CS 241 — Spring 2020 — Assignment 5

Assignments for CS 241

← Assignment 4 | Assignment 5 | Assignment 6→

Friday, June 12th at 5:00 pm | **Monday, June 22nd at 5:00 pm** | Friday, June 26th at 5:00 pm

P1 • P2 • P3 • P4 • P5 • Bonus • P6 • P7 • P8

Marmoset Notes

There are **no secret tests** for this assignment.

Part I. DFAs

For problems 1-6, you are to define DFAs (deterministic finite automata) to recognize several formal languages. Each DFA is represented as a file that will enumerate the alphabet, states, initial state, final states, and transitions that comprise the DFA, in the format described in http://www.student.cs.uwaterloo.ca/~cs241/dfa/DFAfileformat.html. For bonus marks, you may write a Racket or C++ program that reads such a file and implements the DFA recognizer.

You must submit to the Marmoset grading system. For all except the bonus problem, your submission is a text file containing the description of a DFA. You may create this text file with any editor on the Linux student environment. *Hint: some of the DFA description files may be large (of the order of 100 lines) and you may wish to write a program to help you generate them.* For all except the bonus problem, every failed test will result in a message giving the specific input causing the failure. There are no release tests.

For testing (all of your submissions, not just the bonus) an implementation of the bonus problem is available in the student.cs environment. To use it, enter

```
cs241.DFA < inputfile
```

where *inputfile* is a DFA plus a set of test inputs as described in the bonus problem specification.

Problem 1 — 8 marks of 60 (filename: `notdiv3.dfa`)

Write a DFA with alphabet {0,1} that recognizes binary integers that have no useless (leading) zeroes and are not divisible by 3. The first few such integers are 1, 10, 100, 101, 111, ...

Click here to go back to the top of the page.

Problem 2 — 8 marks of 60 (filename: `not23.dfa`)

Write a DFA with alphabet {0,1} that recognizes binary integers that have no useless (leading) zeroes, are not divisible by 2, and are not divisible by 3. The first few such integers are 1, 101, 111, 1011, 1101, ...

Click here to go back to the top of the page.

Problem 3 — 8 marks of 60 (filename: `int.dfa`)

Write a DFA that recognizes a decimal number between -128 and 127 inclusive, with no useless zeroes. Your DFA should recognize *numbers*, not expressions involving numbers and the unary minus operator, so things like `-0` or `--12` should not be accepted.

The alphabet symbols are {0,1,2,3,4,5,6,7,8,9,- }.

Click here to go back to the top of the page.

Problem 4 — 8 marks of 60 (filename: `pets.dfa`)

Write a DFA with alphabet {cat, dog, iguana, mouse} that recognizes any sequence of these animals so long as it contains the name of each animal at least once, in alphabetical order. That is, "**cat dog** mouse **iguana** dog **mouse**" and "iguana **cat** mouse **dog iguana** iguana **mouse** iguana" are in the language, but not "cat dog mouse iguana dog" .

Click here to go back to the top of the page.

Problem 5 — 8 marks of 60 (filename: `abcd.dfa`)

Write a DFA that recognizes any string from the alphabet {a,b,c,d} containing abcabd as a substring.

Click here to go back to the top of the page.

Bonus Problem — 5% bonus on Assignment 5 grade (filename: `dfa.rkt` or `dfa.cc`)

Write a Racket or C++ program that implements a DFA recognizer. Your program should read a file containing a DFA description as defined above, followed by several additional lines of input. Each additional line of input is a string of alphabet symbols separated by spaces. For each string, your program should print true if the string is in the language, and false if it is not. You may assume that the input to your program is valid according to the definition given in the file format specification.

Sample input:

```
2
0
1
5
start
zero
0mod3
1mod3
2mod3
start
2
zero
0mod3
8
start 0 zero
start 1 1mod3
1mod3 0 2mod3
1mod3 1 0mod3
2mod3 0 1mod3
2mod3 1 2mod3
0mod3 0 0mod3
0mod3 1 1mod3
0
```

```
1
1 0
1 1
1 0 0
1 0 1
1 1 0
1 1 1
```

Correct output for sample input:

```
true
false
false
true
false
false
true
false
```

Click here to go back to the top of the page.

