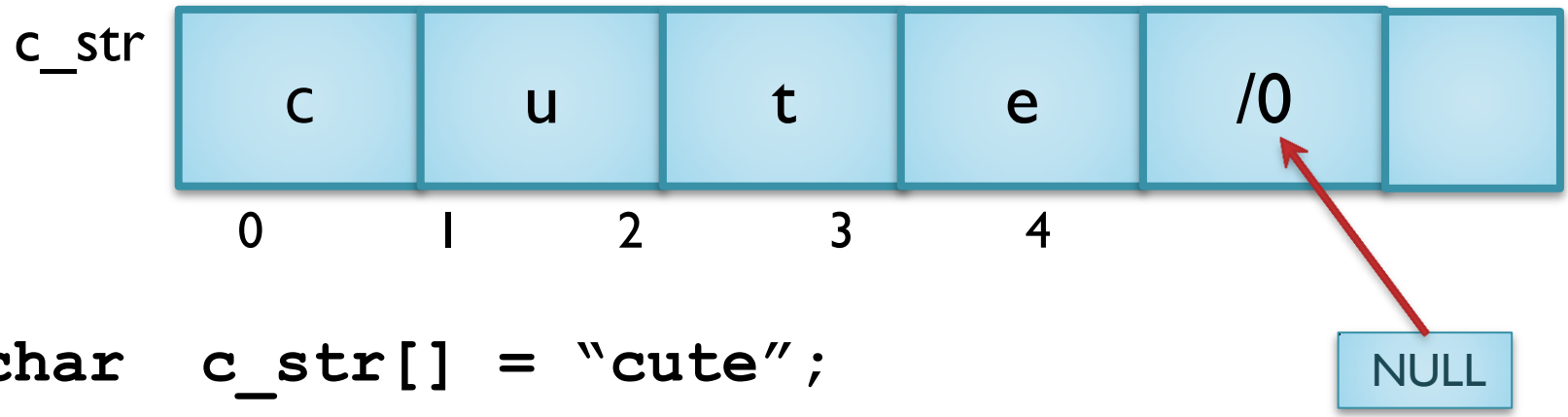
# Алгоритми обробки рядків

«Алгоритмізація та програмування». Лекція 9.

Автор: *к.т.н. доцент каф.301 О.В. Гавриленко*

# НИЗЬКОРІВНЕВІ РЯДКИ




```
char  c_str[] = “cute”;
char * c_str = “cute";
char str[] = {‘c',‘u',‘t', ‘e','\0'};
```

```
//функція підрахунку довжини рядка
int string_length (char str[]) {
    int i;
    for ( i = 0; str[i] != '\0'; i++);
    return(i);
}
```

for ( i = 0; str[i] != NULL; i++);

while (str[i]) i++;

for ( i = 0; str[i]; i++);

# ASCII

## СТАНДАРТНА ЧАСТИНА ANSI

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | | 00100000 | 56 | 8 | 00111000 | 80 | P | 01010000 | 104 | h | 01101000 |
| 33 | ! | 00100001 | 57 | 9 | 00111001 | 81 | Q | 01010001 | 105 | i | 01101001 |
| 34 | " | 00100010 | 58 | : | 00111010 | 82 | R | 01010010 | 106 | j | 01101010 |
| 35 | # | 00100011 | 59 | ; | 00111011 | 83 | S | 01010011 | 107 | k | 01101011 |
| 36 | $ | 00100100 | 60 | < | 00111100 | 84 | T | 01010100 | 108 | l | 01101100 |
| 37 | % | 00100101 | 61 | = | 00111101 | 85 | U | 01010101 | 109 | m | 01101101 |
| 38 | & | 00100110 | 62 | > | 00111110 | 86 | V | 01010110 | 110 | n | 01101110 |
| 39 | ' | 00100111 | 63 | ? | 00111111 | 87 | W | 01010111 | 111 | o | 01101111 |
| 40 | ( | 00101000 | 64 | @ | 01000000 | 88 | X | 01011000 | 112 | p | 01110000 |
| 41 | ) | 00101001 | 65 | A | 01000001 | 89 | Y | 01011001 | 113 | q | 01110001 |
| 42 | * | 00101010 | 66 | B | 01000010 | 90 | Z | 01011010 | 114 | r | 01110010 |
| 43 | + | 00101011 | 67 | C | 01000011 | 91 | [ | 01011011 | 115 | s | 01110011 |
| 44 | , | 00101100 | 68 | D | 01000100 | 92 | \ | 01011100 | 116 | t | 01110100 |
| 45 | - | 00101101 | 69 | E | 01000101 | 93 | ] | 01011101 | 117 | u | 01110101 |
| 46 | . | 00101110 | 70 | F | 01000110 | 94 | ^ | 01011110 | 118 | v | 01110110 |
| 47 | / | 00101111 | 71 | G | 01000111 | 95 | _ | 01011111 | 119 | w | 01110111 |
| 48 | 0 | 00110000 | 72 | H | 01001000 | 96 | ` | 01100000 | 120 | x | 01111000 |
| 49 | 1 | 00110001 | 73 | I | 01001001 | 97 | a | 01100001 | 121 | y | 01111001 |
| 50 | 2 | 00110010 | 74 | J | 01001010 | 98 | b | 01100010 | 122 | z | 01111010 |
| 51 | 3 | 00110011 | 75 | K | 01001011 | 99 | c | 01100011 | 123 | { | 01111011 |
| 52 | 4 | 00110100 | 76 | L | 01001100 | 100 | d | 01100100 | 124 | ¦ | 01111100 |
| 53 | 5 | 00110101 | 77 | M | 01001101 | 101 | e | 01100101 | 125 | } | 01111101 |
| 54 | 6 | 00110110 | 78 | N | 01001110 | 102 | f | 01100110 | 126 | ~ | 01111110 |
| 55 | 7 | 00110111 | 79 | O | 01001111 | 103 | g | 01100111 | 127 | □ | 01111111 |

