# Fundamental Exercise on Computer and Information Engineering 1B
# Schedule

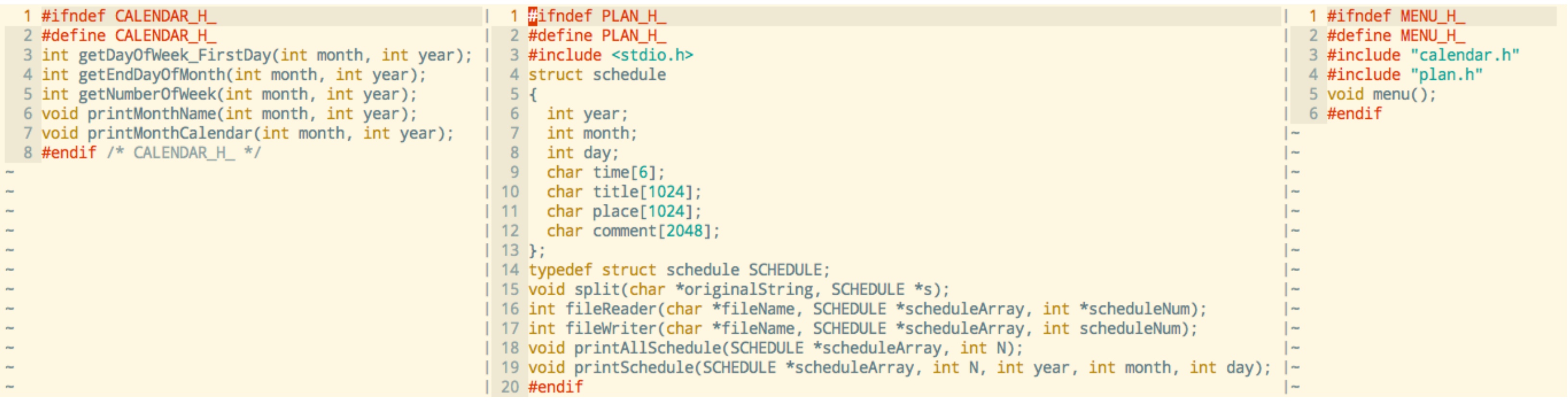
XL15613 Thiago Machado da Silva

June 4, 2015

## Source codes

### Header files

Refer to Figure 1.
Each `.c` file that don't have a `main` function has an `.h` header file with the same name. They usually have `ifdef`, `define` and `endif` to make sure each header is accounted only once. Also external functions have their prototypes in header files.



```c
#ifndef CALENDAR_H_
#define CALENDAR_H_
int getDayOfWeek_FirstDay(int month, int year);
int getEndDayOfMonth(int month, int year);
int getNumberOfWeek(int month, int year);
void printMonthName(int month, int year);
void printMonthCalendar(int month, int year);
#endif /* CALENDAR_H_ */
```

```c
#ifndef PLAN_H_
#define PLAN_H_
#include <stdio.h>
struct schedule
{
  int year;
  int month;
  int day;
  char time[6];
  char title[1024];
  char place[1024];
  char comment[2048];
};
typedef struct schedule SCHEDULE;
void split(char *originalString, SCHEDULE *s);
int fileReader(char *fileName, SCHEDULE *scheduleArray, int *scheduleNum);
int fileWriter(char *fileName, SCHEDULE *scheduleArray, int scheduleNum);
void printAllSchedule(SCHEDULE *scheduleArray, int N);
void printSchedule(SCHEDULE *scheduleArray, int N, int year, int month, int day);
#endif
```

