COS 217: Introduction to Programming Systems

Writing Simple Programs and Building Executables

From Product Specification to Design to Code

PRINCETON UNIVERSITY

Agenda

A simple character processing program: Upper. Why?

- Learn to go from product specification to design to code
- Learn to structure a simple C program. Trace its execution beginning to end
- Learn to use a C library (ctype)

A more complex character processing program: Upper1. Why?

- Assignment 1
- Design step more involved: designing the simple DFA model
- Coding Step: develop a C program to implement the DFA

Building an executable C program

Next time: design decisions in upper, upper1

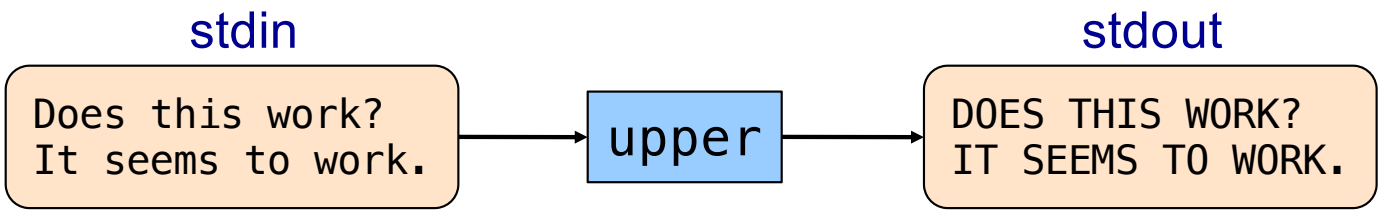
`upper` Product Specification

Read all chars from stdin

Convert every lower-case alphabetic char to upper case

- Leave other kinds of char alone

Print results to stdout



`upper` Program Design

Read a char from stdin

If it is a lowercase alphabetical character, turn it into uppercase, and write it to stdout

If it is not, keep it as is, but just write it to stdout

Can we optimize this, from a code size perspective?

Read a char from stdin

If it is a lowercase alphabetical character, turn it into uppercase

If it is not, keep it as is

Write the resulting character to stdout

upper C Program

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

Now let's walk through this element by element and see the program is executed

Tracing through `upper`: Starting up

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

Block `/* */` comments are the **only** legal ones in C90: no `//`

Execution begins at the **`main()`** function

- No classes in C

Tracing through `upper`: Defining Variables

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

Variables must be declared at the top of a block

We allocate space for c in the stack section of memory

Why **`int`** not **`char`**?

Tracing through `upper`: Reading and Processing Input

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

getchar() tries to read char from stdin

- Success ⇒ returns that char value (as int)
- Failure ⇒ returns a special value: **EOF**

Tracing through `upper`: Reading and Processing Input

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

We read a character at a time in a `while` loop until we hit EOF, for every character we read (not EOF), we process and print it

Simpler version of loop:

```
c = getchar();
while (c != EOF)
{   if (islower(c))
      c = toupper(c);
    putchar(c);
    c = getchar(c);
}
```

Tracing through `upper`: Using Library Functions

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

ctype.h Functions

```
$ man islower
NAME
       isalnum, isalpha, isascii, isblank, iscntrl, isdigit, isgraph,
       islower, isprint, ispunct, isspace, isupper, isxdigit –
       character classification routines

SYNOPSIS
       #include <ctype.h>
       int isalnum(int c);
       int isalpha(int c);
       int isascii(int c);
       int isblank(int c);
       int iscntrl(int c);
       int isdigit(int c);
       int isgraph(int c);
       int islower(int c);
       int isprint(int c);
       int ispunct(int c);
       int isspace(int c);
       int isupper(int c);
       int isxdigit(int c);
```

These functions check whether c, which must have the value of an unsigned char or EOF, falls into a certain character class.

...

islower() checks for a lowercase character.

Leave me out of this?

What build tool will be limited (and thus complain with a warning) if we omit the library preprocessor directive?

A: Preprocessor

B: Compiler

C: Assembler

D: Linker

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

B: Compiler gives warning that it hasn't seen declaration for `islower` or `toupper`

... but build does ultimately succeed.