# ASCII

## КОДОВА СТОРІНКА 1251

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | Ъ | 10000000 | 160 | " | 10100000 | 192 | А | 11000000 | 224 | а | 11100000 |
| 129 | Ѓ | 10000001 | 161 | ў | 10100001 | 193 | Б | 11000001 | 225 | б | 11100001 |
| 130 | ‚ | 10000010 | 162 | Ў | 10100010 | 194 | В | 11000010 | 226 | в | 11100010 |
| 131 | ѓ | 10000011 | 163 | Ј | 10100011 | 195 | Г | 11000011 | 227 | г | 11100011 |
| 132 | „ | 10000100 | 164 | ¤ | 10100100 | 196 | Д | 11000100 | 228 | д | 11100100 |
| 133 | … | 10000101 | 165 | Ґ | 10100101 | 197 | Е | 11000101 | 229 | е | 11100101 |
| 134 | † | 10000110 | 166 | | 10100110 | 198 | Ж | 11000110 | 230 | ж | 11100110 |
| 135 | ‡ | 10000111 | 167 | § | 10100111 | 199 | З | 11000111 | 231 | з | 11100111 |
| 136 | € | 10001000 | 168 | Ё | 10101000 | 200 | И | 11001000 | 232 | и | 11101000 |
| 137 | ‰ | 10001001 | 169 | © | 10101001 | 201 | Й | 11001001 | 233 | й | 11101001 |
| 138 | Љ | 10001010 | 170 | Є | 10101010 | 202 | К | 11001010 | 234 | к | 11101010 |
| 139 | ‹ | 10001011 | 171 | « | 10101011 | 203 | Л | 11001011 | 235 | л | 11101011 |
| 140 | Њ | 10001100 | 172 | ¬ | 10101100 | 204 | М | 11001100 | 236 | м | 11101100 |
| 141 | Ќ | 10001101 | 173 | | 10101101 | 205 | Н | 11001101 | 237 | н | 11101101 |
| 142 | Ћ | 10001110 | 174 | ® | 10101110 | 206 | О | 11001110 | 238 | о | 11101110 |
| 143 | Џ | 10001111 | 175 | Ї | 10101111 | 207 | П | 11001111 | 239 | п | 11101111 |
| 144 | ђ | 10010000 | 176 | ° | 10110000 | 208 | Р | 11010000 | 240 | р | 11110000 |
| 145 | ‘ | 10010001 | 177 | ± | 10110001 | 209 | С | 11010001 | 241 | с | 11110001 |
| 146 | ’ | 10010010 | 178 | І | 10110010 | 210 | Т | 11010010 | 242 | т | 11110010 |
| 147 | “ | 10010011 | 179 | і | 10110011 | 211 | У | 11010011 | 243 | у | 11110011 |
| 148 | ” | 10010100 | 180 | ґ | 10110100 | 212 | Ф | 11010100 | 244 | ф | 11110100 |
| 149 | • | 10010101 | 181 | µ | 10110101 | 213 | Х | 11010101 | 245 | х | 11110101 |
| 150 | – | 10010110 | 182 | ¶ | 10110110 | 214 | Ц | 11010110 | 246 | ц | 11110110 |
| 151 | — | 10010111 | 183 | · | 10110111 | 215 | Ч | 11010111 | 247 | ч | 11110111 |
| 152 | | 10011000 | 184 | ё | 10111000 | 216 | Ш | 11011000 | 248 | ш | 11111000 |
| 153 | ™ | 10011001 | 185 | № | 10111001 | 217 | Щ | 11011001 | 249 | щ | 11111001 |
| 154 | љ | 10011010 | 186 | є | 10111010 | 218 | Ъ | 11011010 | 250 | ъ | 11111010 |
| 155 | › | 10011011 | 187 | » | 10111011 | 219 | Ы | 11011011 | 251 | ы | 11111011 |
| 156 | њ | 10011100 | 188 | ј | 10111100 | 220 | Ь | 11011100 | 252 | ь | 11111100 |
| 157 | ќ | 10011101 | 189 | S | 10111101 | 221 | Э | 11011101 | 253 | э | 11111101 |
| 158 | ћ | 10011110 | 190 | s | 10111110 | 222 | Ю | 11011110 | 254 | ю | 11111110 |

