Software Engineering SENG 265
Software Development Methods
Fall 2021

Assignment 1

Due: 6 October @ 23:55 — submission via Brightspace
(no late submissions accepted due to 200+ students in this course)

Programming environment

For this assignment you must ensure your work executes correctly on the virtual machine you installed as part of Assignment #0 subsequently referred to as *Senjhalla*. This is our “reference platform”. This same environment will also be used by the teaching team when evaluating submitted work from students.

All test files and sample code for this assignment are available on the SENG server in /seng265work/A1/ and you must use scp in a manner similar to what happens in labs in order to copy these files into your *Senjhalla.*

Any programming done outside of *Senjhalla* might not work during evaluation, which will result in lost marks or even 0 marks for the assignment.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you may want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden.** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact the course instructor. Note Code-similarity analysis tools are used to examine submitted work.

Objectives of this assignment

- Understand a problem description, along with the role of the sample input and output for provided.
- Use the programming language C to write the first implementation of a file processor named *process_cal.c* without using dynamic memory.
- Leverage Unix commands, such as diff, in your coding and testing.
- Use git to manage changes in your source code and annotate the evolution of your solution with “messages” given to commits.
- Test your code against the provided test cases.

This assignment: *process_cal.c*

In this assignment you will learn C by solving a problem involving file formats used by several calendar programs, such as “iCal” — what is exported by Google Calendar. These files usually have a filename suffix of “ics”. All assignments this term will build on each other such that by the end you will have written *process_cal* in both C and Python several times. To simplify your solution, the iCal files provided for this term will be less complex than what is possible using the full iCalendar standard.

You are to write a C program that inputs lines of data from a calendar file provided, accepts options and arguments from the command line, and then outputs to the console events from the calendar file into a more readable form. To get started, the teaching team has provided a skeleton version of *process_cal.c***,** plus another different C program with a few functions that may be helpful for your implementation. For example, on the SENG server at UVic (*seng265.seng.uvic.ca*) there is one such *ics* file contained in the */seng265work/A1* subdirectory (named *one.ics*) which is an extract from the schedule of a fictional UVic student named Diana Devps:

```
BEGIN:VCALENDAR
BEGIN:VEVENT
DTSTART:20210214T180000
DTEND:20210214T210000
LOCATION:Burger King
SUMMARY:Romantic dinner with Chris
END:VEVENT
END:VCALENDAR
```

This particular file contains information for a fictional single event taking place on September 26, 2021. Suppose you type the following arguments into your program (after having copied the .ics files from the SENG server as directed at the start of this assignment description):

```
./process_cal --start=2021/2/14 --end=2021/2/14 --file=one.ics
```

then the output to be produced is as follows:

```
February 14, 2021 (Sun)
-----------------------
 6:00 PM to  9:00 PM: Romantic dinner with Chris {{Burger king}
```

Note that the arguments passed to the program were two dates (year/month/day) plus a filename. The output reflects that the script printed out all events within the *iCal* file occurring during the specified range of dates (which, in this case, ends up being a single event).

Another possibility is for events to be **repeating**. For example, an event such as regular coffee appointments with a relative is represented as the single entry in the file shown below. (In order to simplify your implementation of ***process_cal***, we will use only *iCal* weekly frequencies for a single day in a week rather than more general repetition possibilities that can be expressed using the *iCalendar* standard.) This repetition is represented with use of the repeating-event rule, as seen in another provided test file name *many.ics*. Consider the contents of this file, followed by the use of *process_cal* to produce a readable version of events within the file:

```
BEGIN:VCALENDAR
VERSION:A
BEGIN:VEVENT
DTSTART:20210102T111500
DTEND: 20210102T123000
RRULE:FREQ=WEEKLY;WKST=MO;UNTIL=20211002T235959;BYDAY=SA
LOCATION:The Bumptious Barista
SUMMARY:Coffee with Pat
END:VEVENT
END:VCALENDAR
```

```
./process_cal --start=2021/2/1 --end=2021/3/1 --file=many.ics
```

```
September 11, 2021 (Sat)
-----------------------
11:15 AM to 12:30 PM: Coffee with Pat {{The Bumptious Barista}}

September 18, 2021 (Sat)
-----------------------
11:15 AM to 12:30 PM: Coffee with Pat {{The Bumptious Barista}}

September 25, 2021 (Sat)
-----------------------
11:15 AM to 12:30 PM: Coffee with Pat {{The Bumptious Barista}}

October 2, 2021 (Sat)
-----------------------
11:15 AM to 12:30 PM: Coffee with Pat {{The Bumptious Barista}}
```

Note that each line in an ics file has a similar structure:

```
<property>:<value>
```

That is, a property's name and the property's value are separated by a single colon. **In this assignment your program need only pay attention to the following properties:**

- BEGIN
- END
- DTSTART
- DTEND
- RRULE
- LOCATION
- SUMMARY

At the end of this document is a more detailed specification for the input files your program must accept, and properties expected in the output. However, given that such specifications always have some elements of ambiguity, the test-output files (i.e., all those provided to you beginning with the letters *"test"*) can be considered as using the required output format. The *TESTS.md* markdown file describes each of the ten tests, with corresponding test outputs listed as *test01.txt, test02.txt,* etc.

In order check for correctness, please use the UNIX command called *diff.* For example, assuming that *diana-devops.ics* is in the same directory as your compiled program, you can compare your output against what is expected for the ninth test using the command shown below (and which assumes the *test09.txt* file is in the same directory as *diana-devops.ics*):

```
./process_cal --start=2021/2/1 --end=2021/3/1 --file=diana-devops.ics  | diff test09.txt -
```

The ending dash as an argument to *diff* will compare *test09.txt* with the text stream piped into the *diff* command. If no output is produced by *diff*, then the output of your program identically matches the file (i.e., the ninth test passes). Note that the arguments you must pass to *process_cal* for each of the tests is shown within *TESTS.md* (i.e., in */seng265work/A1*) on the UVic SENG server).

