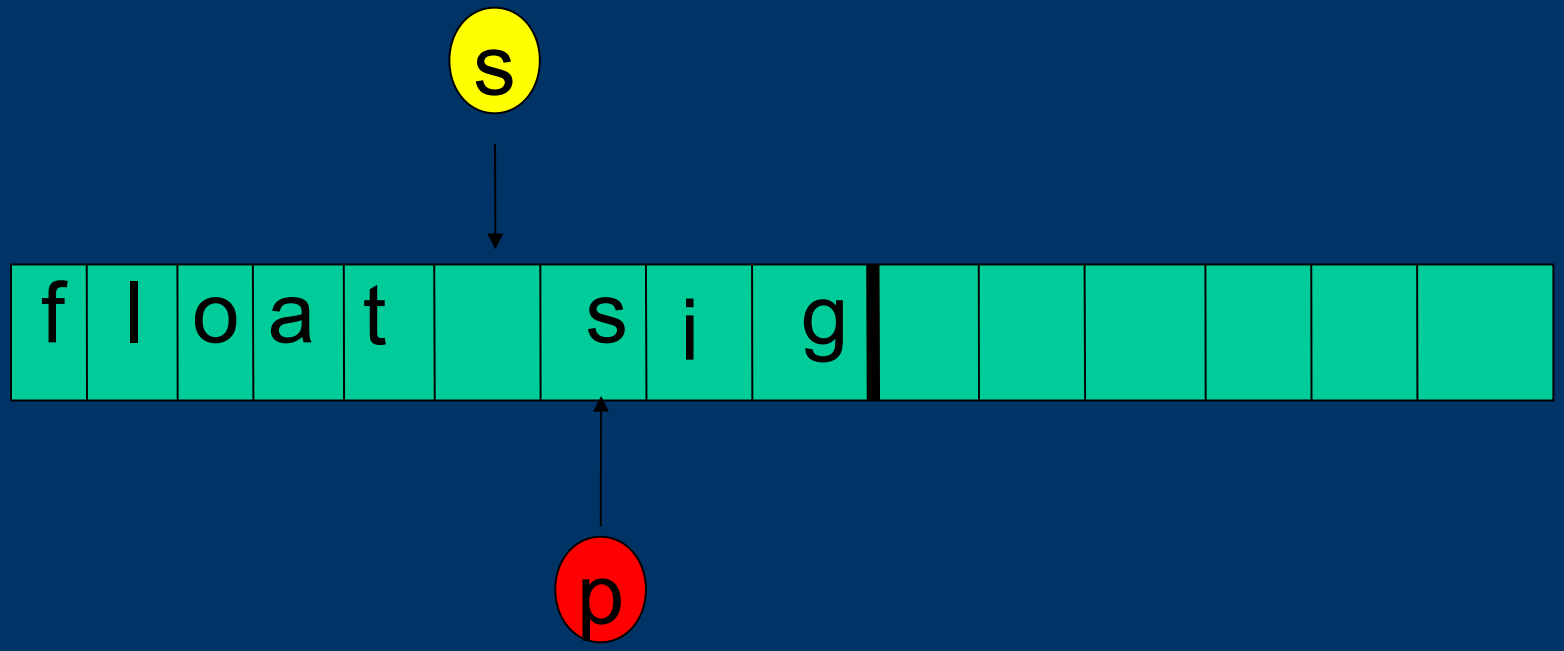
CACHING



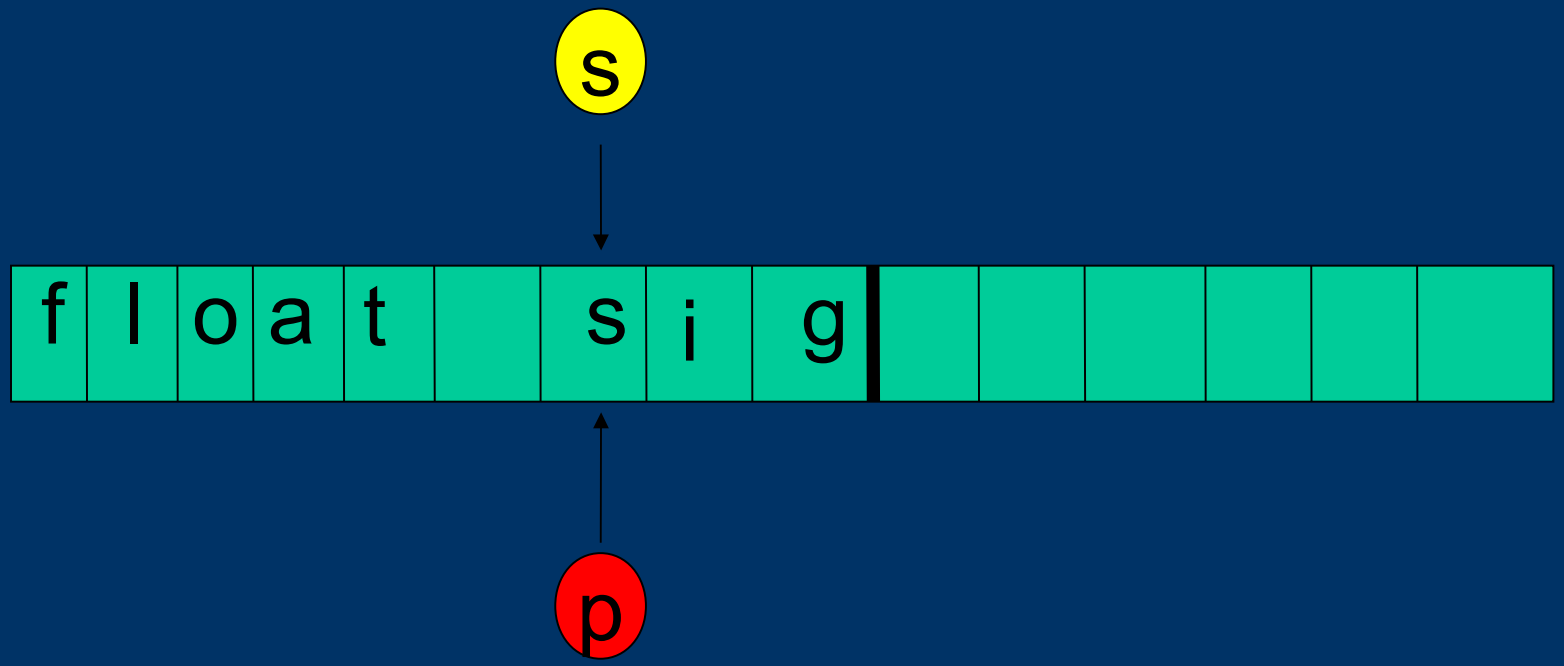
