```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <ctype.h>
#include <stdarg.h>
#include "colors.h"

#define MAX_ARGS 100    // Maximum number of arguments
#define MAX_ARG_LEN 256 // Maximum length of each argument
#define DEBUG 0 // Change to 1 to verbose debug prints

static void
die(const char *fmt, ...)
{
        va_list ap;
        va_start(ap, fmt);
        vfprintf(stderr, fmt, ap);
        va_end(ap);
        if (fmt[0] != '\0' && fmt[strlen(fmt) - 1] == ':') {
                fputc(' ', stderr);
                perror(NULL);
        } else {
                fputc('\n', stderr);
        }
        exit(EXIT_FAILURE);
}

static void *
ecalloc(size_t nmemb, size_t size)
{
        void *p;
        if ((p = calloc(nmemb, size)) == NULL)
                die("calloc:");
        return p;
}

static void *
erealloc(void *p, size_t len)
{
        if ((p = realloc(p, len)) == NULL)
                die("realloc: %s\n", strerror(errno));
        return p;
}

int inArray(const char *str, const char *arr[], int size) {
  if (str == NULL) {
    return 0;
  }
  for (int i=0; i<size; i++) {
    if (strcmp(str, arr[i]) == 0) {
      return 1; // Found
    }
  }
  return 0; // Not Found
}

char* inPath(const char *command) {

  char* path = getenv("PATH");
  char *path_copy = strndup(path, strlen(path));
  const char split[2] = ":";
  struct stat file_stat;

  char *token;
  token = strtok(path_copy, split);

  while (token != NULL) {
    char *fullpath = malloc(100);
    snprintf(fullpath, 100, "%s/%s", token, command);

    if ( (stat(fullpath, &file_stat) == 0) && (file_stat.st_mode & S_IXOTH )) {
      free(path_copy);
      return fullpath;
    }
    token = strtok(NULL, split);
    free(fullpath);
  }
  free(path_copy);
  return NULL;
}

void trim(char *s) {
    // Two pointers initially at the beginning
    int i = 0, j = 0;

    // Skip leading spaces
    while (s[i] == ' ') i++;

    // Shift the characters of string to remove
    // leading spaces
    while (s[j++] = s[i++]);
}

int tokenize(char **tokens, char *str) {
    char separator[2] = " ";
    char* next_token;
    tokens[0] = strtok(str, separator);
    next_token = strtok(0, separator);
    int i=1;
    while (next_token != 0) {
      tokens[i] = malloc(strlen(next_token)+1);
      strcpy(tokens[i], next_token);
      next_token = strtok(0, separator);
      i++;
    }
    return i;
}

static char *util_cat(char *dest, char *end, const char *str)
{
    while (dest < end && *str)
        *dest++ = *str++;
    return dest;
}

size_t join_str(char *out_string, size_t out_bufsz, const char *delim, char **chararr)
{
    char *ptr = out_string;
    char *strend = out_string + out_bufsz;
    while (ptr < strend && *chararr)
    {
        ptr = util_cat(ptr, strend, *chararr);
        chararr++;
        if (*chararr)
            ptr = util_cat(ptr, strend, delim);
    }
    return ptr - out_string;
}
void parse_line(const char *line, char *command_output, char **args_output, int *arg_count) {
    const char *ptr = line; // Pointer to traverse the input string
    char *arg = NULL;
    int index = 0;

    // Skip leading spaces
    while (*ptr && isspace((unsigned char)*ptr)) {
        ptr++;
    }

    // Extract the command (first token)
    while (*ptr && !isspace((unsigned char)*ptr)) {
        *command_output++ = *ptr++;
    }
    *command_output = '\0'; // Null-terminate the command string

    // Parse the arguments
    while (*ptr) {
        // Skip leading spaces
        while (*ptr && isspace((unsigned char)*ptr)) {
            ptr++;
        }

        if (*ptr == '\0') {
            break; // End of string
        }

        if (*ptr == '\'') { // Handle quoted argument
            ptr++; // Skip the opening single quote
            arg = malloc(MAX_ARG_LEN);
            int arg_pos = 0;

            while (*ptr && *ptr != '\'') {
                if (arg_pos < MAX_ARG_LEN - 1) {
                    arg[arg_pos++] = *ptr++;
                } else {
                    fprintf(stderr, "Argument exceeds maximum length\n");
                    free(arg);
                    return;