Tracing through `upper`: The End Game

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

- Eventually getchar() returns EOF
- Loop condition fails
- We exit the loop, having output what we needed to output

Tracing through `upper`: The Exit

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

- return statement returns control to calling function
- return from main() returns to _start, terminates program

Normal execution ⇒ 0 or **EXIT_SUCCESS**
Abnormal execution ⇒ **EXIT_FAILURE**
#include <stdlib.h> to use these constants

Agenda

A simple character processing program: Upper. Why?

- Learn to go from product specification to design to code
- Learn to structure a simple C program. Trace its execution beginning to end
- Learn to use a C library (ctype)

A more complex character processing program: Upper1. Why?

- Assignment 1
- Design step more involved: designing the simple DFA model
- Coding Step: develop a C program to implement the DFA

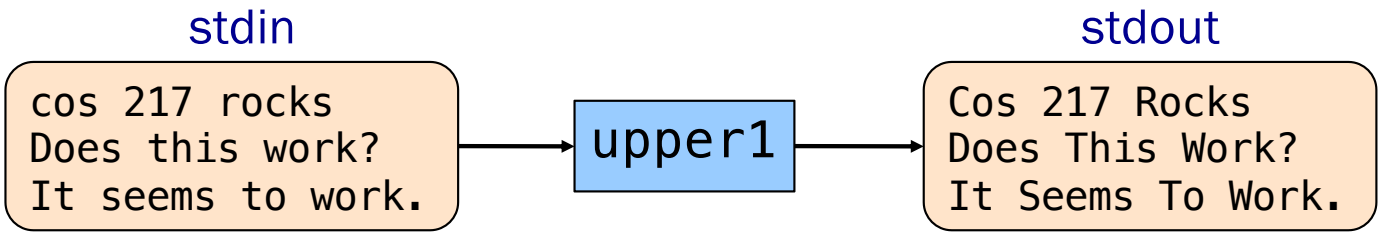
Building an executable C program

Next time: design decisions in upper, upper1

The upper1 program

Functionality

- Read all chars from stdin
- Capitalize the first letter of each word
  - "cos 217 rocks" ⇒ "Cos 217 Rocks"
- Write result to stdout

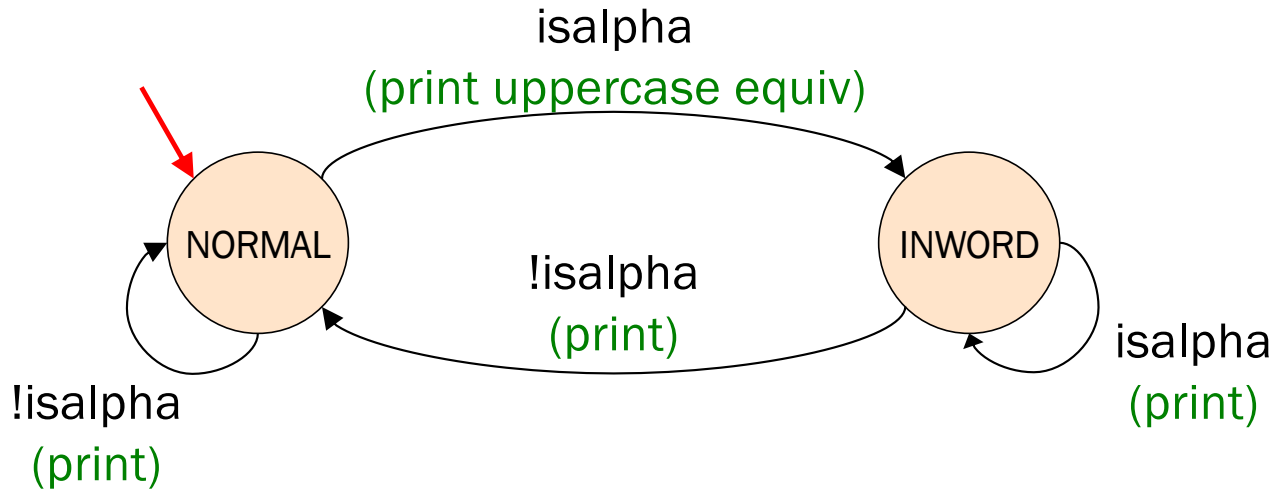


What are the key things we need to do?

