

Princeton University

Computer Science 217: Introduction to Programming Systems



A Taste of C



1

Goals of this Lecture



Help you learn about:

- The basics of C
- Deterministic finite-state automata (DFA)
- Expectations for programming assignments

Why?

- Help you get started with Assignment 1
 - Required readings...
 - + coverage of programming environment in precepts...
 - + minimal coverage of C in this lecture...
 - = enough info to start Assignment 1
- DFAs are useful in many contexts
 - E.g., Assignment 1, Assignment 7

2

Agenda



The charcount program

The upper program

The upper1 program

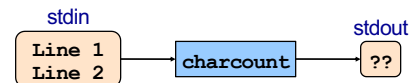
3

The “charcount” Program



Functionality:

- Read all characters from standard input stream
- Write to standard output stream the number of characters read



4

iClicker Question

Q: What is the output of `charcount` on this input?

```

[armlab01:lecture2$./charcount
Line 1
Line 2
  
```

stdout
??

- A. 10
- B. 12
- C. 13
- D. 14
- E. 15

5

The “charcount” Program



The program:

```

charcount.c
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{
    int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {
        charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
  
```

6

Running "charcount"

Run-time trace, referencing the original C code...

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

Execution begins at **main()** function

- No classes in the C language.

7

Running "charcount"

Run-time trace, referencing the original C code...

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

We allocate space for **c** and **charCount** in the stack section of memory

Why **int** instead of **char**?

8

Running "charcount"

Run-time trace, referencing the original C code...

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

getchar() tries to read char from stdin

- Success $\Rightarrow$ returns char (within an int)
- Failure $\Rightarrow$ returns **EOF**

EOF is a special non-char value, different from all possible chars, that **getchar()** returns to indicate failure

9

Running "charcount"

Run-time trace, referencing the original C code...

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

Assuming **c $\neq$ EOF**, we increment **charCount**

10

Running "charcount"

Run-time trace, referencing the original C code...

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

We call **getchar()** again and recheck loop condition

11

Running "charcount"

Run-time trace, referencing the original C code...

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

- Eventually **getchar()** returns **EOF**
- Loop condition fails
- We call **printf()** to write final **charCount**

12

Running “charcount”

Run-time trace, referencing the original C code...

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{
    int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {
        charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

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

Normal execution ⇒ return 0 or **EXIT_SUCCESS**
Abnormal execution ⇒ return **EXIT_FAILURE**

13

13

“charcount” Building and Running

```
$ gcc217 charcount.c
$ ls
.  ..  a.out
$ gcc217 charcount.c -o charcount
$ ls
.  ..  a.out  charcount
```

14

14

“charcount” Building and Running

```
$ gcc217 charcount.c -o charcount
$ ./charcount
Line 1
Line 2
^D
14
$
```

What is this?
What is the effect?

15

15

“charcount” Building and Running

```
$ cat somefile
Line 1
Line 2
$ ./charcount < somefile
14
$
```

What is this?
What is the effect?

16

16

“charcount” Building and Running

```
$ ./charcount > someotherfile
Line 1
Line 2
^D
$ cat someotherfile
14
```

What is this?
What is the effect?

17

17

“charcount” Build Process in Detail

Question:

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

Answer: Four steps

- Preprocess
- Compile
- Assemble
- Link

18

18

“charcount” Build Process in Detail



The starting point

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

- C language
- Missing definitions of `getchar()` and `printf()`

19

19

Preprocessing “charcount”



Command to preprocess:

```
• gcc217 -E charcount.c > charcount.i
```

Preprocessor functionality

- Removes comments
- Handles **preprocessor directives**

20

20

Preprocessing “charcount”



charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

Preprocessor removes
comment

21

21

Preprocessing “charcount”



charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

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

Preprocessor replaces
EOF with -1

22

22

Preprocessing “charcount”



The result

charcount.i

```
...
int getchar();
int printf(char *fmt, ...);
...

int main(void)
{ int c;
  int charCount = 0;
  c = getchar();
  while (c != -1)
  { charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

- C language
- Missing comments
- Missing preprocessor directives
- Contains code from `stdio.h`:
declarations of `getchar()` and `printf()`
- Missing **definitions** of `getchar()` and `printf()`
- Contains substitution for EOF

23

23

Compiling “charcount”



Command to compile:

```
• gcc217 -S charcount.i
```

Compiler functionality

- Translate from C to assembly language
- Use function declarations to check calls of `getchar()` and `printf()`

24

24

Compiling "charcount"

charcount.i

