# buffalo: a terminal text editor

By Micah Cantor

## What goes into a text editor?

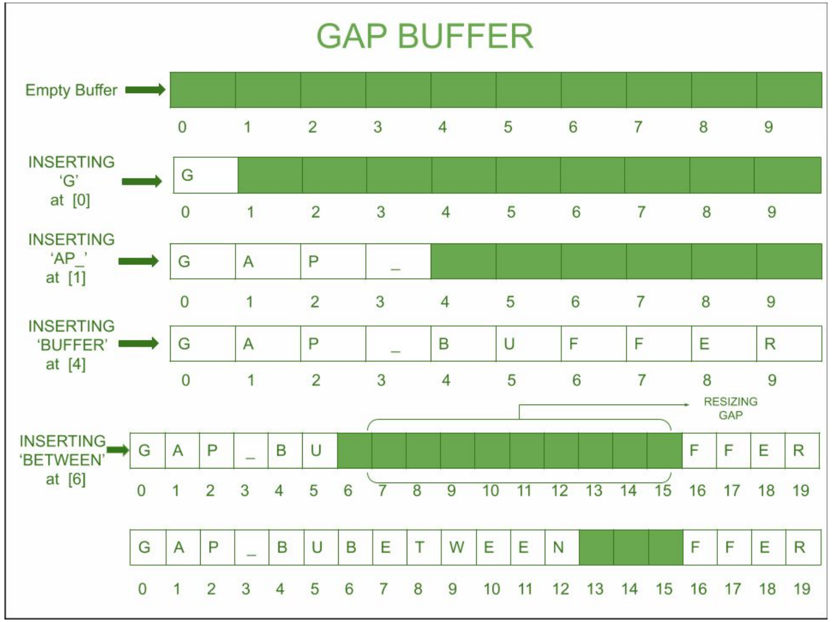
Reading, editing, and saving text files

Support for running build and test commands within the editor

Configuration with a .buffalorc file

## Demo

## Gap buffers: an elegant solution



Source: GeeksForGeeks.com

Efficient insertion/deletion at cursor

Fast for common use cases

Used by emacs

**Problem**: Hard to manage line breaks

## Living on the edge

There’s a lot of edge cases with editing lines of text:

- Splitting/adding lines with enter
- Deleting lines
- Moving cursor between lines
- Moving cursor at start/end of document
- Accounting for scroll position

All require precise accounting of the cursor

gap_buffer_t
me
I don't want to play with you anymore
imgflip.com

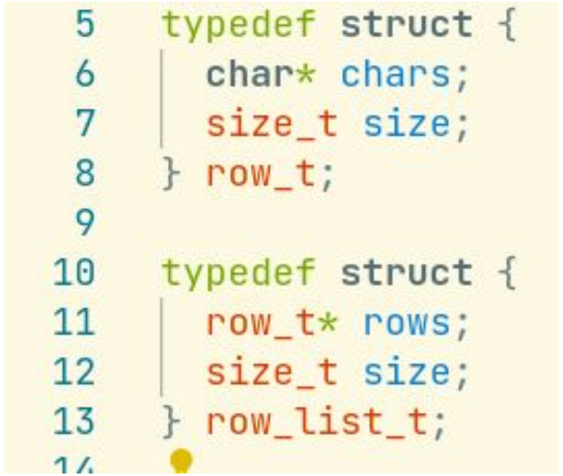
# A more straightforward data structure

Dynamic array of dynamic arrays.

Easier to manage edge cases at line boundaries (of which there are a lot)

Computers are fast ¯_(ツ)_/¯

*We should forget about small efficiencies, say about 97% of the time:* ***premature optimization is the root of all evil.*** - Donald Knuth



```c
typedef struct {
    char* chars;
    size_t size;
} row_t;

typedef struct {
    row_t* rows;
    size_t size;
} row_list_t;
```

## Forms and UI

Initially used ncurses forms from <form.h> to display text

Forms come with scrolling and line management built in

**Problem**: They don’t work with the cursor:

- Can’t jump to a cursor position in the form.
- Cursor position doesn’t update correctly on scroll

