Shazaib Dawood
09/30/2024
CS 211

Outline

- Various ways to declare a C-string, using square brackets vs. a pointer, distinguishing stack arrays vs. pointers to heap-allocated memory
    - square brackets
        - Array allocated on the stack
        - fixed size
        - char strArr[10]
            - creates an array able to store 9 elements plus a null character
    - pointers
        - Array dynamically allocated on the heap
        - flexible size
        - char* strArr = (char*) malloc(sizeof(char) * 10)
            - dynamically allocates an array able to store 9 elements plus the null character
    - stack arrays vs pointers
        - Stack arrays
            - fixed size
            - memory is freed automatically
        - heap arrays
            - size can be changed
            - memory must be freed manually
- Various ways to initialize a C-string, using { }, scanf( ), fgets( ), strcpy( ), etc.
    - braces
        - manual initialization
        - char strArr[6] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’}
    - scanf
        - reads a string until space
        - scanf(“%s”, strArr)
    - fgets
        - can include new line
        - fgets(strArr, numChars, stdin)
    - strcpy
        - strcpy(strArr, “Hello”)
    - directly
        - char strArr = “Hello”
- Brief descriptions, with examples, for the most important string.h library functions

    - strlen
        - returns the length of the string without the null character
        - int length = strlen(strArr)
    - strcpy
        - copies a string into another string if size is enough
        - copies null character as well
        - strcpy(strArr, “Hello”)
    - strcat
        - appends string 2 to the end of string 1 is space is available
        - strcat(str1, str2)
    - strcmp
        - compares 2 strings character by character and returns 0 if both are the same
        - returns > 0 if str1 is greater than str2 and returns. < 0 if opposite
        - int result = strcmp(str1, str2)
- The importance of the null character '\0' in analyzing and using C-strings
    - marks the end of a string
    - overwriting the null character causes functions to not know where the string ends
    - size of the string must be 1 more than the number of elements to account for the null character
- An example of a well-commented string.h library function definition; e.g. write the strlen() function yourself

```
int strlen(const char* strArr) {

    int length = 0; //initalize length to 0

    while(strArr[length] != ‘\0’) { //loop through all chars until null char is found

        length++;//increment length

    }

    return length;//return length

    }
```