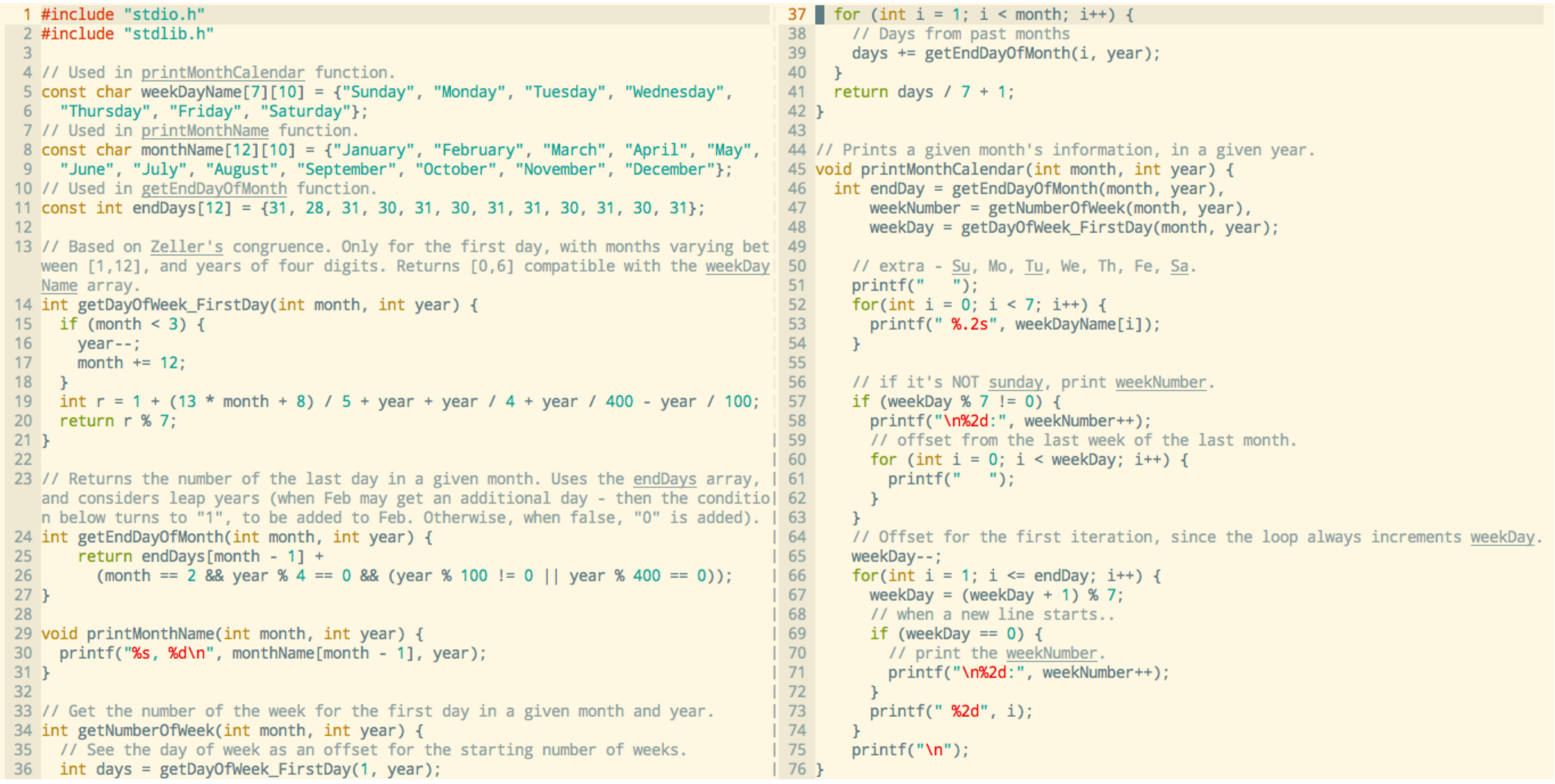
```c
#ifndef MENU_H_
#define MENU_H_
#include "calendar.h"
#include "plan.h"
void menu();
#endif
```

Figure 1: from left to right: `calendar.h`, `plan.h` and `menu.h`.

### Calendar

Refer to Figure 2.
Some constant variables were created, mostly for similar data usage.

There is one bad practice where a `bool` value is used as an `int` value, in `getEndDayIfMonth` function, resulting in a small size function.
There is an extra printing in `printMonthCalendar` function, where the week day name is also printed.

