Hijjawi Faculty for Engineering Technology
Computer Engineering Department
System Programming Lab (Cpe 466)
Lab 2: Simple Shell

Goals:

- Create a simple shell
- Build your tokenizer.

Background:

- A shell is a way to interact with a computer system using a set of standard commands. Shell commands are two kinds: internal and external. Internal commands are built into and part of the shell itself (the shell know how to execute). External commands are stand alone executable commands that the shell executes upon user request. The ability for the shell to execute external commands gives users the ability to scale up and execute commands that are new (e.g. users' own compiled programs).

Practical:

- Write a C program that tokenize a string then save the output token in array of strings and print them on the screen.
- Write a C program "MyShell" that execute the following command using the c-library function system(char * cmd).

1. clear: clears the screen.
2. exit: quits the shell.
3. listen: lists all environment strings.
4. pwd: prints the current working directory.

Good Luck

Help for you

- Libraries may be needed in your task:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

- Some Functions that you may need in your task:

  - `int  scanf ( const char * format specifier, type variable,...);`

Reads data from `stdin` and stores them according to the parameter *format* into the locations pointed by the additional arguments. The additional arguments should point to already allocated objects of the type specified by their corresponding format tag within the *format* string.

  - `int printf ( const char * format, type variable,... );`

Writes to the standard output (stdout) a sequence of data formatted as the *format* argument specifies. After the *format* parameter, the function expects at least as many additional arguments as specified in *format*.

  - `char * gets ( char * str );`

:

Reads characters from `stdin` and stores them as a string into *str* until a newline character (`'\n'`) or the End-of-File is reached.
The ending newline character (`'\n'`) is not included in the string.
A null character (`'\0'`) is automatically appended after the last character copied to *str* to signal the end of the C string.

  - `char *strtok( char *str1, const char *str2 );`

The strtok() function returns a pointer to the next "token" in str1, where str2 contains the delimiters that determine the token. strtok() returns NULL if no token is found. In order to convert a string to tokens, the first call to strtok() should have str1 point to the string to be tokenized. All calls after this should have str1 be NULL.

Example:

```
char str[] = "now # is the time for all # good men to come to
the # aid of their country";
char delims[] = "#";
char *result = NULL;
result = strtok( str, delims );
while( result != NULL ) {
printf( "result is \"%s\"\n", result );
```

```
        result = strtok( NULL, delims );



 OUTPUT:
  result is "now "
  result is " is the time for all "
  result is " good men to come to the "
  result is " aid of their country"


}
```

- `char *strcpy(char *destination, const char *source);`

Copies the C string pointed by *source* into the array pointed by *destination*, including the terminating null character.

- `int strcmp ( const char * str1, const char * str2 );`

Compares the C string *str1* to the C string *str2*.
This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached.

- `void *malloc(size_t size);`

The function malloc() returns a pointer to a chunk of memory of size size, or NULL if there is an error. The memory pointed to will be on the heap, not the stack, so make sure to free it when you are done with it.

Example

```
int *ptr = (int *) malloc(10 * sizeof (int));
```

- `int system(const char *cmd);`

This function runs the command or program specified by *cmd if it is internal command with respect to the shell.*