CS 3113 Project 1A

Roll Your Own Shell (25 points)

Write a small program that loops reading a line from standard input and checks the first word of the input line. If the first word is one of the following internal commands (or aliases) perform the designated task. Otherwise, use the standard ANSI C system function to execute the line through the default system shell.

Internal Commands/Aliases

- `clr` — clear the screen using the system function clear: `system("clear")`.
- `dir <directory>` — list the current directory contents (`ls -al <directory>`) - you will need to provide some command line parsing capability to extract the target directory for listing . Once you have built the replacement command line, use the system function to execute it.
- `environ` — list all the environment strings - the environment strings can be accessed from within a program by specifying the POSIX compliant environment list:
  `extern char **environ;`
  as a global variable. environ is an array of pointers to the environment strings terminated with a NULL pointer. (see environ.c below for examples of use)
- `quit` — quit from the program with a zero return value. Use the standard exit function.

External Commands

For all other command line inputs, relay the command line to the parent shell for execution using the system function.

When parsing the command line you may have to explicitly or implicitly malloc (strdup) storage for a copy of the command line. Ensure that you free any malloced memory after it is no longer needed. You may find strtok useful for parsing.

The C Standard Library has a number of other string related functions that you may find useful (string.h) contains to descriptions of the other main "string" functions). Use the glibc documentation for more information on string functions.

The source of the basis for a simple shell using strtok and system is contained in strtokeg.c below.

Note the number, type and style of comments in strtokeg.c - this is the level of commentry expected of the code you hand in for your projects.

Code should be in 'straight' C using the latest gcc compiler.

Always use nice to execute your test programs at lower priority to ensure they do not inconvenience other users if they go 'haywire' e.g. `nice ./1a` . Below is an example run:

```Bash
my-instance$ make
my-instance$ ./1a
==>environ
TERM_PROGRAM=Google_Terminal
SHELL=/bin/bash
TERM=xterm-256color
LANG=en_US.UTF-8
HOME=/Users/christangrant
LOGNAME=christangrant
_=./1a
OLDPWD=/Users/christangrant
==>dir
total 40
drwxr-xr-x  5 cgrant  cgrant   170 Sep 29 22:42 .
drwxr-xr-x 17 cgrant  cgrant   578 Sep 28 10:16 ..
-rw-r--r--  1 cgrant  cgrant   368 Dec  9  2003 makefile
-rwxr-xr-x  1 cgrant  cgrant  9112 Aug 15 14:07 1a
-rw-r--r--  1 cgrant  cgrant  3757 Dec  9  2003 1a.c
==>
```

Requirements

For this project, create f1-micro virtual machine on your google cloud instance. Choose the `us-central1-a` zone. Be sure to select an external ip and to allow all project keys. Use the latest version of Ubuntu. It is okay to use the same instance from the previous project.

Add the ssh public key for the `cs3113fa17` user. The key is available here: http://cs.ou.edu/~cgrant/cs3113fa17.pub. To add your key, click on the your instances. Then, select SSH Keys > Edit. Select add item, Enter the Key above, exactly verbatim (there should be no newlines within the key text). Finally, be sure to click save.

Run the start up script available here to start your project https://www.cs.ou.edu/~cgrant/cs3113fa17.sh.

The code for this project should go in the `/projects/1a/`
Create a `makefile` that will compile all necessary files and create a `1a.out` executable file.

Grading Criteria:

Task	Percent
Instance is reachable	10%
Code compiles with `make`	10%
Each function works correctly	80%
Total	**100%**

You will submit only your external ip address for grading.

Extra Info

atexit

```c
int atexit(void (*fcm)(void));
```

Registers fcn to be called when program terminates normally (or when main returns). Returns non-zero on failure.

exit

```c
void exit(int status);
```

Terminates program normally. Functions installed using atexit are called (in reverse order to that in which installed), open files are flushed, open streams are closed and control is returned to environment. *status* is returned to environment in implementation-dependent manner. Zero or EXIT_SUCCESS indicates successful termination and EXIT_FAILURE indicates unsuccessful termination. Implementations may define other values.

malloc

```c
void* malloc(size_t size);
```

Returns pointer to uninitialised newly-allocated space for an object of size *size*, or NULL on error.

strdup

```c
char* strdup(const char *s);
```

Returns a pointer to a new string that is a duplicate of the string pointed to by `s`. The space for the new string is obtained using `malloc`. If the new string cannot be created, a null pointer is returned.

strtok

```c
char* strtok(char* s, const char* ct);
```

Searches `s` for next token delimited by any character from `ct`. Non-NULL `s` indicates the first call of a sequence. If a token is found, it is NULL-terminated and returned, otherwise `NULL` is returned. `ct` need not be identical for each call in a sequence. **Beware!** This call does not `malloc` space for the returned string - `strtok` returns a pointer to the string in the original buffer `s`, and replaces the delimiting character with NULL.

system

```c
int system(const char* s);
```

If `s` is not `NULL`, passes `s` to the default command shell for execution, and returns status reported by the command processor when the command is complete; if `s` is NULL, non-zero returned if environment has a command processor (UNIX systems will always have one - by default it will use `sh`).

environ.c

C

```c
/*
  environ - skeleton program displaying environment

  usage:

    environ

    displays environment with each name, value pair on a separate line
*/

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

extern char **environ;                    // environment array

int main(int argc, char **argv)
{
    char ** env = environ;

    while (*env) printf("%s\n",*env++);  // step through environment

    exit(0);
}
```

strtokeg.c

C

```c
/*
  strtokeg - skeleton shell using strtok to parse command line

  usage:

    ./a.out

    reads in a line of keyboard input at a time, parsing it into
    tokens that are separated by white spaces (set by #define
    SEPARATORS).

    can use redirected input

    if the first token is a recognized internal command, then that
    command is executed. otherwise the tokens are printed on the
    display.
```

```
    internal commands:

        clear - clears the screen

        quit - exits from the program

 */

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

#define MAX_BUFFER 1024                         // max line buffer
#define MAX_ARGS 64                             // max # args
#define SEPARATORS " \t\n"                      // token sparators

int main (int argc, char ** argv)
{
    char buf[MAX_BUFFER];                       // line buffer
    char * args[MAX_ARGS];                      // pointers to arg strings
    char ** arg;                                // working pointer thru args
    char * prompt = "==>" ;                     // shell prompt

/* keep reading input until "quit" command or eof of redirected input */

    while (!feof(stdin)) {

/* get command line from input */

        fputs (prompt, stdout);                 // write prompt
        if (fgets (buf, MAX_BUFFER, stdin )) { // read a line

/* tokenize the input into args array */

            arg = args;
            *arg++ = strtok(buf,SEPARATORS);   // tokenize input
            while ((*arg++ = strtok(NULL,SEPARATORS)));
                                               // last entry will be NULL

            if (args[0]) {                     // if there's anything there

/* check for internal/external command */

                if (!strcmp(args[0],"clear")) { // "clear" command
```

```
                    system("clear");
                    continue;
                }

                if (!strcmp(args[0],"quit"))   // "quit" command
                    break;                     // break out of 'while' loop

/* else pass command onto OS (or in this instance, print them out) */

                arg = args;
                while (*arg) fprintf(stdout,"%s ",*arg++);
                fputs ("\n", stdout);
            }
        }
    }
    return 0;
}
```