# КЛАС string

```
#include <iostream>
#include <string>

int main()
{
    std::string str;

    std::cout << "Enter your name: ";
    std::getline(std::cin, str);

    std::cout << "Hello, " << str << "!!! \n";

    return 0;
}
```

# КЛАСstring

| | |
|---|---|
| (конструктор) | створює basic_string |
| (деструктор) | знищує рядок, звільняючи внутрішню пам'ять, що використовується |
| operator = | надає значення рядку |
| assign | надає символи рядку |
| get_allocator | повертає пов'язаний аллокатор |

## ДОСТУП ДО ЕЛЕМЕНТІВ

| | |
|---|---|
| at | звертається до вказаного символу з перевіркою кордонів |
| operator[] | отримує доступ до вказаного символу |
| front | звертається до першого символу |
| back | отримує доступ до останнього символу |
| data | повертає вказівник на перший символ рядка |
| c_str | повертає немодифіковану стандартну версію масиву символів С рядок |

# КЛАСstring

## ІТЕРАТОРИ

| | |
|---|---|
| begin /cbegin | повертає ітератор на початок |
| end /cend | повертає ітератор на кінець |
| rbegin/crbegin | повертає зворотний ітератор на початок |
| rend /crend | повертає зворотний ітератор на кінець |

## МІСТКІСТЬ

| | |
|---|---|
| empty | перевіряє, чи пустий рядок |
| size/length | повертає кількість символів |
| max_size | повертає максимальну кількість символів |
| reserve | резервує пам'ять |
| capacity | повертає кількість символів, які можуть зберігатися у виділеній на даний момент пам'яті |
| shrink_to_fit | зменшує використання пам'яті за рахунок звільнення невикористовуваної пам'яті |

# КЛАСstring

## ОПЕРАЦІЇ

| | |
|---|---|
| clear | очищує вміст |
| insert | вставляє символи |
| erase | видаляє символи |
| push_back | додає символ у кінець |
| pop_back | видаляє останній символ |
| append | додає символи в кінець |
| operator += | додає символи в кінець |
| compare | порівнює два рядки |
| replace | замінює вказану частину рядка |
| substr | повертає підрядок |
| copy | копіює символи |
| resize | змінює кількість збережених символів |
| swap | міняє місцями вміст |

# КЛАСstring

## ПОШУК

| | |
|---|---|
| find | знаходить перше входження заданого підрядка |
| rfind | шукає останнє входження підрядка |
| find_first_of | шукає перше входження символів |
| find_first_not_of | шукає першу відсутність символів |
| find_last_of | шукає останнє входження символів |
| find_last_not_of | шукає останню відсутність символів |

## ОПЕРАТОРИ

| | |
|---|---|
| operator+ | об'єднує два рядки або рядок і символ |
| operator==<br>operator!=<br>operator<<br>operator><br>operator<=<br>operator>=<br>operator<=> | лексикографічно порівнює два рядки |

# КЛАСstring

## ВВІД ВИВІД

| | |
|---|---|
| operator <<<br>operator >> | виконує потокове введення і виведення рядків |
| getline | зчитує дані з потоку введення-виведення рядок |

## ЧИСЛОВІ ПЕРЕТВОРЕННЯ

| | |
|---|---|
| stoi/stol/stoll | перетворює рядок на ціле число зі знаком |
| stoul/stoull | перетворює рядок на ціле число без знаку |
| Stof/ stod/stold | перетворює рядок на значення з плаваючою крапкою |
| to_string | перетворює ціле значення або значення з плаваючою крапкою у string |
| to_wstring | перетворює ціле значення або значення з плаваючою крапкою у wstring |

# КЛАС string

- Документація англійською:

http://www.cplusplus.com/reference/string/string/

```
bool empty() const;
```

**Test if string is empty**

Returns whether the string is empty (i.e. whether its length is 0).

This function does not modify the value of the string in any way. To clear the content of a string, see string::clear.

### Parameters

none

### Return Value

true if the string length is 0, false otherwise.

# КЛАС string

**Example**

```
1  // string::empty
2  #include <iostream>
3  #include <string>
4
5  int main ()
6  {
7    std::string content;
8    std::string line;
9    std::cout << "Please introduce a text. Enter an empty line to finish:\n";
10   do {
11     getline(std::cin,line);
12     content += line + '\n';
13   } while (!line.empty());
14   std::cout << "The text you introduced was:\n" << content;
15   return 0;
16 }
```

This program reads the user input line by line and stores it into string `content` until an empty line is introduced.