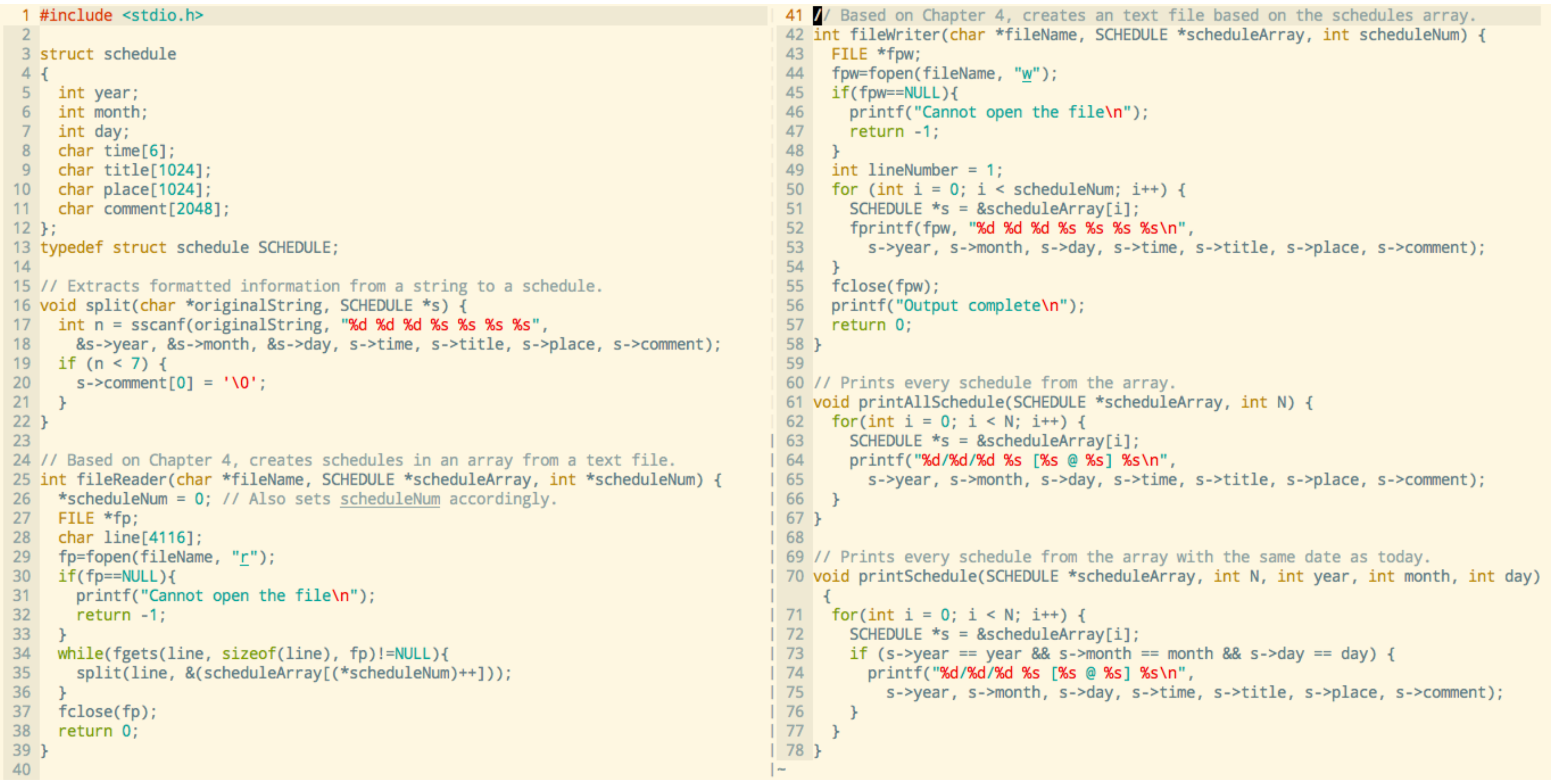


```c
#include "stdio.h"
#include "stdlib.h"

// Used in printMonthCalendar function.
const char weekDayName[7][10] = {"Sunday", "Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday"};
// Used in printMonthName function.
const char monthName[12][10] = {"January", "February", "March", "April", "May",
  "June", "July", "August", "September", "October", "November", "December"};
// Used in getEndDayOfMonth function.
const int endDays[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

// Based on Zeller's congruence. Only for the first day, with months varying between [1,12], and years of four digits. Returns [0,6] compatible with the weekDayName array.
int getDayOfWeek_FirstDay(int month, int year) {
  if (month < 3) {
    year--;
    month += 12;
  }
  int r = 1 + (13 * month + 8) / 5 + year + year / 4 + year / 400 - year / 100;
  return r % 7;
}

// Returns the number of the last day in a given month. Uses the endDays array, and considers leap years (when Feb may get an additional day - then the condition below turns to "1", to be added to Feb. Otherwise, when false, "0" is added).
int getEndDayOfMonth(int month, int year) {
    return endDays[month - 1] +
      (month == 2 && year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
}

void printMonthName(int month, int year) {
  printf("%s, %d\n", monthName[month - 1], year);
}

// Get the number of the week for the first day in a given month and year.
int getNumberOfWeek(int month, int year) {
  // See the day of week as an offset for the starting number of weeks.
  int days = getDayOfWeek_FirstDay(1, year);
  for (int i = 1; i < month; i++) {
    // Days from past months
    days += getEndDayOfMonth(i, year);
  }
  return days / 7 + 1;
}

// Prints a given month's information, in a given year.
void printMonthCalendar(int month, int year) {
  int endDay = getEndDayOfMonth(month, year),
      weekNumber = getNumberOfWeek(month, year),
      weekDay = getDayOfWeek_FirstDay(month, year);

    // extra - Su, Mo, Tu, We, Th, Fe, Sa.
    printf("   ");
    for(int i = 0; i < 7; i++) {
      printf(" %.2s", weekDayName[i]);
    }

    // if it's NOT sunday, print weekNumber.
    if (weekDay % 7 != 0) {
      printf("\n%2d:", weekNumber++);
      // offset from the last week of the last month.
      for (int i = 0; i < weekDay; i++) {
        printf("   ");
      }
    }
    // Offset for the first iteration, since the loop always increments weekDay.
    weekDay--;
    for(int i = 1; i <= endDay; i++) {
      weekDay = (weekDay + 1) % 7;
      // when a new line starts..
      if (weekDay == 0) {
        // print the weekNumber.
        printf("\n%2d:", weekNumber++);
      }
      printf(" %2d", i);
    }
    printf("\n");
}
```

