Course	60-256	November 1, 2014
Instructor	Dr. B. Boufama	
Assignment	05	
Due date	November 13, 9pm (**to be emailed to sood111@uwindsor.ca**)	

The C program below, given as example in class, behaves like a small shell. It processes single commands entered by the user. In particular, the program assembles commands and execute them, either in the background or foreground. The commands/programs location can be anywhere in $**PATH** and might have arguments.

Modify this solution to make the program capable of processing a sequence of commands/programs separated by ';'.

E.g., if the input-line entered by the user is

ls -F; pwd; google-chrome&; date

your program should assemble these 4 commands and execute them in sequence.

The example to be modified:

```
Algorithm:
While(1)
begin
  get command line from user
  assemble command args
  duplicate current process
  child should exec to the new program
  if (background execution) then
    parent process accept without waiting for its child to terminate
  else
    parent process waits for its child to terminate
end         // You can download this porgram from
```

http://boufama.myweb.cs.uwindsor.ca/256/examples/chap5/chap5_ex_3.c

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

int readArgs(char *, char *[]);
int main(int argc, char *argv[]){
  pid_t pid;
  char line[255], *argList[20];
  int ampersand, status, i;
```

```
  printf("This program will execute commands/programs for you\n");
  while(1){
    printf("To exit, enter CTR-C, or\n");
    printf("enter a program name with its arguments> ");
    fgets(line, 255, stdin);
    ampersand=readArgs(line, argList);
    if((pid = fork()) == -1){
      perror("impossible to fork");
      exit(1);
    }
    if(pid > 0)  // This is parent
      if(ampersand){
        waitpid(pid, &status, WNOHANG);
        printf("Process [%d]\n", pid);
      }
      else{
        waitpid(pid, &status, 0);
        printf("My child has terminated\n");
      }
    else                                  // this is the child
      if(execvp(argList[0], argList)==-1){
        perror("child Process");
        exit(22);
      }
  }
  exit(0);
}
int readArgs(char *line, char *argList[]){
  static int yes=0;
  int i=0, offset=0, found=0;
  char name[50];

  while(yes & argList[i] != NULL)
    free(argList[i++]);

  i=0;
  while(sscanf(line+offset, "%s", name)==1){
    argList[i] = (char *) malloc(strlen(name)+1);
    strcpy(argList[i++], name);

    while(line[offset] == ' ')  // skip extra blanks
      offset++;
```

```
    offset += strlen(name);
  }
  if(!strcmp(argList[i-1], "&")){
    argList[i-1] = NULL;
    found = 1;
  }else{
    if(argList[i-1][strlen(argList[i-1])-1] == '&'){
      argList[i-1][strlen(argList[i-1])-1]='\0';
      found = 1;
    }
    argList[i] = NULL;
  }
  yes=1;
  return(found);
}
```