- Recognize when we're *"in a word"* vs *"**not** in a word"*
- Reason about what to do with that information in a systematic way
  - *if in a word, don't capitalize until we leave the word*
  - *if not in word, capitalize next time we see a non-whitespace char*

upper1 Program Design

Deterministic Finite State Automaton (DFA)



- States, one of which is designated as the start
- Transitions labeled by individual or categories of chars
- Optionally, actions on transitions
- Usually (but not here) a notion of accept ✅ and reject ❌ states

Agenda

A simple character processing program: Upper. Why?

- Learn to go from product specification to design to code
- Learn to structure a simple C program. Trace its execution beginning to end
- Learn to use a C library (ctype)

A more complex character processing program: Upper1. Why?

- Assignment 1
- Design step more involved: designing the simple DFA model
- Coding Step: develop a C program to implement the DFA

Building an executable C program

Next time: design decisions in upper, upper1

Agenda

A simple C program: Upper. Why?

- Learn to structure a simple C program. Trace its execution beginning to end
- Learn to use of a C library (ctype)
- Basis for a more complex program that implements a DFA (upper1)
- Will come back to it this simpler program when looking at the build process

A DFA character processing program: Upper1. Why?

- Assignment 1
- Going from product specification to design to program
- Design: designing the simple DFA model
- Coding: develop a C program to implement the DFA

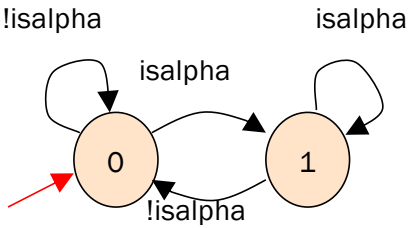
Building an executable C program

Next time: design decisions in upper, upper1

upper1 C Program, Version 1

```
#include <stdio.h>
#include <ctype.h>
int main(void) {
   int c;
   int state = 0;
   while ((c = getchar()) != EOF) {
      switch (state) {
         case 0:
            if (isalpha(c)) {
               putchar(toupper(c)); state = 1;
            } else {
               putchar(c); state = 0;
            }
            break;
         case 1:
            if (isalpha(c)) {
               putchar(c); state = 1;
            } else {
               putchar(c); state = 0;
            }
            break;
      }
   }
   return 0;
}
```

That's a B.
What's wrong?



`upper1` C Program, Toward Version 2

Problem:

- The program works, but…
- States should have names

Solution:

- Define your own named constants:

- `enum Statetype {NORMAL, INWORD};`
  - Define an enumeration type
    (a type with literals that are semantically meaningful names for a subset of integer values)
    (values start at 0 or a specifically assigned value)
    (subsequent values increment by 1 over previous if not specifically assigned)
- `enum Statetype state;`
  - Define a variable of that type

upper1 C Program, Version 2

```
…
enum Statetype {NORMAL, INWORD};
int main(void) {
   int c;
   enum Statetype state = NORMAL;
   while ((c = getchar()) != EOF) {
      switch (state) {
         case NORMAL:
            if (isalpha(c)) {
               putchar(toupper(c)); state = INWORD;
            } else {
               putchar(c); state = NORMAL;
            }
            break;
         case INWORD:
            if (isalpha(c)) {
               putchar(c); state = INWORD;
            } else {
               putchar(c); state = NORMAL;
            }
            break;
      }
   }
   return 0;
}
```

That's a B+.
What's wrong?

`upper1` C Program, Toward Version 3

Problem:

- The program works, but…
- Deeply nested statements
- No modularity

Solution:

- Handle each state in a separate function

upper1 C Program, Version 3

```
#include <stdio.h>
#include <ctype.h>
enum Statetype {NORMAL, INWORD};

enum Statetype
handleNormalState(int c)
{
   enum Statetype state;
   if (isalpha(c)) {
      putchar(toupper(c));
      state = INWORD;
   } else {
      putchar(c);
      state = NORMAL;
   }
   return state;
}
```

```
enum Statetype
handleInwordState(int c)
{
   enum Statetype state;
   if (!isalpha(c)) {
      putchar(c);
      state = NORMAL;
   } else {
      putchar(c);
      state = INWORD;
   }
   return state;
}
```

```
int main(void)
{
   int c;
   enum Statetype state = NORMAL;
   while ((c = getchar()) != EOF) {
      switch (state) {
         case NORMAL:
            state = handleNormalState(c);
            break;
         case INWORD:
            state = handleInwordState(c);
            break;
      }
   }
   return 0;
}
```

That's an A-.
What's wrong?

