Important Questions of Chapter 3

String Processing

Chapter 3: String Processing

1. What is a string? Or Define String.

A string is a data structure used to store a sequence of characters. Strings often store text-based data, such as words, phrases, and sentences. A string is generally considered a data type. It is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using character encoding.

In C programming, a string is a sequence of characters terminated with a null character '\0'. For example:

```
char c[] = "c string";
```

2. Describe briefly the three types of structure used for storing strings.

A string is a data structure used to store a sequence of characters. Generally, three types of structures are used to store strings which are:

1. Record-Oriented, Fixed Length Storage
2. Variable Length Storage with Fixed Maximum
3. Linked Storage

Fixed Length Storage

Strings are often stored in a fixed-length structure. This is because it is easy to determine the size of the string and, thus, the amount of memory needed for it. This also makes it easy to store the string in a contiguous block of memory, which can be necessary for performance.

When a string is stored in a fixed length structure, the size of the string is known in advance. This makes it easy to allocate the necessary amount of memory for the string. It also makes it easy to store the string in a contiguous block of memory, which can improve performance.

Variable Length Storage

Variable length storage is a type of data storage where the amount of storage space used is based on the amount of data being stored. This is in contrast to fixed-length storage, where a set amount of storage space is allocated regardless of how much data is actually being stored.

Variable length storage is often more efficient, as it only uses the amount of storage space that is necessary.

Linked Storage

Strings are also often stored in a linked list, a data structure that stores a collection of elements linked together. Each element in a linked list is called a node, and each node has a reference to the next node in the list.

A linked list is a good choice for storing a string because it can be easily resized. For example, if a string is stored in an array with a size of 32, and the string grows to 33 characters, the entire array must be copied to a new array with a size of 64. However, if the string is stored in a linked list, only the new node needs to be added to the list.

Read my article for more information: https://www.esakib.com/2022/10/string-storage-data-structure.html

3. Discuss various string operations with examples

String operations are some of the most basic and important operations that can be performed on data structures. Indexing, substring, length, and concatenation are all common string operations that are used to manipulate and analyze string data.

Indexing

Indexing also called pattern matching, refers to finding the position where a string pattern P first appears in a given string text T. We call this operation INDEX and write:

INDEX (text, pattern)

If the pattern P does not appear in the text T, then INDEX is assigned the value 0. The arguments "text" and "pattern" can be either string constant or string variables.

Example: Suppose, T contains the text: 'HELLO WORLD'

Then, INDEX(T, 'WOR') has the value of 7, INDEX(T, 'LLO') has the value of 3, and INDEX(T, 'LOL') has the value of 0 because the 'LOL' pattern is not in the text T.

Substring

The string operation that helps us to find the part of the text or string between the start and end indexes is called the substring operation. To access a substring from a string required 3 information which is: the name of the string or itself, the position of the substring in the string, and the length of the substring or the position of the last character of the substring.

SUBSTRING(string, initial, length)

Example:
SUBSTRING('HELLO WORLD', 4, 6) = 'LO WOR'
SUBSTRING('HELLO WORLD', 3, 5) = 'LLO W'

Concatenation

String concatenation is the operation of joining character strings end-to-end. For example, the concatenation of "show" and "piece" is "showpiece". Let S1 and S2 are two strings and the concatenation of S1 and S2 can be denoted as S1//S2

Example: Suppose S1 = 'SNOW' and S2 = 'BALL', then S1//S2 = 'SNOWBALL' and S1//' '//S2 = 'SNOW BALL'

Length

The number of characters in a string is called its length. In this string operation basically, we find the length of different strings.

Example: Suppose the string S = 'HELLO WORLD', then LENGTH (S) = 11

4. What are the operations associated with word/text processing?

The operations associated with word/text processing are

1. Replacement
2. Insertion
3. Deletion

Replacement: Replacing one string in the text by another.

Suppose in a given text T we want to replace the first occurrence of a pattern P1, by a pattern P2. We will denote this operation by

REPLACE(text, pattern1, pattern2)

Example: REPLACE('XABYABZ', 'AB', 'C')='XCYABZ'

Insertion: Inserting a string in the middle of the text.

Suppose in a given text T we want to insert a string S so that S begins in position K. We denote this operation by

INSERT(text, position, string)

Example: INSERT ('ABCDEFG', 3, 'XYZ') = 'ABXYZCDEFG'

Deletion: Deleting a string from the text.

Suppose in a given text T we want to delete the substring which begins at position K and has length L. We denote this operation by

DELETE(text, position, length)

Example: DELETE ('ABCDEFG', 4, 2) = 'ABCFG'

5. Write an algorithm that deletes every occurrence of pattern P in a test T

A text T and a pattern P are in memory. This algorithm deletes every occurrence of P in T.

Step 1. [Find the index of P.] Set K:= INDEX(T, P).
Step 2. Repeat while K ≠ 0:
(a) [Delete P from T]
Set T:= DELETE(T, INDEX(T, P), LENGTH(P))
(b) [Update index.] Set K:= INDEX(T, P).
[End of loop]
Step 3. Write: T.
Step 4. Exit.

6. Write an algorithm that replaces every occurrence of P in the text T by Q.

A text T and patterns P and Q are in memory. This algorithm replaces every occurrence of P in T by Q.

Step 1. [Find the index of P.] Set K:= INDEX(T, P).
Step 2. Repeat while K ≠ 0:
(a) [Replace P by Q.] Set T:= REPLACE(T, P, Q).
(b) [Update index.] Set K:= INDEX(T, P).
[End of loop.]
Step 3. Write: T.
Step 4. Exit.

7. Write the first pattern matching algorithm.

P and T are strings with lengths R and S, respectively, and are stored as arrays with one character per element. This algorithm finds the INDEX of P in T.

Step 1. [Initialize.] Set K:= 1 and MAX: = S-R+1
Step 2. Repeat Steps 3 to 5 while K ≤ MAX:
Step 3. Repeat for L= 1 to R: [Tests each character of P.]
If P[L] ≠ T [K + L-1], then: Go to Step 5.
[End of the inner loop.]
Step 4. [Success.] Set INDEX = K, and Exit.
Step 5. Set K:= K + 1.
[End of Step 2 outer loop.]
Step 6. [Failure.] Set INDEX = 0.
Step 7. Exit.