Notes for Programming in C Lab Session #7

October 7, 2020

1 Introduction

The purpose of this lab session is to write a small program that makes use of arena-style allocation.

2 Overview

In this lab, you will define some functions to do match strings against a subset of regular expressions. In the `re.h` header file, we define the following data type

```
enum re_tag { CHR, SEQ, ALT };
typedef struct re Regexp;
struct re {
  enum re_tag type;
  union data {
    struct { char c; } chr;
    struct { Regexp *fst; Regexp *snd; } pair;
  } data;
};
```

This is a data type for representing trees. We define an enumeration `enum re_tag`, which says that we have 3 possibilities, either a single-character `CHR`, an alternative `ALT`, and a sequential composition `SEQ`.

Next, we define a structure type `Regexp`, with two fields. The first is the `type` field, which is one of the tags from the enumeration above. The second is a union type `data`, which is either a character `chr`, or a structure containing a pair of pointers to two regular expressions. The `type` fields determines which case of the union to use – a valid `Regexp` structure has a character if the type field is `CHR`, and a pair if the type field is `SEQ` or `ALT`.

The idea is that a CHR-regexp matches a single character string, the SEQ-regexp matches if the first part of the string matches the first element of the pair and the second part of the string, and the ALT-regexp matches if the string matches either the first or the second element of the pair.

Below, I give a table of example regexps, strings and whether or not there is a match. Here, `x`, `a`, `b`, `c`, `u`, and `v` are characters, juxtaposition represents SEQuential composition, and (+) denotes ALTernative.

Regexp	`"ab"`	`"xab"`	`"xba"`	`"axu"`
x(ab+ba)	✗	✓	✓	✗
(a+b+c)(x+y)(u+v+w)	✗	✗	✗	✓

3 Instructions

1. Download the `lab7.tar.gz` file from the class website.

2. Extract the file using the command `tar xvzf lab7.tar.gz`.
3. This will extract the `lab7/` directory. Change into this directory using the `cd lab7/` command.
4. In this directory, there will be files `lab7.c`, `re.h`, and `re.c`.
5. There will also be a file `Makefile`, which is a build script which can be invoked by running the command `make` (without any arguments). It will automatically invoke the compiler and build the `lab7` executable.
6. As usual, a sanitized version of the executable can be built with `make sane`.
7. Run the `lab7` executable, and see if your program works. The expected correct output is in a comment in the `lab7.c` file.

4 The Types and Functions to Implement

- `struct arena`

  The `re.h` file contains a declaration of the arena structure, but does not define it. Define an arena type for allocating pointers to `Regexp` structures, following the pattern of lecture 6.

- `Regexp *re_alloc(arena_t a);`

  Given an arena `a`, the `re_alloc` function should allocate a new `Regexp` and return a pointer to it. If the arena lacks room to allocate a new `Regexp`, it should return `NULL`.

- `void arena_free(arena_t a);`

  Given an arena `a`, the `arena_free` function should deallocate the arena and its associated storage. This should be a simple, non-recursive function!

- `Regexp *re_chr(arena_t a, char c);`

  Allocate a regexp matching the character `c`, allocating from the arena `a`. Return NULL if no memory is available.

- `Regexp *re_alt(arena_t a, Regexp *r1, Regexp *r2);`

  Allocate a regexp representing the alternative of `r1` and `r2` from the arena `a`. Return NULL if no memory is available.

- `Regexp *re_seq(arena_t a, Regexp *r1, Regexp *r2);`

  Allocate a regexp representing the sequencing of `r1` and `r2` from the arena `a`. Return NULL if no memory is available.

- `int re_match(Regexp *r, char *s, int i);`

  Given a regular expression `r`, a string `s`, and a valid index into the string `i`, this function should return an integer. If the function returns a nonnegative `j`, then the regular expression should match the substring running from i to j, including i but not j. (So if the `re_match(r, s, 5)` returns 8, then the subrange `s[5], s[6], s[7]` should match the regexp `r`.)

  It may help to look at the `re_print` function to see how to switch between the alternative branches of the regexp type.