```
...
int getchar();
int printf(char *fmt, ...);
...

int main(void)
{
    int c;
    int charCount = 0;
    c = getchar();
    while (c != -1)
    {
        charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

- Compiler sees function declarations
- So compiler has enough information to check subsequent calls of getchar() and printf()

25

25

Compiling "charcount"

charcount.i

```
...
int getchar();
int printf(char *fmt, ...);
...

int main(void)
{
    int c;
    int charCount = 0;
    c = getchar();
    while (c != -1)
    {
        charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

- Definition of main() function
- Compiler checks calls of getchar() and printf() when encountered
- Compiler translates to assembly language

26

26

Compiling "charcount"

The result:
charcount.s

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

.section .text
.global main

main:
    stp    x29, x30, [sp, -32]!
    add    x29, sp, 0
    str    w29, [x29, 24]
    bl     getchar
    str    w0, [x29, 28]
    .L3:
    ldr     w0, [x29, 24]
    add    w0, w0, 1
    str    w0, [x29, 24]
    bl     getchar
    str    w0, [x29, 28]
    .L2:
    ldr     w0, [x29, 28]
    cmn    w0, #1
    bne    .L3
    adrp    x0, .LC0
    add    x0, x0, :lo12:.LC0
    ldr     w1, [x29, 24]
    bl     printf
    mov     w0, 0
    ldp     x29, x30, [sp], 32
    ret
```

- Assembly language
- Missing definitions of getchar() and printf()

27

27

Assembling "charcount"

Command to assemble:

• gcc217 -c charcount.s

Assembler functionality

- Translate from assembly language to machine language

28

28

Assembling "charcount"

The result:

charcount.o

Machine language
version of the
program

No longer human
readable

- Machine language
- Missing definitions of getchar() and printf()

29

29

Linking "charcount"

Command to link:

• gcc217 charcount.o -o charcount

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

30

30

Linking “charcount”

The result:

charcount

Machine language
version of the
program

No longer human
readable

- Machine language
- Contains definitions of `getchar()` and `printf()`

Complete! Executable!

31

iClicker Question

Q: There are other ways to `charcount` – which is best?

A.

```
for (c=getchar(); c!=EOF; c=getchar())  
    charCount++;
```

B.

```
while ((c=getchar()) != EOF)  
    charCount++;
```

C.

```
for (;;)
{
    c = getchar();
    if (c == EOF)
        break;
    charCount++;
}
```

D.

```
c = getchar();
while (c!=EOF)
{
    charCount++;
    c = getchar();
}
```

32

Review of Example 1

Input/Output

- Including `stdio.h`
- Functions `getchar()` and `printf()`
- Representation of a character as an integer
- Predefined constant `EOF`

Program control flow

- The `for` and `while` statements
- The `break` statement
- The `return` statement

Operators

- Assignment: `=`
- Increment: `++`
- Relational: `==` `!=`

33

33

Agenda

The `charcount` program

The upper program

The `upper1` program

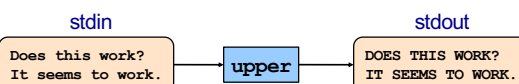
34

34

Example 2: “upper”

Functionality

- Read all chars from `stdin`
- Convert each lower-case alphabetic char to upper case
 - Leave other kinds of chars alone
- Write result to `stdout`



35

35

“upper” Building and Running

```
$ gcc217 upper.c -o upper
$ cat somefile
Does this work?
It seems to work.
$ ./upper < somefile
DOES THIS WORK?
IT SEEMS TO WORK.
$
```

36

36

ASCII

American Standard Code for Information Interchange

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 NUL									HT	LF					
16															
32 SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
48 0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64 @	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80 P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96 `	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112 p	q	r	s	t	u	v	w	x	y	z	{		}	~	

Partial map

Note: Lower-case and upper-case letters are 32 apart

37

"upper" Version 1

```
#include <stdio.h>
int main(void)
{ int c;
  while ((c = getchar()) != EOF)
  { if ((c >= 97) && (c <= 122))
    { c -= 32;
      putchar(c);
    }
  }
  return 0;
}
```

What's wrong?