# КЛАС string

**Example**

```
1  // string::length
2  #include <iostream>
3  #include <string>
4
5  int main ()
6  {
7    std::string str ("Test string");
8    std::cout << "The size of str is " << str.length() << " bytes.\n";
9    return 0;
10 }
```

Output:

```
The size of str is 11 bytes
```

# КЛАС string

## std::string::erase

C++98 | C++11

```
sequence (1)    string& erase (size_t pos = 0, size_t len = npos);
character (2)   iterator erase (iterator p);
range (3)       iterator erase (iterator first, iterator last);
```

**Erase characters from string**

Erases part of the string, reducing its length:

***(1) sequence***

Erases the portion of the string value that begins at the character position *pos* and spans *len* charact
the *end of the string*, if either the content is too short or if *len* is string::npos.
Notice that the default argument erases all characters in the string (like member function clear).

***(2) character***

Erases the character pointed by *p*.

***(3) range***

Erases the sequence of characters in the range [first,last).

# КЛАС string

## Parameters

`pos`

Position of the first character to be erased.
If this is greater than the string length, it throws out_of_range.
Note: The first character in *str* is denoted by a value of `0` (not `1`).

`len`

Number of characters to erase (if the string is shorter, as many characters as possible are erased).
A value of string::npos indicates all characters until the end of the string.

`p`

Iterator to the character to be removed.

`first, last`

Iterators specifying a range within the string] to be removed: `[first,last)`. i.e., the range includes all the characters between *first* and *last*, including the character pointed by *first* but not the one pointed by *last*.

`size_t` is an unsigned integral type (the same as member type `string::size_type`).
Member types `iterator` and `const_iterator` are random access iterator types that point to characters of the string.

## Return value

The *sequence version (1)* returns `*this`.
The others return an iterator referring to the character that now occupies the position of the first character erased, or string::end if no such character exists.

Member type `iterator` is a random access iterator type that points to characters of the string.

# КЛАС string

**Example**

```
1  // string::erase
2  #include <iostream>
3  #include <string>
4
5  int main ()
6  {
7    std::string str ("This is an example sentence.");
8    std::cout << str << '\n';
9                                          // "This is an example sentence."
10   str.erase (10,8);                     //            ^^^^^^^^
11   std::cout << str << '\n';
12                                         // "This is an sentence."
13   str.erase (str.begin()+9);            //           ^
14   std::cout << str << '\n';
15                                         // "This is a sentence."
16   str.erase (str.begin()+5, str.end()-9);  //       ^^^^^
17   std::cout << str << '\n';
18                                         // "This sentence."
19   return 0;
20 }
```

Output:

```
This is an example sentence.
This is an sentence.
This is a sentence.
This sentence.
```

# Домашнє завдання

- Тема 9 – конспект Бібліотека <string>
  - http://www.cplusplus.com/reference/string/string/
  - https://www.youtube.com/watch?v=OWwDDAVM44I
- Слайди 6-10:

  2 методи, для кожного:
  - Сигнатура
  - Опис параметрів, результату
  - ПРИКЛАД коду

# Бінарний пошук

- **двійковий пошук**
- або **метод половинного ділення**
- на кожному кроці область пошуку зменшується вдвічі

