**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

**Agenda**

The charcount program
The upper program
The upper1 program

**The “charcount” Program**

Functionality:
- Read all chars from stdin (standard input stream)
- Write to stdout (standard output stream) the number of chars read

```
#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;
}
```

**iClicker Question**

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

A. 10
B. 12
C. 13
D. 14
E. 15
Running “charcount”

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

```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
", charCount);
    return 0;
}
```

Execution begins at `main()` function
- No classes, no methods in the C language

```
0:  #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
", charCount);
        return 0;
    }
```

Why isn’t `int` instead of `char`?

```
1:  #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
", charCount);
        return 0;
    }
```

Computer allocates space for `c` and `charCount` in the stack section of memory

```
2:  #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
", charCount);
        return 0;
    }
```

Why isn’t `int` instead of `char`?

```
3:  #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
", charCount);
        return 0;
    }
```

getchar() tries to read char from stdin
- Success ⇒ returns char (within an int)
- Failure ⇒ returns EOF

```
4:  #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
", charCount);
        return 0;
    }
```

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

```
5:  #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
", charCount);
        return 0;
    }
```

```
6:  #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
", charCount);
        return 0;
    }
```

Assuming `c ≠ EOF`, computer increments `charCount`

```
7:  #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
", charCount);
        return 0;
    }
```

```
8:  #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
", charCount);
        return 0;
    }
```

Eventually `getchar()` returns `EOF`
- Computer breaks out of loop
- Computer calls `printf()` to write `charCount`

```
9:  #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
", charCount);
        return 0;
    }
```

```
10: #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
", charCount);
        return 0;
    }
```

Computer calls `getchar()` again, and repeats

```
11: #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
", charCount);
        return 0;
    }
```

```
12: #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
", charCount);
        return 0;
    }
```

```
13: #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
", charCount);
        return 0;
    }
```
Running “charcount”

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

```
#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
", charCount);
  return 0;
}
```

- Computer executes return statement
- Return from main() terminates program

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

“charcount” Building and Running

```
$ gcc217 charcount.c -o charcount
$ ./charcount
cat somefile
Line 1
Line 2
^D
14
$ ./charcount < somefile
Line 1
Line 2
^D
$ ./charcount > someotherfile
Line 1
Line 2
^D
$ cat someotherfile
14
```

What is this? What is the effect?

“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

```
#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
", charCount);
  return 0;
}
```

- C language
- Missing definitions of getchar() and printf()
Preprocessing “charcount”

Command to preprocess:
• gcc217 -E charcount.c > charcount.i

Preprocessor functionality
• Removes comments
• Handles preprocessor directives

Preprocessor replaces
#include <stdio.h>
with contents of
/usr/include/stdio.h
Preprocessor replaces
EOF with -1

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()
Compiling “charcount”

charcount.i

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

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

Compiling “charcount”

The result: charcount.s

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

Assembling “charcount”

Command to assemble:
- gcc217 -c charcount.s

Assembler functionality
- Translate from assembly language to machine language

Assembling “charcount”

The result: charcount.o

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

Linking “charcount”

Command to link:
- gcc217 charcount.o -o charcount

Linker functionality
- Resolve references
- Fetch machine language code from the standard C library (/usr/lib/libc.a) to make the program complete

Linking “charcount”

The result: charcount

- Machine language
- Contains definitions of getchar() and printf()
  Complete! Executable!
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. c = getchar();
   while (c!=EOF)
      charCount++;
   c = getchar();

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

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: == !=

Agenda

The charcount program
The upper program
The upper1 program

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

ASCII

American Standard Code for Information Interchange

Note: Lower case and upper case letters are 32 apart
```c
#include <stdio.h>
int main(void)
{
    int c;
    while ((c = getchar()) != EOF)
    {
        if ((c >= 97) && (c <= 122))
            c -= 32;
        putchar(c);
    }
    return 0;
}
```

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

```
#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);
```

```
#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);
```
**iClicker Question**

Q: Is the if statement really necessary?

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

```c
#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;
}
```

**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.

```c
#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;
}
```

**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()
Agenda

The charcount program
The upper program
The upper1 program

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

 stdin
 cos 217 rocks
 Does this work?
 It seems to work.

 upper1

 stdout
 Cos 217 Rocks
 Does This Work?
 It Seems To Work.

“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.

“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"

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

“upper1” Version 1

```c
#include <ctype.h>
#include <stdio.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;
                break;
            }
            else
            {
                putchar(c); state = 0;
                break;
            }
        case 1:
            if (isalpha(c))
            {
                putchar(c); state = 0;
                break;
            }
            else
            {
                putchar(c); state = 1;
                break;
            }
        }
    }
    return 0;
}
```

That’s a B. What’s wrong?
“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

```
#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?

“upper1” Toward Version 3

Problem:
• The program works, but…
• Deeply nested statements
• No modularity

Solution:
• Handle each state in a separate function

```
#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?

“upper1” Toward Final Version

Problem:
• The program works, but…
• No comments

Solution:
• Add (at least) function-level comments

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
**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 letters. Write the result to stdout. Return 0.

Describes what the function does from caller's viewpoint

---

**“upper1” Final Version**

```c
/* 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;
}
```

```c
/* 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;
}
```

---

**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
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!

Appendix:
Additional DFA Examples

Another DFA Example
Does the string have “nano” in it?
• “banano” ⇒ yes
• “nnnnnnnanOfff” ⇒ yes
• “bananananano” ⇒ yes
• “banananashanana” ⇒ no

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”

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”