38

Character Literals

Examples

'a'	the a character
	97 on ASCII systems
'\n'	newline
	10 on ASCII systems
'\t'	horizontal tab
	9 on ASCII systems
'\\'	backslash
	92 on ASCII systems
'\''	single quote
	39 on ASCII systems
'\0'	the null character (alias NUL)
	0 on all systems

39

"upper" Version 2

```
#include <stdio.h>
int main(void)
{ int c;
  while ((c = getchar()) != EOF)
  { if ((c >= 'a') && (c <= 'z'))
    { c += 'A' - 'a';
      putchar(c);
    }
  }
  return 0;
}
```

Arithmetic on chars?

What's wrong now?

40

EBCDIC

Extended Binary Coded Decimal Interchange Code

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 NUL															
16															
32															
48															
64 SP															
80 &															
96 -															
112															
128	a	b	c	d	e	f	g	h	i						
144	j	k	l	m	n	o	p	q	r						
160	s	t	u	v	w	x	y	z							
176															
192	A	B	C	D	E	F	G	H	I						
208	J	K	L	M	N	O	P	Q	R						
224	S	T	U	V	W	X	Y	Z							
240	0	1	2	3	4	5	6	7	8	9					

Partial map

Note: Lower case not contiguous; same for upper case

41

Character Literals

Examples

'a'	the a character
	97 on ASCII systems
	129 on EBCDIC systems
'\n'	newline
	10 on ASCII systems
	37 on EBCDIC systems
'\t'	horizontal tab
	9 on ASCII systems
	5 on EBCDIC systems
'\\'	backslash
	92 on ASCII systems
	224 on EBCDIC systems
'\''	single quote
	39 on ASCII systems
	125 on EBCDIC systems
'\0'	the null character (alias NUL)
	0 on all systems

42

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*
falls into various
character classes

43

ctype.h Functions

```
$ man toupper
NAME
    toupper, tolower - convert letter to upper or lower case

SYNOPSIS
    #include <ctype.h>
    int toupper(int c);
    int tolower(int c);

DESCRIPTION
    toupper() converts the letter c to upper case, if possible.
    tolower() converts the letter c to lower case, if possible.

    If c is not an unsigned char value, or EOF, the behavior of
    these functions is undefined.

RETURN VALUE
    The value returned is that of the converted letter, or c if
    the conversion was not possible.
```

44

“upper” Version 3

```
#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;
}
```

45

45

iClicker Question

Q: Is the if statement really necessary?

A. Gee, I don't know.
Let me check
the man page!

```
#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;
}
```

46

ctype.h Functions

```
$ man toupper
NAME
    toupper, tolower - convert letter to upper or lower case

SYNOPSIS
    #include <ctype.h>
    int toupper(int c);
    int tolower(int c);

DESCRIPTION
    toupper() converts the letter c to upper case, if possible.
    tolower() converts the letter c to lower case, if possible.

    If c is not an unsigned char value, or EOF, the behavior of
    these functions is undefined.

RETURN VALUE
    The value returned is that of the converted letter, or c if
    the conversion was not possible.
```

47

iClicker Question

Q: Is the if statement really necessary?

A. Yes, necessary
for correctness.

B. Not necessary,
but I'd leave it in.

C. Not necessary,
and I'd get rid of it.

```
#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;
}
```

48

Review of Example 2

Representing characters

- ASCII and EBCDIC character sets
- Character literals (e.g., 'A' or 'a')

Manipulating characters

- Arithmetic on characters
- Functions such as islower() and toupper()

49

Agenda

The charcount program

The upper program

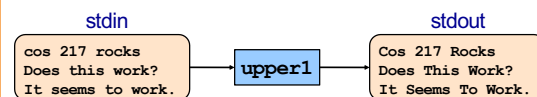
The upper1 program

50

Example 3: “upper1”

Functionality

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



51

“upper1” Building and Running

```

$ gcc217 upper1.c -o upper1
$ cat somefile
cos 217 rocks
Does this work?
It seems to work.
$ ./upper1 < somefile
Cos 217 Rocks
Does This Work?
It Seems To Work.
$
  
```

52

“upper1” Challenge

Problem

- Must remember where you are
- Capitalize “c” in “cos”, but not “o” in “cos” or “c” in “rocks”

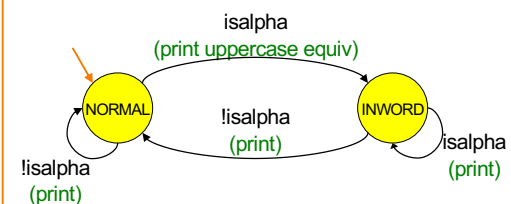
Solution

- Maintain some extra information
- “In a word” vs “not in a word”

53

Deterministic Finite Automaton

Deterministic Finite State Automaton (DFA)

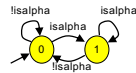


- **States**, one of which is denoted the **start** state
- **Transitions** labeled by chars or char categories
- Optionally, **actions** on transitions

54

“upper1” Version 1

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    int c;
    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?