# Задача

**Дано** ціле число **x** і масив **a [1..n]**, відсортований в порядку неспадання чисел, тобто для будь-якого k:

$$1 \le k < n,\ a[k-1] \le a[k].$$

**Знайти** таке **i**, що a [i] = x або вивести сповіщення, що елемента x в масиві немає.

# Ідея бінарного методу

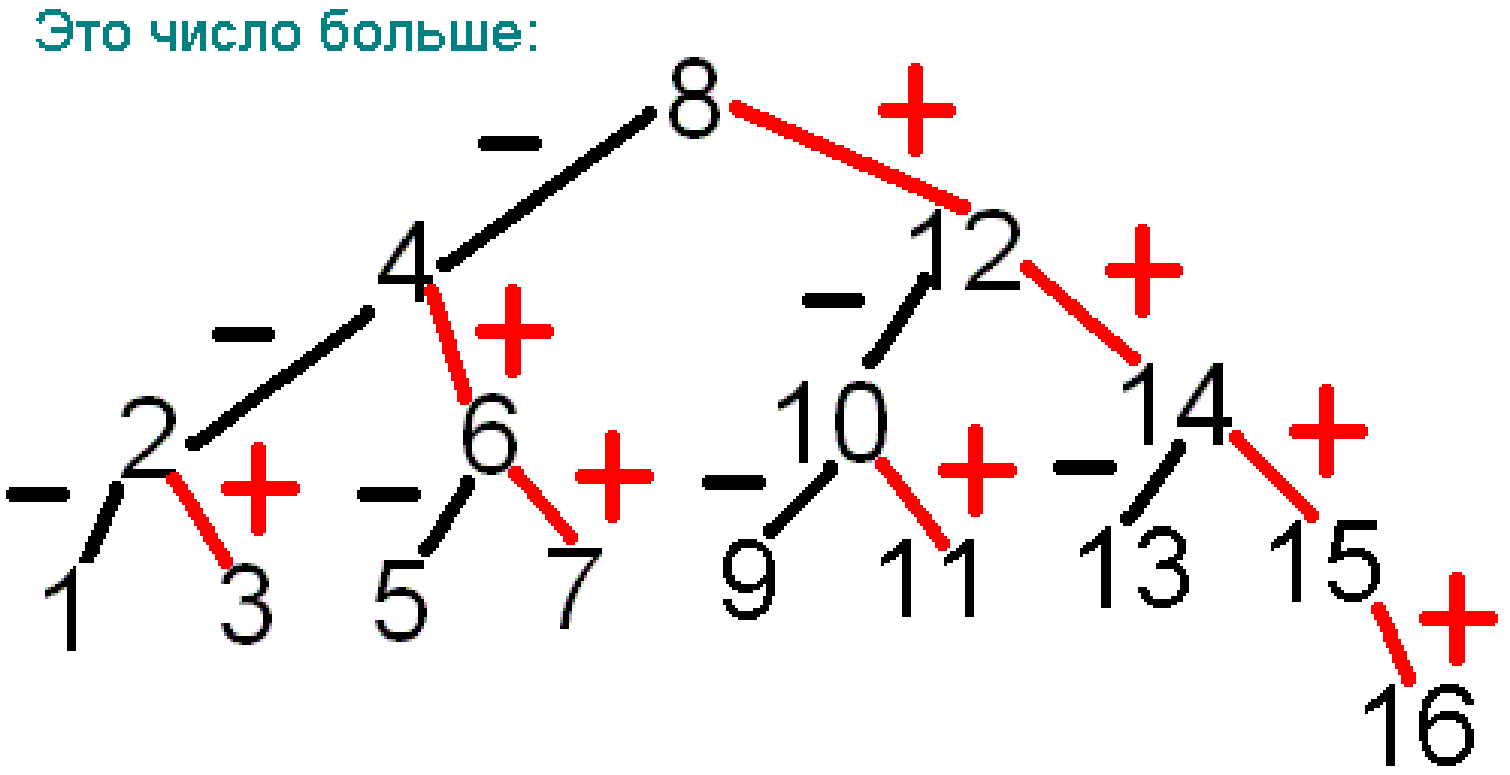
1) перевірити, чи є **x середнім елементом масиву**.

2) Якщо так, то відповідь отримана.

3) Якщо немає, то можливі два випадки:

- **x менше** середнього елемента. Отже, після цього даний метод можна застосувати ***до лівої половини масиву***.
- **x більше** середнього елемента. Аналогічно, тепер цей метод слід застосувати ***до правої частини масиву***.

# Ідея бінарного методу



# Приклад:

Масив $\boldsymbol{a}$:

3 5 6 8 12 15 17 18 20 25

$\boldsymbol{x} = 6$

**Шаг 1.**

Знайдемо номер середнього елемента:

$$\mathbf{m} = \frac{\mathbf{1 + 10}}{\mathbf{2}} = \mathbf{5}$$

Так як 6 < a[5]

3 5 6 8 ~~**12** 15 17 18 20 25~~

# Приклад (продовження):

**Крок 2.** Розглянемо лише перші 4 елемента масиву. Індекс середнього елемента:

**аналогічно:**

$$m = \frac{1 + 4}{2} = 2$$

~~3~~ ~~**5**~~ 6 8 ~~**12**~~ ~~15~~ ~~17~~ ~~18~~ ~~20~~ ~~25~~

**Крок 3.** Розглянемо два елемента

$$m = \frac{3 + 4}{2} = 3$$

~~3~~ ~~**5**~~ **6** 8 ~~**12**~~ ~~15~~ ~~17~~ ~~18~~ ~~20~~ ~~25~~

**A[3] = 6! Його номер = 3**

# Бінарний пошук

*У загальному випадку* формула пошуку значення середнього елемента m:

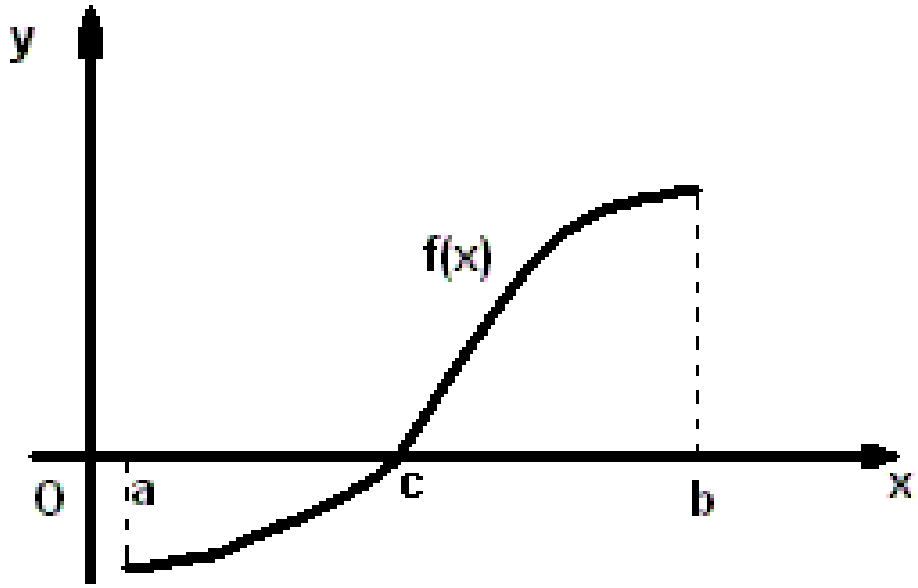
$$m = \frac{L + R}{2}$$

Де **L** - індекс першого,

а **R** - індекс останнього елемента даної частини масиву.