Agenda

A simple C program: Upper. Why?

- Learn to structure a simple C program. Trace its execution beginning to end
- Learn to use of a C library (ctype)
- Basis for a more complex program that implements a DFA (upper1)
- Will come back to it this simpler program when looking at the build process

A DFA character processing program: Upper1. Why?

- Assignment 1
- Going from product specification to design to program
- Design: designing the simple DFA model
- Coding: develop a C program to implement the DFA

Building an executable C program

Next time: design decisions in upper, upper1

Agenda

A simple C program: Upper. Why?

- Learn to structure a simple C program. Trace its execution beginning to end
- Learn to use of a C library (ctype)
- Basis for a more complex program that implements a DFA (upper1)
- Will come back to it this simpler program when looking at the build process

A DFA character processing program: Upper1. Why?

- Assignment 1
- Going from product specification to design to program
- Design: designing the simple DFA model
- Coding: develop a C program to implement the DFA

Building an executable C program

Next time: design decisions in upper, upper1

Building the `upper` Executable

We'll go back to upper, just because the code fits better on a slide

```
$ gcc217 upper.c
$ ls
.    ..   a.out
$
$ gcc217 upper.c -o upper
$ ls
.    ..   a.out     upper
$ ./upper
cos 217 rocks
Does this work?
It seems to work.
^D
```

What do we see in our terminal emulator after this?

`upper` Build Process in Detail

The starting point:

`upper.c`

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

- C language
- Missing declarations of `getchar()`, `putchar()`, islower(), toupper()
- Missing definitions of getchar(), putchar(), islower(), toupper()

`upper` Build Process in Detail

Question:

- Exactly what happens when you issue the command
  `gcc217 upper.c –o upper`

Answer: Four steps

- Preprocess
- Compile
- Assemble
- Link

`upper` Build Process: Preprocessor

Command to preprocess:

- `gcc217 –E upper.c > upper.i`

Preprocessor functionality

- Removes comments
- Handles preprocessor directives

upper Build Process: Preprocessor

upper.c

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

Preprocessor removes comment (this is A1!)

upper Build Process: Preprocessor

upper.c

```
#include <stdio.h>
#include <ctype.h>
/* Turn letters in stdin to uppercase
and print result to stdout. Return 0. */
int main(void)
{  int c;
   while ((c = getchar()) != EOF)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

Preprocessor replaces
`#include <stdio.h>`
with contents of
`/usr/include/stdio.h`
Similarly for ctype.h

Preprocessor replaces
`EOF` with –1

upper Build Process: Preprocessor

The result

upper.i

```
...
int getchar();
int putchar();
...
int islower(int a)
int toupper(int a)
...

int main(void)
{  int c;
   while ((c = getchar()) != -1)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

- C language
- Without comments
- Without preprocessor directives
- Contains code from `stdio.h`: **declarations** of `getchar()`, `putchar()`, `etc.`
- Missing **definitions** of `getchar()`, `putchar()`, `etc.`
- Contains value for `EOF`

`upper` Build Process: Compiler

Command to compile:

- `gcc217 –S upper.i`

Compiler functionality

- Translate from C to assembly language
- Check syntax
- Check types. Use function declarations to check calls of `getchar()`, `putchar()`, `islower()`, `toupper()`

upper Build Process: Compiler

upper.i

```
...
int getchar();
int putchar());
...
int islower(int a);
int toupper(int a);
...
int main(void)
{  int c;
   while ((c = getchar()) != -1)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

- Compiler sees function **declarations**
- These give compiler enough information to check subsequent calls of `getchar()`, `putchar()`, `islower()`, `toupper()`

upper Build Process: Compiler

upper.i

```
...
int getchar();
int putchar());
int islower(int a);
int toupper(int a);
...
int main(void)
{  int c;
   while ((c = getchar()) != -1)
   {  if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