Figure 2: `calendar.c`.

## Plan

Refer to Figure 3.
In this file, there aren't a lot of data copying since pointers are heavily used, regarding the function's parameters initialization.
In `split` function, the comment input is intended to be optional. If no commentary is detected, the `comment` char vector is formated to be an "empty string".



```c
#include <stdio.h>

struct schedule
{
  int year;
  int month;
  int day;
  char time[6];
  char title[1024];
  char place[1024];
  char comment[2048];
};
typedef struct schedule SCHEDULE;

// Extracts formatted information from a string to a schedule.
void split(char *originalString, SCHEDULE *s) {
  int n = sscanf(originalString, "%d %d %d %s %s %s %s",
    &s->year, &s->month, &s->day, s->time, s->title, s->place, s->comment);
  if (n < 7) {
    s->comment[0] = '\0';
  }
}

// Based on Chapter 4, creates schedules in an array from a text file.
int fileReader(char *fileName, SCHEDULE *scheduleArray, int *scheduleNum) {
  *scheduleNum = 0; // Also sets scheduleNum accordingly.
  FILE *fp;
  char line[4116];
  fp=fopen(fileName, "r");
  if(fp==NULL){
    printf("Cannot open the file\n");
    return -1;
  }
  while(fgets(line, sizeof(line), fp)!=NULL){
    split(line, &(scheduleArray[(*scheduleNum)++]));
  }
  fclose(fp);
  return 0;
}

// Based on Chapter 4, creates an text file based on the schedules array.
int fileWriter(char *fileName, SCHEDULE *scheduleArray, int scheduleNum) {
  FILE *fpw;
  fpw=fopen(fileName, "w");
  if(fpw==NULL){
    printf("Cannot open the file\n");
    return -1;
  }
  int lineNumber = 1;
  for (int i = 0; i < scheduleNum; i++) {
    SCHEDULE *s = &scheduleArray[i];
    fprintf(fpw, "%d %d %d %s %s %s %s\n",
      s->year, s->month, s->day, s->time, s->title, s->place, s->comment);
  }
  fclose(fpw);
  printf("Output complete\n");
  return 0;
}

// Prints every schedule from the array.
void printAllSchedule(SCHEDULE *scheduleArray, int N) {
  for(int i = 0; i < N; i++) {
    SCHEDULE *s = &scheduleArray[i];
    printf("%d/%d/%d %s [%s @ %s] %s\n",
      s->year, s->month, s->day, s->time, s->title, s->place, s->comment);
  }
}

// Prints every schedule from the array with the same date as today.
void printSchedule(SCHEDULE *scheduleArray, int N, int year, int month, int day)
 {
  for(int i = 0; i < N; i++) {
    SCHEDULE *s = &scheduleArray[i];
    if (s->year == year && s->month == month && s->day == day) {
      printf("%d/%d/%d %s [%s @ %s] %s\n",
        s->year, s->month, s->day, s->time, s->title, s->place, s->comment);
    }
  }
}
```