                }
            }

            if (*ptr == '\'') {
                ptr++; // Skip the closing single quote
            }
            arg[arg_pos] = '\0';
        } else { // Handle unquoted argument
            arg = malloc(MAX_ARG_LEN);
            int arg_pos = 0;

            while (*ptr && !isspace((unsigned char)*ptr)) {
                if (arg_pos < MAX_ARG_LEN - 1) {
                    arg[arg_pos++] = *ptr++;
                } else {
                    fprintf(stderr, "Argument exceeds maximum length\n");
                    free(arg);
                    return;
                }
            }
            arg[arg_pos] = '\0';
        }

        // Add the argument to the output array
        if (index < MAX_ARGS) {
            args_output[index++] = arg;
        } else {
            fprintf(stderr, "Too many arguments\n");
            free(arg);
            return;
        }
    }

    *arg_count = index; // Update the argument count
}


int main() {
  // Flush after every printf
  setbuf(stdout, NULL);

  const char *commands[] = {"exit", "echo", "type", "pwd", "cd"};
  int commands_size = sizeof(commands) / sizeof(commands[0]);

  char exit_command[]="exit";
  char echo_command[]="echo";
  char type_command[]="type";
  char pwd_command[]="pwd";
  char cd_command[]="cd";

  char s[2] = " ";
  char quote[2] = "'";

  int running = 1;
  while (running) {
    printf(BHGRN "$ " CRESET);

    // Wait for user input
    char input[100];
    printf(CYN);
    fgets(input, 100, stdin);
    printf(CRESET);

    input[strcspn(input, "\n")] = '\0';

    char **argv = (char**)malloc(5*sizeof(char*));
    char *args = ecalloc(8, 128);
    char *args_quotes = ecalloc(8, 128);
    int argc = 0;
    char cmd[256];

    parse_line(input, cmd, argv, &argc);

    if (argc > 0) {
      join_str(args, 1024, s, argv);

      for (int i=0; i<argc; i++){
        strcat(args_quotes, "'");
        strcat(args_quotes, argv[i] );
        strcat(args_quotes, "'");
        strcat(args_quotes, " ");
      }
    }
    else {
      args = NULL;
      args_quotes = NULL;
    }

    if (DEBUG) {
      for (int i=0; i<argc; i++) {
        printf("argv %d: %s\n", i, argv[i]);
      }
      printf("cmd: %s\nargc: %d\nargs: %s\nargsq: %s\n---\n", cmd, argc, args,args_quotes);
      }
    if (cmd == NULL) { continue; }

    // EXIT COMMAND
    if (strcmp(cmd, exit_command) == 0) {
      running=0;
      break;
    }

    // ECHO COMMAND
    else if (strcmp(cmd, echo_command) == 0) {
      if (args != NULL) {
      printf("%s\n", args);}
    }

    // TYPE COMMAND
    else if (strcmp(cmd, type_command) == 0) {
      if (argc == 0) {printf("You need to specify a command\n");
      }
      else {

        if (inArray(argv[0], commands, commands_size) == 1) {
           printf("%s is a shell builtin\n", argv[0]);
           }

        else if (inPath(argv[0]) != NULL) {
           char* fp;
           fp = inPath(argv[0]);
           printf("%s is %s\n", argv[0], fp);
           free(fp);
      }
        else { printf("%s: not found\n", argv[0]);}
      }   }

    // PWD COMMAND
    else if (strcmp(cmd, pwd_command) == 0) {
      char cwd[1024];
      getcwd(cwd, sizeof(cwd));
      printf("%s\n",cwd);
    }

    else if (strcmp(cmd, cd_command) == 0) {
      if (argc == 0) {printf("You need to specify a directory\n");
      }
      else {
        if (!strcmp(argv[0], "~")) {
          strcpy(args, getenv("HOME"));
        }

        int result = chdir(args);
        if ((result != 0) && (errno == ENOENT)) {
           printf("cd: %s: No such file or directory\n", args);
        }
      }
    }

    // EXECUTE COMMAND IN PATH
    else if ( inPath(cmd) != NULL ) {

      char* fullCommand = ecalloc(1, 100);
      if (args_quotes != NULL) {
        snprintf(fullCommand, 100, "%s %s\n",cmd, args_quotes);
      }
      else {
        snprintf(fullCommand, 100, "%s\n",cmd);
      }
      int returnCode = system(fullCommand);
      free(fullCommand);
    }

    else {
      printf("%s: command not found\n", input);
    }

    free(argv);
    free(args);
  }
  return 0;
}
```