Part II. The WLP4 Programming Language

This is the beginning of a series of several assignments that involve learning WLP4 and writing a compiler that translates WLP4 to MIPS assembly language.

Please consult WLP4 Programming Language Tutorial for an informal explanation of WLP4; The WLP4 Language Specification should be consulted for the definitive specification of the WLP4 language.

The following WLP4 program computes the sum of two integers, a and b.

```
//
// WLP4 program with two integer parameters, a and b
//   returns the sum of a and b
//
int wain(int a, int b) {
    return a + b;   // unhelpful comment about summing a and b
}
```

You may test this program on the student.cs environment by placing it in a file named `test.wlp4` and entering the following commands, which compile it to MIPS machine language and run it with the familiar `mips.twoints` and `mips.array` commands from Assignments 1 and 2:

```
wlp4c < test.wlp4 > test.mips
mips.twoints test.mips
```

Problem 6 — 5 marks of 60 (filename: `exp.wlp4`)

It is sometimes useful to compute x^n, but the naive approach requires n multiplications, which can be unacceptably inefficient. Instead, for fixed-sized numbers, exponentiation can be achieved in O(log(n)) time, where n is the exponent, using a method called exponentiation by squaring. Write a WLP4 program that takes two parameters: x and n, and returns x^n, where x and n are integers, and n is non-negative, using the exponentiation by squaring method. **Your program must solve the problem using iteration.** Hint: https://en.wikipedia.org/wiki/Exponentiation_by_squaring

Click here to go back to the top of the page.

Problem 7 — 5 marks of 60 (filename: `binsearch.wlp4`)

Write a WLP4 **function** that performs binary search on an array of integers. The function must be called `binsearch` and must accept 3 parameters: a pointer to the beginning of an array of integers, an integer representing the array's size, and an integer value to search for in the array, in that order. You may assume that the array is sorted in ascending order and contains each unique value exactly once. If the search value is in the array, your function should return its index. If it is not, your function should return -1. To test the function, you will of course need to write a `wain` function as well. **However, you are to submit only the function `binsearch`; the file you submit should not contain a `wain` function.** We will use our own `wain` function to test your implementation. Hint: https://en.wikipedia.org/wiki/Binary_search_algorithm

Click here to go back to the top of the page.

Problem 8 — 10 marks of 60 (filename: `wlp4.dfa`)

Using the DFA description file format, create a deterministic finite automaton that recognizes any valid WLP4 token, from the list given in the WLP4 specification. The alphabet consists of every character that may appear in any token; this does **not** include white space characters.

[While a *lexical scanner* for WLP4 must allow WLP4 tokens to be separated by white space so that it may distinguish between two consecutive tokens (eg 42 and 17) whose concatenation would constitute a single token (eg 4217), a DFA that accepts only input comprising a *single* token has no such need.]

*Hint: WLP4 tokens include keywords (such as int, wain, return, etc.), as well as identifiers. When scanning WLP4, two approaches are equally valid. One approach is to distinguish between keywords within the DFA, using separate states for each keyword. Another approach is to design a DFA that just recognizes identifiers and keywords the same way, and write additional code outside of the DFA to determine whether the token is a specific keyword or an identifier. For this problem, you may choose either approach. Note that the approach of using separate states for each keyword will result in a **significantly** larger DFA.*

Hint: In the case of numeric values, similarly to keywords, there are two valid ways of dealing with determining their validity. One approach is to design a DFA that does the explicit bounds checking in various states: from A5P4, you know how much work this is. The alternative, which is what we expect for this problem, is to deal with valid numerical syntax inside the DFA and deal with bounds checking in additional code written outside the DFA. In other words, ensure that you accept numbers that have the correct form, even if they would be out-of-bounds in terms of valid WLP4 numeric values.

Click here to go back to the top of the page.