- Definition of `main()` function
- Compiler checks calls of `getchar()`, `putchar()`, `islower()`, `toupper()`
- Compiler translates C code to assembly language directives and instructions progressively

`upper` Build Process: Compiler

`upper.s`

```
        .section        .rodata
.LC0:
        .string "%d\n"

        .section        .text
        .global main
main:
        stp     x29, x30, [sp, -32]!
        add     x29, sp, 0
        str     wzr, [x29,24]
        bl      getchar
        str     w0, [x29,28]
        b       .L2
.L3:

          ….
.L2:

        …..
```

- Assembly language
- Missing definitions of `getchar()`, `putchar()`, `islower()`, `toupper()`

`upper` Build Process: Assembler

Command to assemble:

- `gcc217 –c upper.s`

Assembler functionality

- Translate from assembly language to machine language

`upper` Build Process: Assembler

The result:

`upper.o`

```
Machine language
version of the
program

No longer human
readable
```

- Machine language
- (Still) Missing definitions of `getchar`, `putchar()`, `islower()`, `toupper()`

`upper` Build Process: Linker

Command to link:

- `gcc217 upper.o –o upper`

Linker functionality

- Resolve references within the code
- Fetch machine language code from the standard C library (`/usr/lib/libc.a`) to make the program complete
- Produce final executable

`upper` Build Process: Linker

The result:

`upper`

```
Machine language
version of the
program

No longer human
readable
```

- Machine language
- Contains definitions of `getchar()`, `printf()`, `islower()`, `toupper()`

Complete. Executable.

Agenda

A simple C program: Upper. Why?

- Learn to structure a simple C program. Trace its execution beginning to end
- Learn to use of a C library (ctype)
- Basis for a more complex program that implements a DFA (upper1)
- Will come back to it this simpler program when looking at the build process

A DFA character processing program: Upper1. Why?

- Assignment 1
- Going from product specification to design to program
- Design: designing the simple DFA model
- Coding: develop a C program to implement the DFA

Building an executable C program

Next time: design decisions in upper, upper1

Agenda

A simple C program: Upper

- Structure and execution
- Use of a C library (ctype)
- Basis for a more complex program that implements a DFA (upper1)
- Will come back to it this simpler program when looking at the build process

A DFA character processing program: Upper1

- Design: designing the simple DFA model
- Coding: develop a C program to implement the DFA

Building an executable C program

Next time: design decisions in upper, upper1

Sample Exam Question (Spring 2020, Exam 1)

(1c – 5 points) Consider the following program:

```
1     #include <stdio.h>
2     enum mottoWords
3     {
4        IN, THE,
5        SERVICE = 1746,
6        OF = 1896,
7        HUMANITY
8     };
9
10    int main(void) {
11       printf("%d %d %d %d %d\n",
12                IN, THE, SERVICE, OF, HUMANITY);
13       return 0;
14    }
```

What values are printed to standard output? You can write GARBAGE to represent an uninitialized value that is printed. Answer in the space below the code box.

Sample Exam Question (Spring 2020, Exam 1)

(1d – 8 points) Consider the following program:

```
1     #include <stdio.h>
2     int main(void) {
3        char c;
4        scanf("%c", &c);
5        switch(c) {
6           case 'a':
7              printf("do ");
8              break;
9           case 'b':
10             printf("re ");
11          default:
12             printf("mi ");
13          case 'c':
14             printf("fa ");
15             break;
16          case 'd':
17             printf("so ");
18       }
19       printf("\n");
20       return 0;
21    }
```

Recall that `default` is triggered if the `switch` expression matches none of the `case` options. What is printed to standard output for each of the five inputs below? Answer beside each corresponding input.