Figure 3: `plan.c`.

## Menu

Refer to Figure 4.
Some constant data are declared in the top of the code.
A `valid` function is implemented. It returns `1` for valid and `0` for invalid, even though I think it's a good practice to use `true` for valid and `false` for invalid.
In `menu` function, There'd be less memory de/allocation if I declared `s` (`SCHEDULE*` type) outside of the `for` loop. But this variable lifetime is enclosed in a small scope, where it is actually used, and I consider this an advantage.

## calImpl3

Refer to Figure 5.
The commands show, from the left pane to the right pane: Compilation, testing the existence of `testSchedule.txt`, then `calImpl3` execution. Then in the

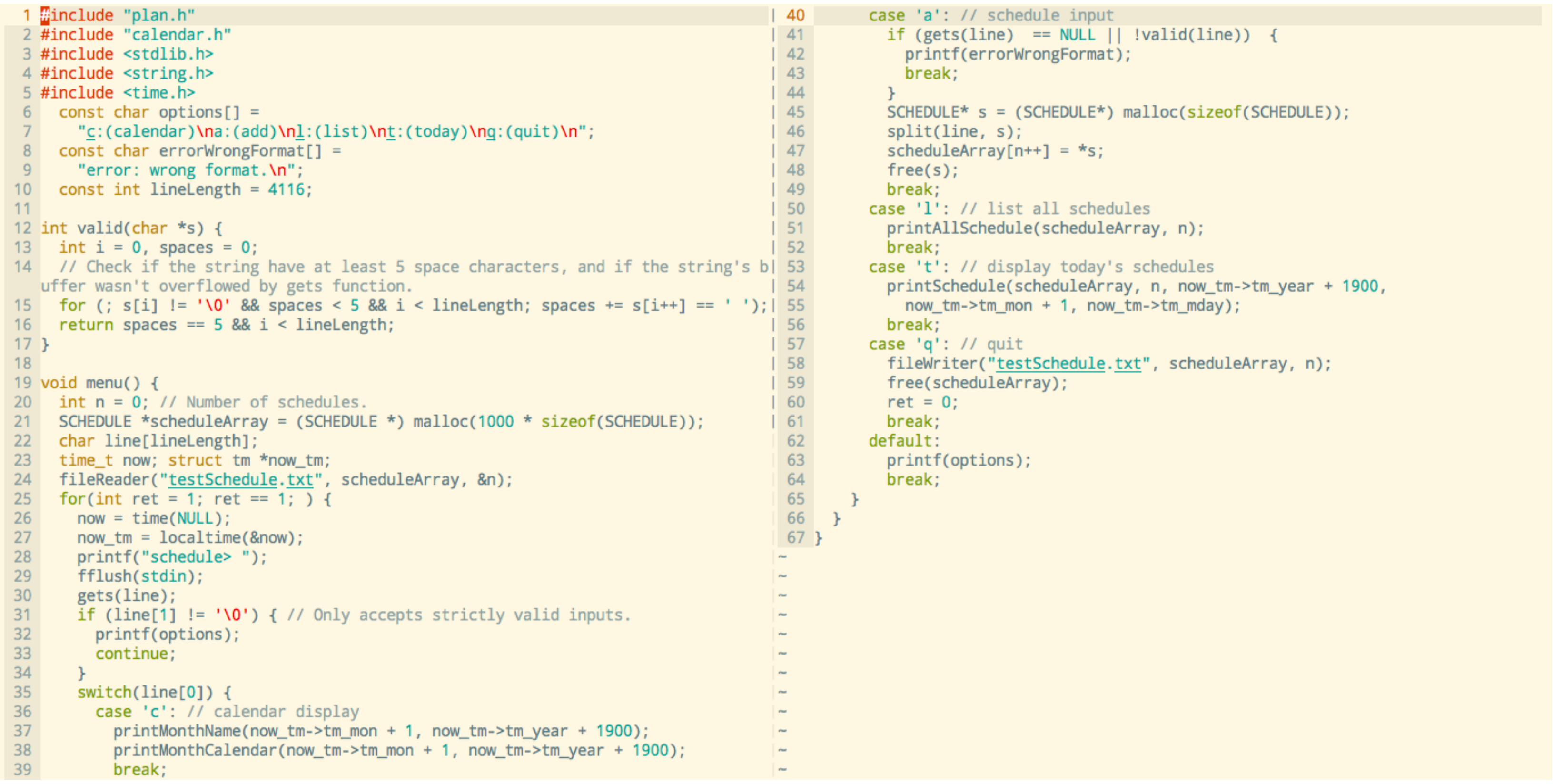
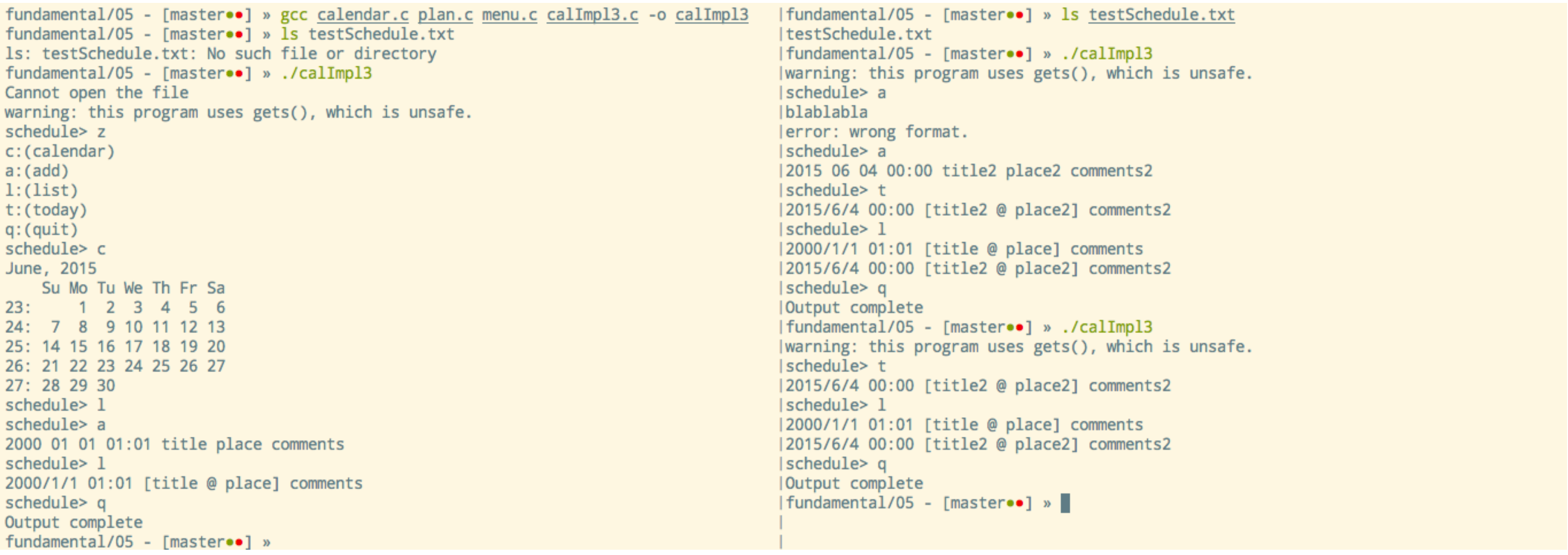

```c
#include "plan.h"
#include "calendar.h"
#include <stdlib.h>
#include <string.h>
#include <time.h>
  const char options[] =
    "c:(calendar)\na:(add)\nl:(list)\nt:(today)\nq:(quit)\n";
  const char errorWrongFormat[] =
    "error: wrong format.\n";
  const int lineLength = 4116;

int valid(char *s) {
  int i = 0, spaces = 0;
  // Check if the string have at least 5 space characters, and if the string's buffer wasn't overflowed by gets function.
  for (; s[i] != '\0' && spaces < 5 && i < lineLength; spaces += s[i++] == ' ');
  return spaces == 5 && i < lineLength;
}

void menu() {
  int n = 0; // Number of schedules.