CACHING



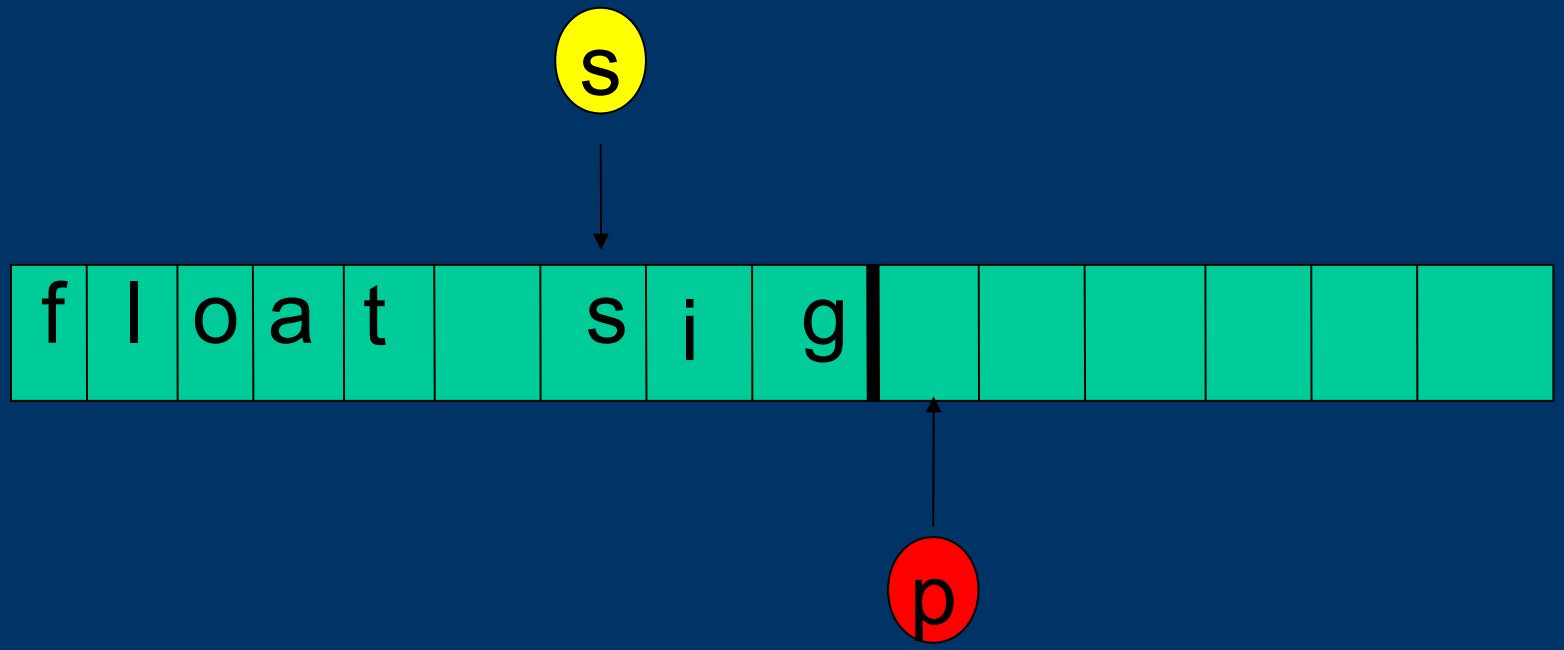
CACHING



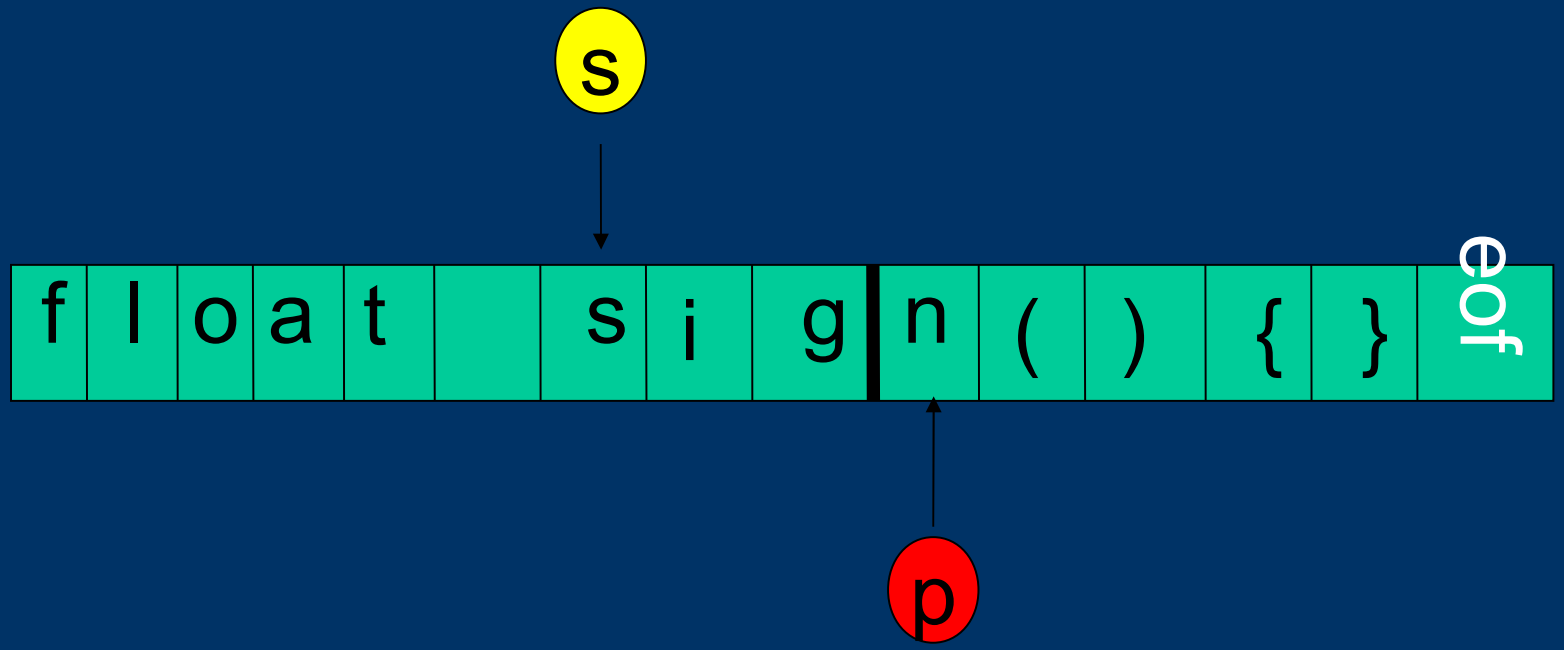
CACHING



CACHING



CACHING



Iplementation:...

END

END OF SECOND LECTURE