```c
/*******************************************************************************//**

  @file         main.c

  @author       Stephen Brennan

  @date         Thursday,  8 January 2015

  @brief        LSH (Libstephen SHell)

*******************************************************************************/

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

/*
  Function Declarations for builtin shell commands:
 */
int lsh_cd(char **args);
int lsh_help(char **args);
int lsh_exit(char **args);
int lsh_pwd(char **args);
int lsh_echo(char **args);
int lsh_whoami(char **args);
/*
  List of builtin commands, followed by their corresponding functions.
 */
char *builtin_str[] = {
  "cd",
  "help",
  "exit",
  "pwd",
  "echo",
  "whoami"
};

int (*builtin_func[]) (char **) = {
  &lsh_cd,
  &lsh_help,
  &lsh_exit,
  &lsh_pwd,
  &lsh_echo,
  &lsh_whoami
};

int lsh_num_builtins() {
  return sizeof(builtin_str) / sizeof(char *);
}

/*
  Builtin function implementations.
*/

/**
   @brief Builtin command: change directory.
   @param args List of args.  args[0] is "cd".  args[1] is the directory.
   @return Always returns 1, to continue executing.
 */
int lsh_cd(char **args)
{
  if (args[1] == NULL) {
    fprintf(stderr, "lsh: expected argument to \"cd\"\n");
  } else {
    if (chdir(args[1]) != 0) {
      perror("lsh");
    }
  }
  return 1;
}
```

```
74
75  /**
76     @brief Builtin command: print help.
77     @param args List of args.  Not examined.
78     @return Always returns 1, to continue executing.
79   */
80  int lsh_help(char **args)
81  {
82    int i;
83    printf("Stephen Brennan's LSH\n");
84    printf("Type program names and arguments, and hit enter.\n");
85    printf("The following are built in:\n");
86
87    for (i = 0; i < lsh_num_builtins(); i++) {
88      printf("  %s\n", builtin_str[i]);
89    }
90
91    printf("Use the man command for information on other programs.\n");
92    return 1;
93  }
94
95  /**
96     @brief Builtin command: exit.
97     @param args List of args.  Not examined.
98     @return Always returns 0, to terminate execution.
99   */
100 int lsh_exit(char **args)
101 {
102   return 0;
103 }
104
105 /**
106    my builtin command pwd
107  */
108
109 int lsh_pwd(char **args)
110 {
111   char cwd[1024];
112   if (getcwd(cwd, sizeof(cwd)) != NULL) {
113     printf("Current working directory is this: %s\n", cwd);
114   } else {
115       perror("getcwd() error");
116       return 1;
117   }
118   return 1;
119 }
120
121 /**
122    my builtin command echo
123  */
124
125 int lsh_echo(char **args)
126 {
127   printf("This is lsh_echo: \n");
128   int i = 1;
129   while(args[i] != NULL){
130     printf("%s ", args[i]);
131     i++;
132   }
133   printf("\n");
134   return 1;
135 }
136
137 /**
138    my builtin command whoami
139  */
140
141 int lsh_whoami(char **args)
142 {
143   printf("You are %s\n", getlogin());
144   return 1;
145 }
146
```

```c
/**
  @brief Launch a program and wait for it to terminate.
  @param args Null terminated list of arguments (including program).
  @return Always returns 1, to continue execution.
 */
int lsh_launch(char **args)
{
  pid_t pid;
  int status;

  pid = fork();
  if (pid == 0) {
    // Child process
    if (execvp(args[0], args) == -1) {
      perror("lsh");
    }
    exit(EXIT_FAILURE);
  } else if (pid < 0) {
    // Error forking
    perror("lsh");
  } else {
    // Parent process
    do {
      waitpid(pid, &status, WUNTRACED);
    } while (!WIFEXITED(status) && !WIFSIGNALED(status));
  }

  return 1;
}

/**
   @brief Execute shell built-in or launch program.
   @param args Null terminated list of arguments.