  SCHEDULE *scheduleArray = (SCHEDULE *) malloc(1000 * sizeof(SCHEDULE));
  char line[lineLength];
  time_t now; struct tm *now_tm;
  fileReader("testSchedule.txt", scheduleArray, &n);
  for(int ret = 1; ret == 1; ) {
    now = time(NULL);
    now_tm = localtime(&now);
    printf("schedule> ");
    fflush(stdin);
    gets(line);
    if (line[1] != '\0') { // Only accepts strictly valid inputs.
      printf(options);
      continue;
    }
    switch(line[0]) {
      case 'c': // calendar display
        printMonthName(now_tm->tm_mon + 1, now_tm->tm_year + 1900);
        printMonthCalendar(now_tm->tm_mon + 1, now_tm->tm_year + 1900);
        break;
      case 'a': // schedule input
        if (gets(line)  == NULL || !valid(line))  {
          printf(errorWrongFormat);
          break;
        }
        SCHEDULE* s = (SCHEDULE*) malloc(sizeof(SCHEDULE));
        split(line, s);
        scheduleArray[n++] = *s;
        free(s);
        break;
      case 'l': // list all schedules
        printAllSchedule(scheduleArray, n);
        break;
      case 't': // display today's schedules
        printSchedule(scheduleArray, n, now_tm->tm_year + 1900,
          now_tm->tm_mon + 1, now_tm->tm_mday);
        break;
      case 'q': // quit
        fileWriter("testSchedule.txt", scheduleArray, n);
        free(scheduleArray);
        ret = 0;
        break;
      default:
        printf(options);
        break;
    }
  }
}
```