Please use diff. Your output must exactly match the expected test output in order for a test to pass. Do not attempt to visually check all test cases – the eye is often willing to deceive the brain, especially with respect to the presence (or absence) of horizontal and vertical spaces. diff will fail if there is extra or missing white space. In this case, the test is considered a failed test.

Exercises for this assignment

1. Write your program. Amongst other tasks you will need to:
   - obtain a filename argument from the command line
   - read text input from a file, line by line, and the text within those lines
   - construct some representation of not only the events in the file, but what is needed to print out the events in the range of dates
   - You should use the -std=c99 flag when compiling your program as this will be used during assignment evaluation (i.e., the flag ensures the 1999 C standard is used during compilation).
2. **DO NOT use malloc(), calloc() or any of the dynamic memory functions.** For this assignment you can assume that the longest input line will have 80 characters, and the total number of events output generated by a *from/to/testfile* combination (including repeats of events) will never exceed 500.
3. Keep all of your code in one file for this assignment (that is, *process_cal.c*). In later assignments we will use the separable compilation facility available in C.
4. Use the test files to guide your implementation effort. Start with the simple example in test 01 and move onto 02, 03, etc. in order. (You may want to avoid tests 9 and 10 until you have significant functionality already completed.) **Refrain from writing the program all at once, and budget time to anticipate when things go wrong!** Use the Unix command *diff* to compare your output with what is expected.
5. For this assignment you can assume all test inputs will be well-formed (i.e., the teaching team will not evaluate your submission for handling of input or for arguments containing errors). Later assignments might specify error-handling as part of their requirements.
6. Use git when working on your assignment.
7. Remember, that the ONLY acceptable method of submission is through Brightspace.

What you must submit

- A single C source file named **process_cal.c**, submitted to the Assignment 1 page **in Brightspace**. Brightspace is the **only** acceptable way of submission.
- Note that Brightspace will allow multiple submissions of an assignment. However, only the latest submission will be retained and earlier submissions will be deleted. It is your responsibility to ensure that the correct file is submitted.
- **No dynamic memory-allocation routines are to be used for Assignment 1.**

Evaluation

Assignment 1 grading scheme is as follows.

- **A grade:** A submission completing the requirements of the assignment which is well-structured and clearly written. Global variables are not used. *process_cal* runs without any problems; that is, all tests pass and therefore no extraneous output is produced. **B grade**: A submission completing the requirements of the assignment. *process_cal* runs without any problems; that is, all tests pass and therefore no extraneous output is produced. The program is clearly written.
- **C grade**: A submission completing most of the requirements of the assignment. *process_cal* runs with some problems.
- **D grade**: A serious attempt at completing requirements for the assignment. *process_cal* runs with quite a few problems.
- **F grade**: Either no submission given, or submission represents little work or none of the tests pass.process_cal, version 1

This program will be written in C. Its name must be "process_cal.c" and it must be found in Brightspace under submissions for Assignment 1.

Input specification

1. All input is from ASCII-data test files.
2. Data lines for an "event" begin with a line "BEGIN:VEVENT" and end with a line "END:VEVENT".
3. Starting time: An event's starting date and time is contained on a line of the format "DTSTART:<icalendardate>" where the characters following the colon comprise the date/time in icalendar format.
4. Ending time: An event's ending date and time is contained on a line of the format "DTEND:<icalendardate>" where the characters following the colon comprise the date/time in icalendar format.
5. Event location: An event's location is contained on a line of the format "LOCATION:<string>" where the characters following the colon comprise the string describing the event location. These strings will never contain the ":" character.
6. Event description: An event's description is contained on a line of the format "SUMMARY:<string>" where the characters following the colon comprise the string describing the event's nature. These strings will never contain the ":" character.
7. Repeat specification: If an event repeats, this will be indicated by a line of the format "RRULE:FREQ=<frequency>;UNTIL=<icalendardate>". The only frequencies you must account for are weekly frequencies. The date indicated by UNTIL is the last date on which the event will occur (i.e., is inclusive). Note that this line contains a colon (":") and semicolon (";") and equal signs ("=").
8. Events within the input stream are not necessarily in chronological order.
9. Events may overlap in time.
10. No event will ever cross a day boundary.
11. All times are local time (i.e., no timezones will appear in a date/time string). This semester we will ignore the effect of switching to daylight savings.

Output specification

1. All output is to stdout.
2. All events which occur from 12:00 am on the --start date and to 11:59 pm on the --end date must appear in chronological order based on the event's starting time that day.
3. If events occur on a particular date, then that date must be printed only once in the following format:

   ```
   <month text> <day>, <year> (<day of week>)
   ------------------------------------------
   ```
   Note that the line of dashes below the date must match the length of the date. You may use function such strftime() from the C library in order to create the calendar-date line.
4. Days are separated by a single blank line. There is no blank line at the start or at the end of the program's output.
5. Starting and ending times given in 12-hour format with "am" and "pm" as appropriate. For example, five minutes after midnight is represented as "12:05 am".
6. A colon is used to separate the start/end times from the event description
7. The event SUMMARY text appears on the same line as the even time. (This text may include parentheses.)
8. The event LOCATION text appears on after the SUMMARY text and is surrounded by square brackets.

Events from the same day are printed on successive lines in chronological order by starting time. Do not use blank lines to separate the event lines within the same day.

In the case of tests provided by the instructor, the Unix "diff" utility will be used to compare your program's output with what is expected for that test. Significant differences reported by "diff" may result in grade reductions.