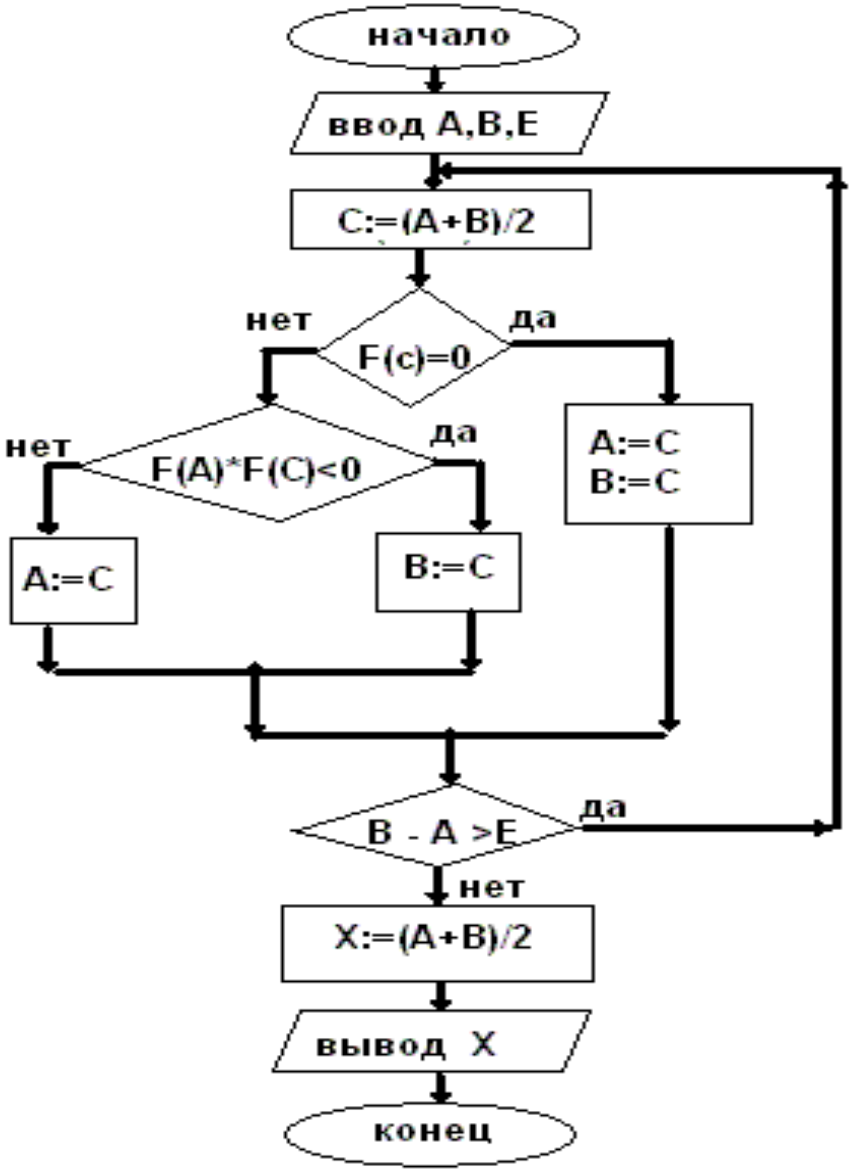
# Приклад

***Пошук кореня функції***




e – задана точність

# Приклад (продовження):



# Пошук підрядка у рядку

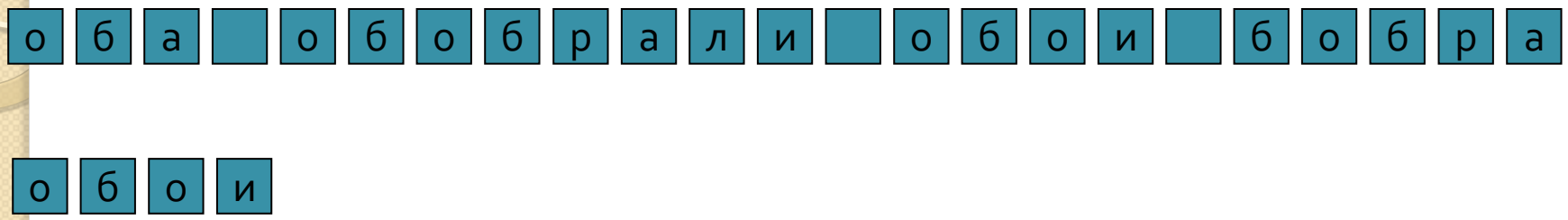
«Наївний» алгоритм

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1-й шаг цикла | H | E | L | L | O | | W | O | R | L | D |
| | L | | | | | | | | | | |
| 2-й шаг цикла | H | E | L | L | O | | W | O | R | L | D |
| | | L | | | | | | | | | |
| 3-й шаг цикла | H | E | L | L | O | | W | O | R | L | D |
| | | | L | O | | | | | | | |
| 4-й шаг цикла | H | E | L | L | O | | W | O | R | L | D |
| | | | | L | O | | | | | | |

