Chars and strings

Jordi Cortadella
Department of Computer Science

Representation of characters (char)

- Character (char). Represent letters, digits, punctuation marks and control characters.

- Every character is represented by a code (integer number). There are various standard codes:
  - American Standard Code for Information Interchange (ASCII)
  - Unicode (wider than ASCII)

- Some characters are grouped by families (uppercase letters, lowercase letters and digits). Characters in a family have consecutive codes: 'a'...'z', 'A'...'Z', '0'...'9'

- Operators: given the integer encoding, arithmetic operators can be used, even though only addition and subtraction make sense, e.g. 'C'+1='D', 'm'+4='q', 'G'-1='F'.

Strings

- Represent sequences of characters.

- Examples
  - "Hello, world!", "This is a string", ":-)", "3.1416"
  - "" is the empty string (no characters)
  - 'A' is a character, "A" is a string

ASCII code
Strings

• **Strings** can be treated as vectors of characters.

• Variables can be declared as follows:
  – `string s1;`
  – `string s2 = "abc";`
  – `string s3(10,'x');`

• Note: use `#include <string>` in the header of a program using strings.

---

String matching

• String x appears as a substring of string y at position i if y[i..i+x.size()-1] = x

• Example: “tree” is the substring of “the tree there” at position 4.

• Problem: given x and y, return the smallest i such that x is the substring of y at position i. Return -1 if x does not appear in y.

---

Strings

• Examples of the operations we can do on strings:

  – Comparisons: `==`, `!=`, `<`, `>`, `<=`, `>=`  
    Order relation assuming lexicographical order.

  – Access to an element of the string: `s3[i]`

  – Length of a string: `s.size()`

---

String matching

• Solution: search for such i

  • For every i, check whether x = y[i..i+x.size()-1]

  • In turn, this is a search for a possible mismatch between x and y: a position j where x[j] ≠ y[i+j]

  • If there is no mismatch, we have found the desired i. As soon as a mismatch is found, we proceed to the next i.
String matching

// Returns the smallest i such that x == y[i..i+x.size()-1].
// Returns -1 if x does not appear in y

int substring(const string& x, const string& y);

String matching

// Inv: x is not a substring of y at positions 0..i-1
for (int i = 0; i <= y.size() - x.size(); ++i) {
    int j = 0;
    // Inv: x[0..j-1] == y[i..i+j-1]
    while (j < x.size() and x[j] == y[i + j]) ++j;
    if (j == x.size()) return i;
}

return -1;

String matching

A more compact solution with only one loop (but less readable).

int substring(const string& x, const string& y) {
    int i = 0, j = 0;
    // Inv: x[0..j-1] == y[i..i+j-1]
    while (i + x.size() <= y.size() and j < x.size()) {
        if (x[j] == y[i + j]) ++j;
        else {
            j = 0;
            ++i;
        }
    }
    if (j == x.size()) return i;
    return -1;
}
Anagrams

• An anagram is a pair of sentences that contain exactly the same letters, even though they may appear in a different order.

• Non-alphabetic characters are ignored.

• Example:

AVE MARIA, GRATIA PLENA, DOMINUS TECUM

VIRGO SERENA, PIA, MUNDA ET IMMACULATA

A possible strategy for solving the problem could be as follows:

• First, we read the first sentence and count the number of occurrences of each letter. The occurrences can be stored in a vector.

• Next, we read the second sentence and discount the appearance of each letter.

• If a counter becomes negative, the sentences are not an anagram.

• At the end, all occurrences must be zero.

bool anagram(const string& s1, const string& s2) {
    const int N = int('z') - int('a') + 1;
    vector<int> count(N, 0);
    // Read the first sentence
    for (int i = 0; i < s1.size(); ++i) {
        char c = s1[i];
        if (c >= 'a' && c <= 'z') ++count[int(c) - int('a')];
        else if (c >= 'A' && c <= 'Z') ++count[int(c) - int('A')];
    }
    // Read the second sentence
    for (int i = 0; i < s2.size(); ++i) {
        char c = s2[i];
        if (c >= 'a' && c <= 'z') c = c - int('a') + int('A');
        if (c >= 'A' && c <= 'Z') { // Discount if it is a letter
            int k = int(c) - int('A');
            --count[k];
            if (count[k] < 0) return false;
        }
    }
    // Check that the two sentences are an anagram
    for (int i = 0; i < N; ++i) {
        if (count[i] != 0) return false;
    }
    return true;
}

Design a function that checks that two strings are an anagram. The function has the following specification:

// Returns true if s1 and s2 are an anagram, // and false otherwise.
bool anagram(const string& s1, const string& s2);
Summary

• Strings can be accessed as arrays of chars.

• String matching (or string searching) is a very frequent operation in file editing and web searching.

• Recommendation: design algorithms that are independent from the specific encoding of chars.