<form.h>
me
I don't want to play with you anymore
imgflip.com

## The main loop

Runs in a separate thread

Clears and re-renders the entire screen after each input

Expensive, but makes scrolling and editing simple

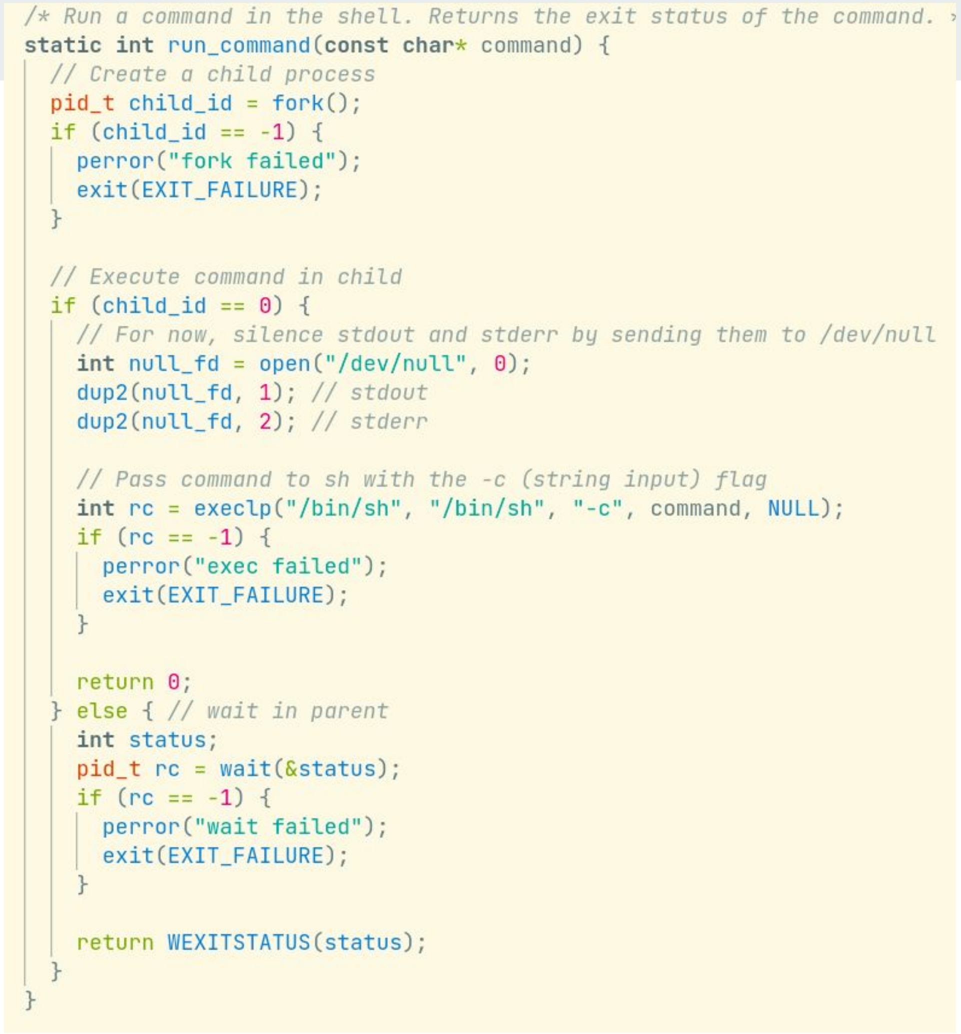
```c
/* Run the main UI loop. Intended to be called by pthread_create. */
void* ui_run(void* arg) {
  // Cast arg to buffalo state pointer
  buffalo_state_t* bs = (buffalo_state_t*)arg;

  // Loop as long as the program is running
  while (bs->running) {
    // Get a character
    int ch = getch();

    // If there was no character, try again
    if (ch == -1) continue;

    if (ch == CTRL('Q')) { // quit
      quit(bs);
    } else if (ch == CTRL('S')) { // save
      save(bs);
    } else if (ch == CTRL('B')) { // build
      build(bs);
    } else if (ch == CTRL('T')) { // test
      test(bs);
    } else if (ch == '\x1b') { // escape to start arrow key
      arrow_key(bs);
    } else { // edit
      edit(ch, bs);
    }

    clear_editor();
    display_editor(bs);

    // Move back to current row/col
    move(bs->row + HEADER_HEIGHT + 1 - bs->scroll_offset, bs->col);
  }

  return NULL;
}
```

# In-editor commands

Support running configured build and test commands from the editor (Ctrl+B and Ctrl+T)

Spawn a subprocess, execute the command in sh, wait, display result

Sends stdout and stdin to /dev/null to silence them.