a

b

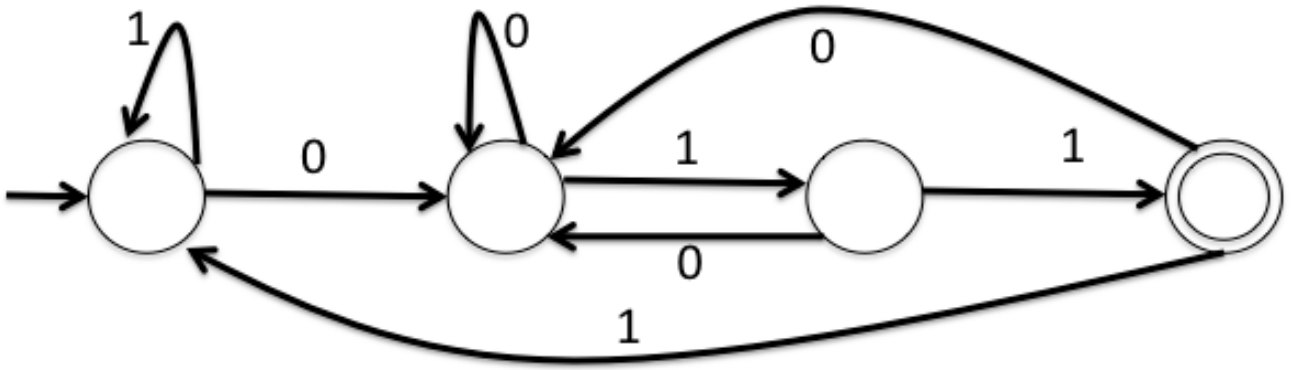
c

d

e

Sample Exam Question (Fall 2015, Exam 1)

State concisely what sequences (and only those sequences) this four-state DFA accepts. Assume all sequence characters are either '0' or '1', that the leftmost state is the initial state, and that the rightmost state is the only accept state. (6 points / 100)



Appendix:

Additional DFA Examples

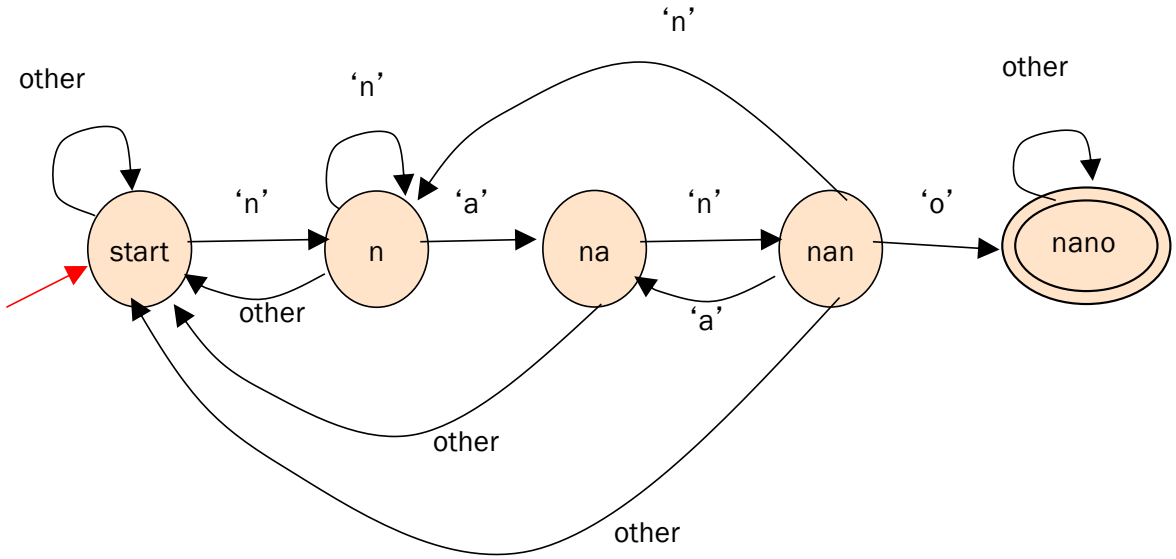
Another DFA Example

Does the string have "nano" in it?

- "banano" ⇒ yes
- "nnnnnnnanofff" ⇒ yes
- "banananonano" ⇒ yes
- "bananananashanana" ⇒ no

Double circle is accepting state

Single circle is rejecting state



Yet Another DFA Example

Old (Hard!) Exam Question
Compose a DFA to identify whether or not
a string is a floating-point literal

Valid literals

- "-34"
- "78.1"
- "+298.3"
- "-34.7e-1"
- "34.7E-1"
- "7."
- ".7"
- "999.99e99"

Invalid literals

- "abc"
- "-e9"
- "1e"
- "+"
- "17.9A"
- "0.38+"
- "."
- "38.38f9"