   @return 1 if the shell should continue running, 0 if it should terminate
 */
int lsh_execute(char **args)
{
  int i;

  if (args[0] == NULL) {
    // An empty command was entered.
    return 1;
  }

  for (i = 0; i < lsh_num_builtins(); i++) {
    if (strcmp(args[0], builtin_str[i]) == 0) {
      return (*builtin_func[i])(args);
    }
  }

  return lsh_launch(args);
}

/**
   @brief Read a line of input from stdin.
   @return The line from stdin.
 */
char *lsh_read_line(void)
{
#ifdef LSH_USE_STD_GETLINE
  char *line = NULL;
  ssize_t bufsize = 0; // have getline allocate a buffer for us
  if (getline(&line, &bufsize, stdin) == -1) {
    if (feof(stdin)) {
      exit(EXIT_SUCCESS);  // We received an EOF
    } else  {
      perror("lsh: getline\n");
      exit(EXIT_FAILURE);
    }
  }
  return line;
#else
#define LSH_RL_BUFSIZE 1024
```

```
220     int bufsize = LSH_RL_BUFSIZE;
221     int position = 0;
222     char *buffer = malloc(sizeof(char) * bufsize);
223     int c;
224
225     if (!buffer) {
226       fprintf(stderr, "lsh: allocation error\n");
227       exit(EXIT_FAILURE);
228     }
229
230     while (1) {
231       // Read a character
232       c = getchar();
233
234       if (c == EOF) {
235         exit(EXIT_SUCCESS);
236       } else if (c == '\n') {
237         buffer[position] = '\0';
238         return buffer;
239       } else {
240         buffer[position] = c;
241       }
242       position++;
243
244       // If we have exceeded the buffer, reallocate.
245       if (position >= bufsize) {
246         bufsize += LSH_RL_BUFSIZE;
247         buffer = realloc(buffer, bufsize);
248         if (!buffer) {
249           fprintf(stderr, "lsh: allocation error\n");
250           exit(EXIT_FAILURE);
251         }
252       }
253     }
254   #endif
255   }
256
257   #define LSH_TOK_BUFSIZE 64
258   #define LSH_TOK_DELIM " \t\r\n\a"
259   /**
260      @brief Split a line into tokens (very naively).
261      @param line The line.
262      @return Null-terminated array of tokens.
263    */
264   char **lsh_split_line(char *line)
265   {
266     int bufsize = LSH_TOK_BUFSIZE, position = 0;
267     char **tokens = malloc(bufsize * sizeof(char*));
268     char *token, **tokens_backup;
269
270     if (!tokens) {
271       fprintf(stderr, "lsh: allocation error\n");
272       exit(EXIT_FAILURE);
273     }
274
275     token = strtok(line, LSH_TOK_DELIM);
276     while (token != NULL) {
277       tokens[position] = token;
278       position++;
279
280       if (position >= bufsize) {
281         bufsize += LSH_TOK_BUFSIZE;
282         tokens_backup = tokens;
283         tokens = realloc(tokens, bufsize * sizeof(char*));
284         if (!tokens) {
285           free(tokens_backup);
286           fprintf(stderr, "lsh: allocation error\n");
287           exit(EXIT_FAILURE);
288         }
289       }
290
291       token = strtok(NULL, LSH_TOK_DELIM);
292     }
```

```
293     tokens[position] = NULL;
294     return tokens;
295   }
296
297   /**
298      @brief Loop getting input and executing it.
299    */
300   void lsh_loop(void)
301   {
302     char *line;
303     char **args;
304     int status;
305
306     do {
307       printf("lsh_project>>> ");
308       line = lsh_read_line();
309       args = lsh_split_line(line);
310       status = lsh_execute(args);
311
312       free(line);
313       free(args);
314     } while (status);
315   }
316
317   /**
318      @brief Main entry point.
319      @param argc Argument count.
320      @param argv Argument vector.
321      @return status code
322    */
323   int main(int argc, char **argv)
324   {
325     // Load config files, if any.
326
327     // Run command loop.
328     lsh_loop();
329
330     // Perform any shutdown/cleanup.
331
332     return EXIT_SUCCESS;
333   }
334
335
```