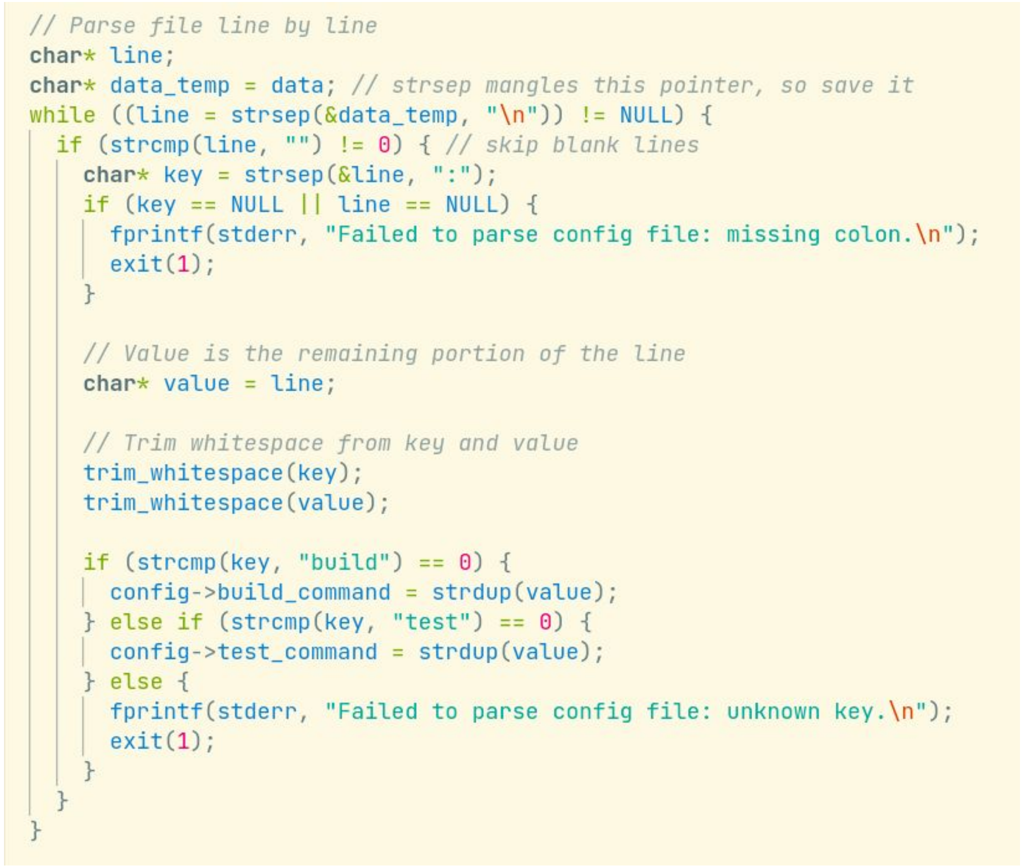


```c
/* Run a command in the shell. Returns the exit status of the command.
static int run_command(const char* command) {
  // Create a child process
  pid_t child_id = fork();
  if (child_id == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
  }

  // Execute command in child
  if (child_id == 0) {
    // For now, silence stdout and stderr by sending them to /dev/null
    int null_fd = open("/dev/null", 0);
    dup2(null_fd, 1); // stdout
    dup2(null_fd, 2); // stderr

    // Pass command to sh with the -c (string input) flag
    int rc = execlp("/bin/sh", "/bin/sh", "-c", command, NULL);
    if (rc == -1) {
      perror("exec failed");
      exit(EXIT_FAILURE);
    }

    return 0;
  } else { // wait in parent
    int status;
    pid_t rc = wait(&status);
    if (rc == -1) {
      perror("wait failed");
      exit(EXIT_FAILURE);
    }

    return WEXITSTATUS(status);
  }
}
```

# Configuration

Editor can be configured in a .buffalorc file in current or home directory

JSON or YAML would be best, but uses bespoke format for simplicity



```c
// Parse file line by line
char* line;
char* data_temp = data; // strsep mangles this pointer, so save it
while ((line = strsep(&data_temp, "\n")) != NULL) {
  if (strcmp(line, "") != 0) { // skip blank lines
    char* key = strsep(&line, ":");
    if (key == NULL || line == NULL) {
      fprintf(stderr, "Failed to parse config file: missing colon.\n");
      exit(1);
    }

    // Value is the remaining portion of the line
    char* value = line;

    // Trim whitespace from key and value
    trim_whitespace(key);
    trim_whitespace(value);

    if (strcmp(key, "build") == 0) {
      config->build_command = strdup(value);
    } else if (strcmp(key, "test") == 0) {
      config->test_command = strdup(value);
    } else {
      fprintf(stderr, "Failed to parse config file: unknown key.\n");
      exit(1);
    }
  }
}
```

## Future steps

Tons of avenues for future improvement:

- Optimize rendering and data structures for efficiency
- Standard editing features like undo/redo and copy/paste
- Find and replace
- Unicode support
- Extended configuration
- Syntax highlighting
- … much more