```
/* DO NOT REMOVE THIS COMMENT: CSE 2431 Lab 1 AU 25 Code 080311 */

/* ******   PUT YOUR NAME HERE:Yuhan Zhou          ******** */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>


#define MAXLINE 40 /* 40 chars per line, that is, per command, is limit for this
shell. */

/**
setup() reads in the next command line string stored in the input buffer.
separating it into distinct tokens using whitespace as delimiters.  Setup
modifies the args parameter so that it holds pointers to the null-terminated
strings  which are the tokens in the most recent user command line as well
as a NULL pointer, indicating the end of the argument list, which comes
after the string pointers that have been assigned to args.
**/

void setup(char inputBuff[], char *args[],int *background)
{
    int length,  /* Num characters in the command line */
        i,       /* Index for inputBuff arrray          */
        j,       /* Where to place the next parameter into args[] */
        start;   /* Beginning of next command parameter */

    /* Read what the user enters */
    length = read(STDIN_FILENO, inputBuff, MAXLINE);

    j = 0;
    start = -1;

    if (length == 0)
        exit(0);            /* Cntrl-d was entered, end of user command stream */

    if (length < 0){
        perror("error reading command");
      exit(-1);           /* Terminate with error code of -1 */
    }

    /* Examine every character in the input buffer */
    for (i = 0; i < length; i++) {

        switch (inputBuff[i]){
        case ' ':
        case '\t' :          /* Argument separators */

            if(start != -1){
                args[j] = &inputBuff[start];    /* Set up pointer */
                j++;
            }

            inputBuff[i] = '\0'; /* Add a null char; make a C string */
            start = -1;
            break;
```

```
        case '\n':             /* Final char examined */
            if (start != -1){
                args[j] = &inputBuff[start];
                j++;
            }

            inputBuff[i] = '\0';
            args[j] = NULL; /* No more arguments to this command */
            break;

        case '&':
            *background = 1;
            inputBuff[i] = '\0';
            break;

        default :             /* Some other character */
            if (start == -1)
                start = i;
      }

    }
    args[j] = NULL; /* Just in case the input line was > 40 characters */
}

int main(void)
{
    char inputBuff[MAXLINE]; /* Input buffer  to hold the command entered */
    char *args[MAXLINE/2+1]; /* Command line arguments */
    int background;          /* Equals 1 if a command is followed by '&', else 0 */

    pid_t ret_val;         /* to hold value returned by fork */
    while (1){              /* Program terminates normally inside setup if
appropriate */

      background = 0;

      printf("CSE2431Shell$ ");  /* print shell prompt */
        fflush(0);

        setup(inputBuff, args, &background);       /* Get command user just entered
*/

      /*        Fill in the code for the steps below this comment:
       (1) Fork a child process using fork(),
       (2) Write code to check for three possible cases (use ret_val):
           a) if ERROR (child process not created); print error message and call
exit(0);
           b) else the child process will execute the command by invoking execvp(),
           c) else the parent will:
              if background == 0, wait for the child before continuing to execute;
            otherwise parent returns to top of loop to print the prompt
            and call the setup() function.
      */

      /* CODE FOR STEPS DESCRIBED ABOVE SHOULD GO BELOW TO IMPLEMENT SHELL */
    /* Fork a child process */
    ret_val = fork();
    if(ret_val < 0){
        perror("fork error");
```

```
        exit(0);
    }

    if(ret_val == 0){
        /*Child process. Execute the given command.*/
        if(execvp(args[0], args) < 0){
        /*If execvp fails,report it and exit*/
            perror("execvp error");
            exit(0);
        }
    }
    else{
        /*This is the parent proces*/
        if(background == 0){
        /*I think background is like a flag. if it's 0, then it will wait for the
child to finish*/
            waitpid(ret_val, NULL, 0);
        }
    }

    }
    return (0);
}
```