Name : Mu In Nasif
Roll : 001910501036
Class : BCSE II
Sem : 3
Session : 2020-2021

Data Structures and Algorithms Assignment Set 2

Question 5:Define an ADT for String. Write C data representation and functions for the operations on the String in a Header file, with array as the base data structure, without using any inbuilt function in C. Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

Solution approach:

We can define strings as sequences of characters, where each character is accessible by its position (an integer); “Hello” may be said to be an example of a string where ‘H’ has position 1, ‘e’ has position 2, position 3 and 4 have ‘l’ (multiple positions may have common characters) and so on. Since strings are very similar (almost congruent) to arrays of characters in this regard, we will implement strings as arrays of characters, where each character has the property that any two characters have some comparison based ordering defined (‘A’ may be defined to be less than ‘B’, etc.). We will also use the sentinel value NIL as a character to indicate the end of a string (NIL is present immediately after the last character of the string). This description is very much identical to how strings are implemented in the standard C library available to programmers, and so we will also define a similar suite of operations paralleling the standard C library:

1. Finding the length/number of characters of a string (excluding NIL) (paralleling strlen)

2. Copying a string from a source array to a destination array (paralleling strcpy)

3. Adding a string immediately after the end of a second string (called concatenation operation, paralleling strcat)

4. Comparison of two strings in lexicographical order as defined by character comparisons (paralleling strcmp). We define equality of strings as strings having same length and same characters at same positions. To find whether a string A is less than string B, we use the following method which is the same as those found in dictionaries: We start from the starting position of both A and B. If at such a position the character at A is less than (comes before in ordering) than the character at B in the same position, then A is less than B. (Example, “axe” will come before “ball” inside a dictionary.) While advancing, if we reach the end of A before we reach the end of B (A has less number of characters than B) then A is less than B. (Example, “as” comes before “assimilate”.)

5. Searching for a character from the starting end of a string (paralleling strchr)

6. Obtaining a substring (part of a string) from a bigger string (paralleling strncpy)

7. Searching for a substring within a string from the starting end; for example “ll” is found in “Hello” from position 3 (paralleling strstr)

8. Reversing the character sequence within the string (many earlier C++ implementations had strrev for this; although now it is absent)

Pseudocode:

```
//Assume all indexes start from 1. Assume each character can be compared as in integers.

string_length(A) → length {
        //Return the length of the string
        i = 1
        while A[i] is not equal to NIL: i = i + 1
        return i – 1 //number of (valid) characters is the number of characters excluding NIL
}

stringCopy(A,B) { // copy A to B
        I = 1
        while S[i] is not equal to NIL {
                D[i] = S[i] //copy valid character
                i = i + 1
        }
        D[i] = NIL //Indicate valid end of string in destination
}

string_concatenation(D, S) { //Copy S to the end of D
        i = 1
        while D[i] is not equal to NIL: i = i + 1 //First go to the end of D
        j = 1
        while S[j] is not equal to NIL {
                 D[i] = S[j] //Copy S to D starting from the end of D
                i = i + 1, j = j + 1
        }
        D[i] = NIL
}
```

```
string_compare(A, B) {//Check whether A comes before B, is equal to B or after B
        i = 1
        while A[i] is not equal to NIL and B[i] is not equal to NIL {
                if A[i] < B[i]: return A is less than B
                elif A[i] > B[i]: return A is greater than B
                        i = i + 1
        }
        if A[i] is not equal to B[i] { //One of the strings is smaller, they did not reach NIL at the
        same position.
                if A[i] is NIL: return A is less than B //because A finished before B
                else return A is greater than B //because B finished earlier than A
        }
        else return A is equal to B //Same characters at same positions and same length
}

string_char_search(A, ch) {//Search for ch inside A and return the index
        i = 1
        while A[i] is not equal to NIL {
                if A[i] equals ch: return I i = i + 1
        }
        return NIL //As we did not find the character.
}

sub_string(A, startindex, endindex) {//Index parameters inclusive (assumed valid), return
substring Create string (array) B with space for (endindex – startindex + 2) characters
        for i = startindex to endindex inclusive: B[i – startindex + 1] = A[i]
        B[endindex - startindex + 2] = NIL //Properly terminate the string return B
}
```

Code:

a2q5_string.h is the implementation for the operations as described above; a2q5_driver.c is a testing program for the same. Both conform to ANSI C89