Computer Architecture Sec: 06

Travis Thiel

Programming Assignment 1: A Partial Tokenizer

This program accepts a string, a character array, through the terminal and saves it as a parameter in the main function. An initial token is then created in dynamic memory and the string that is passed through main is copied into the token object's character array. If the string passed through main is null and no string was actually passed through the program will exit and print an error message to the user. A while loop then runs through the newly created token once and breaks up, "tokenizes" it using TKGetNextToken(). After each token is printed to the user the space allocated for it is then freed. Once the while loop ends the space allocated for the original token and its character array is freed using TKDestroy(). Main then returns 0 and the program exits.

The original token is created by TKCreate() by using a while loop and then a for loop. I could have used the function strcpy() to copy the string passed to main into the token but I decided to use a combination of a while loop and then a for loop to remove extraneous space characters. The while loop increments through the string until it finds a non-space character. This is in order to remove the white space at the beginning. The for loop does the actual copying, going through the rest of the string and skipping any duplicate space characters. This runs through the given string one time when it is created. I chose this method over strcpy() because it is more efficient. Both strcpy() and my TKCreate() goes through the give string once, but TKCreate() shortens that string by removing unnecessary space characters allowing TKGetNextToken() to be called less. In the worst case where there is no spaces TKCreate() will run identical to strcpy() with an extra if statement in the loop, but for the majority of the time it will improve performance.

The actual "tokenizing" is done using TKGetNextToken() which utilizes a do while loop to switch between a total of eight separate states. These states keep track of whether the token is a zero, octal, hexadecimal, Integer, float, malformed, or and escape character. The space for each token is allocated dynamically and each state either copies the current character into the token and continues the while loop or exits the loop when the next character is either malformed or does not fit in the current state. The space for the token is then reallocated to better fit the new size which is more likely than not smaller than the original string passed, though there are cases when the string passed is a single token. This function also utilizes three integer variables to keep track of whether or not a '.', a 'e' or 'E', or a '+' or '-'are already present in the current token. These variables are reset when the do while loop is exited and then the token is returned.

**The state that checks for escape characters works for escape characters within a single index of a character array such as '\a', but I could not figure out how to enter an escape character through the terminal. When inputted through the terminal an escape character is split into two characters in their own indices. **