55

55

“upper1” 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
- `enum Statetype state;`
- Define a variable of that type

56

56

“upper1” Version 2

```
#include <stdio.h>
#include <ctype.h>
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?

57

57

“upper1” Toward Version 3

Problem:

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

Solution:

- Handle each state in a separate function

58

58

“upper1” 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?

59

59

“upper1” Toward Final Version

Problem:

- The program works, but...
- No comments

Solution:

- Add (at least) function-level comments

60

60

Function Comments

Function comment should describe **what the function does** (from the caller's viewpoint)

- Input to the function
 - Parameters, input streams
- Output from the function
 - Return value, output streams, (call-by-reference parameters)

Function comment should **not** describe **how the function works**

61

Function Comment Examples

Bad main() function comment

```
Read a character from stdin. Depending upon
the current DFA state, pass the character to
an appropriate state-handling function. The
value returned by the state-handling function
is the next DFA state. Repeat until end-of-file.
```

Describes **how the function works**

Good main() function comment

```
Read text from stdin. Convert the first character
of each "word" to uppercase, where a word is a
sequence of characters. Write the result to
stdout. Return 0.
```

Describes **what the function does** from caller's viewpoint

62

"upper1" Final Version

```
/*-----*/
/* upper1.c
/* Author: Bob Dondoro
/*-----*/

#include <stdio.h>
#include <ctype.h>

enum Statetype {NORMAL, INWORD};
```

Continued on
next page

63

"upper1" Final Version

```
/*-----*/
/* Implement the NORMAL state of the DFA. c is the current
DFA character. Write c or its uppercase equivalent to
stdout, as specified by the DFA. Return the next state. */

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

Continued on
next page

64

"upper1" Final Version

```
/*-----*/
/* Implement the INWORD state of the DFA. c is the current
DFA character. Write c to stdout, as specified by the DFA.
Return the next state. */

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

Continued on
next page

65

"upper1" Final Version

```
/*-----*/
/* Read text from stdin. Convert the first character of each
"word" to uppercase, where a word is a sequence of
letters. Write the result to stdout. Return 0. */

int main(void)
{
    int c;
    /* Use a DFA approach. state indicates the DFA state. */
    enum Statetype state = NORMAL;
    while ((c = getchar()) != EOF)
    {
        switch (state)
        {
            case NORMAL:
                state = handleNormalState(c);
                break;
            case INWORD:
                state = handleInwordState(c);
                break;
        }
    }
    return 0;
}
```

66

Review of Example 3

Deterministic finite-state automaton

- Two or more states
- Transitions between states
 - Next state is a function of current state and current character
- Actions can occur during transitions

Expectations for COS 217 assignments

- Readable
 - Meaningful names for variables and literals
 - Reasonable max nesting depth
- Modular
 - Multiple functions, each of which does one well-defined job
- Function-level comments
 - Should describe what function does
- See K&P book for style guidelines specification

67

67

Summary

The C programming language

- Overall program structure
- Control statements (`if`, `while`, `for`, and `switch`)
- Character I/O functions (`getchar()` and `putchar()`)

Deterministic finite state automata (DFA)

Expectations for programming assignments

- Especially Assignment 1

Start Assignment 1 soon!

68

68

Appendix:

Additional DFA Examples

69

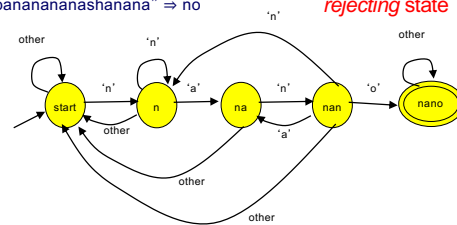
69

Another DFA Example

Does the string have "nano" in it?

- "banano" $\Rightarrow$ yes
- "nnnnnnnanoff" $\Rightarrow$ yes
- "banananonano" $\Rightarrow$ yes
- "bananananashanana" $\Rightarrow$ no

Double circle is
accepting state
Single circle is
rejecting state



70

70

Yet Another DFA Example

Old 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"

71

71