# Пошук підрядка у рядку

```
public static int simpleSearch(String where, String what)
{
    int n = where.length();
    int m = what.length();
    extLoop:    // зовнішній цикл пошуку в рядку
    for (int i = 0; i <= n-m; i++) {
        // внутрішній цикл порівняння:
        for (int j = 0; j < m; j++) {
            if (where.charAt(i+j) != what.charAt(j))
                continue extLoop;
        }
        return i;
    }
    return -1;
}
```

# Пошук підрядка у рядку




Кількість порівнянь символів:

3 + 1 + 1 + 1 + 4 + 1 + 3 + 1 + 1 + 1 + 1 + 1 + 1 + 4 = 24

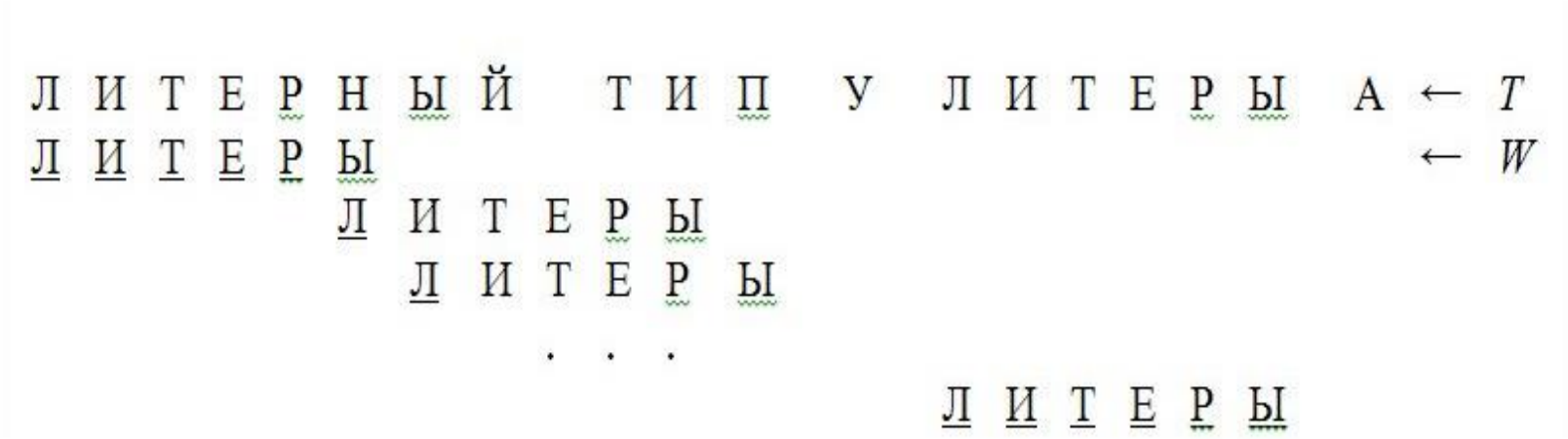
Найгірший випадок:

```
simpleSearch("aaaaaaaaaaaaaaaaaaaaaaaaaaaaab",
             "aaaaaaab");
```