Figure 4: `menu.c`.

right pane, another `testSchedule.txt` existence test, then two more `calImpl3` executions.



```
fundamental/05 - [master●●] » gcc calendar.c plan.c menu.c calImpl3.c -o calImpl3
fundamental/05 - [master●●] » ls testSchedule.txt
ls: testSchedule.txt: No such file or directory
fundamental/05 - [master●●] » ./calImpl3
Cannot open the file
warning: this program uses gets(), which is unsafe.
schedule> z
c:(calendar)
a:(add)
l:(list)
t:(today)
q:(quit)
schedule> c
June, 2015
    Su Mo Tu We Th Fr Sa
23:     1  2  3  4  5  6
24:  7  8  9 10 11 12 13
25: 14 15 16 17 18 19 20
26: 21 22 23 24 25 26 27
27: 28 29 30
schedule> l
schedule> a
2000 01 01 01:01 title place comments
schedule> l
2000/1/1 01:01 [title @ place] comments
schedule> q
Output complete
fundamental/05 - [master●●] »
```

```
fundamental/05 - [master●●] » ls testSchedule.txt
testSchedule.txt
fundamental/05 - [master●●] » ./calImpl3
warning: this program uses gets(), which is unsafe.
schedule> a
blablabla
error: wrong format.
schedule> a
2015 06 04 00:00 title2 place2 comments2
schedule> t
2015/6/4 00:00 [title2 @ place2] comments2
schedule> l
2000/1/1 01:01 [title @ place] comments
2015/6/4 00:00 [title2 @ place2] comments2
schedule> q
Output complete
fundamental/05 - [master●●] » ./calImpl3
warning: this program uses gets(), which is unsafe.
schedule> t
2015/6/4 00:00 [title2 @ place2] comments2
schedule> l
2000/1/1 01:01 [title @ place] comments
2015/6/4 00:00 [title2 @ place2] comments2
schedule> q
Output complete
fundamental/05 - [master●●] » █
```

Figure 5: Commands related to `calImpl3` (compilation and some tests). First, the left pane were executed, then the right pane.