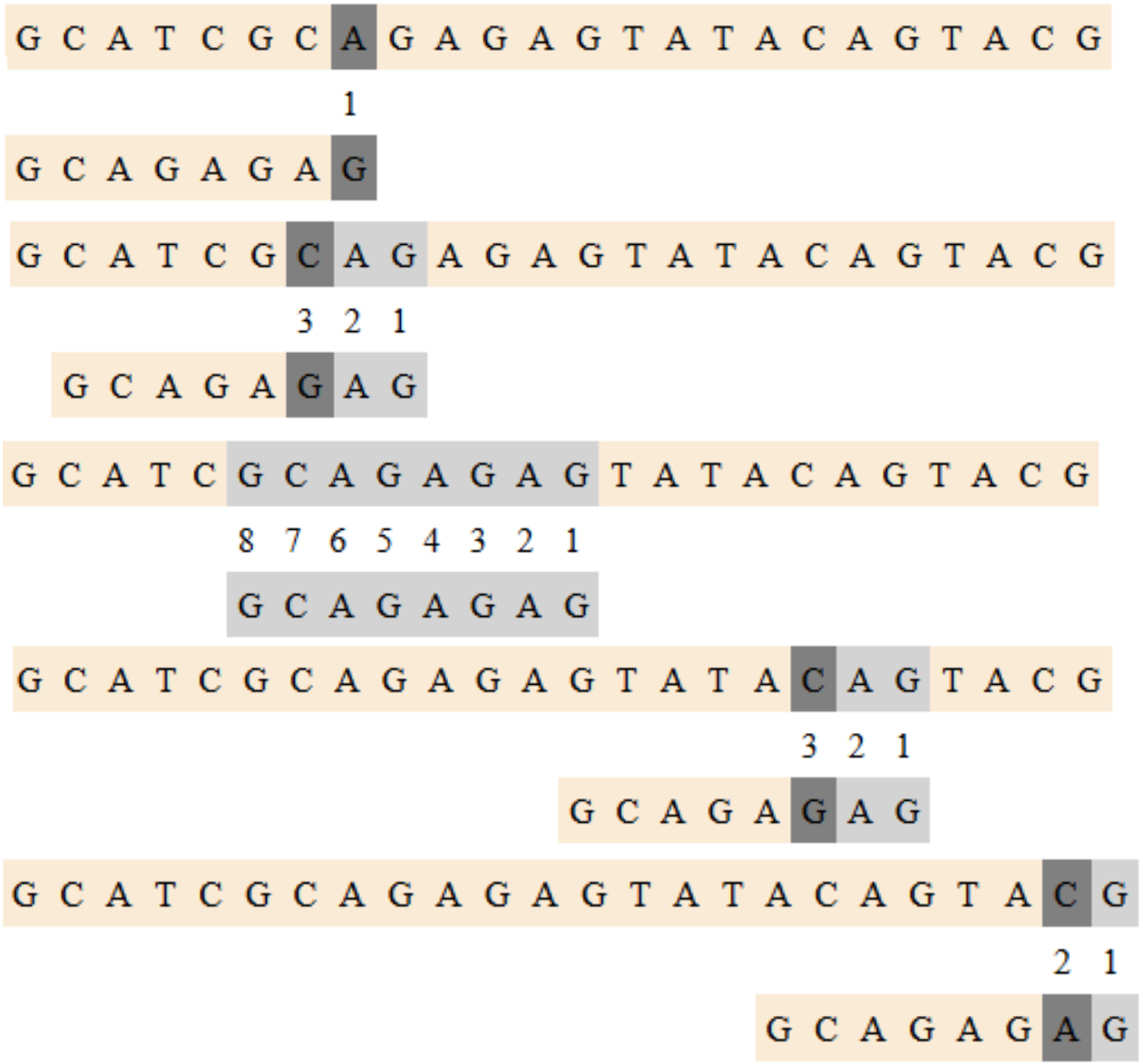
# Покращені алгоритми пошуку

- Зсув зразка
  - Алгоритм Д. Кнута, Д. Моріса й В. Пратта (КМП-пошук)
  - Алгоритм Р. Боуера й Д. Мура (БМ-пошук)
- Хеш-функції
  - Алгоритм Рабіна-Карпа (РК-пошук)

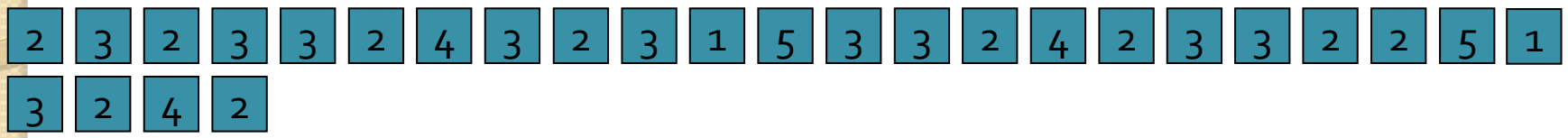
# КМП пошук



# БМ пошук


G C A T C G C A G A G A G T A T A C A G T A C G
1
G C A G A G A G
G C A T C G C A G A G A G T A T A C A G T A C G
3 2 1
G C A G A G A G
G C A T C G C A G A G A G T A T A C A G T A C G
8 7 6 5 4 3 2 1
G C A G A G A G
G C A T C G C A G A G A G T A T A C A G T A C G
3 2 1
G C A G A G A G
G C A T C G C A G A G A G T A T A C A G T A C G
2 1
G C A G A G A G

# РК-пошук




Функція: $\sum_{i=1}^{m} S_i = 11$

Значення функції на підрядках:

10 **11** 10 12 12 **11** 12 9 **11** 12 12 13 12 **11**

Кількість порівнянь символів:

0 + 3 + 0 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 0 + 0 + 4 = 9

# ХЕШ-функції

Один з кращих способів визначити хеш-функцію від рядка S:

$$h(S) = S[0] + S[1] * P + S[2] * P^2 + \ldots + S[3] * P^3 + \ldots + S[N] * P^N$$

де **P** - деяке число, просте,

приблизно дорівнює кількості символів у вхідному алфавіті

(Наприклад, P = 31 або P = 53)