A Taste of C

Jennifer Rexford

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 env 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;
  while (c != EOF)
  {
    charCount++;
    c = getchar();
  }
  printf("%d\n", charCount);
  return 0;
}
```

“charcount” Building and Running

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

What is this?
What is the effect?
Building and Running

$ cat somefile
line 1
line 2
$ charcount < somefile
14
$

Building and Running

$ charcount > someotherfile
Line 1
Line 2
^D
$

$ cat someotherfile
14
$

Building and Running in Detail

Question:
• Exactly what happens when you issue the command
gcc217 charcount.c –o charcount

Answer: Four steps
• Preprocess
• Compile
• Assemble
• Link

Preprocessing “charcount”

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

Preprocessor functionality
• Removes comments
• Handles preprocessor directives

Preprocessing “charcount”

#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;
}

charcount.c

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

The result
charcount.i
int getchar();
int printf(char *fmt, ...);
int main(void)
{  int c;
  int charCount = 0;
  c = getchar();
  while (c != EOF)
  {  charCount++;
      c = getchar();
  }  printf("%d\n", charCount);
  return 0;
}

Why int instead
of char?

• C language
• Missing comments
• Missing preprocessor
directives
• Contains code from stdio.h
  • Declarations of getchar() and printf()
  • Missing definitions of
    getchar() and printf()

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()

The result:
charcount.s
.globl main
.type main, @function
main:
  subq $4, %rsp
  movl $0, (%rsp)
call getchar
  cmpl $-1, %eax
  je endloop
  incl (%rsp)
call getchar
  jmp loop
endloop:
  movq $format, %rdi
  movl (%rsp), %esi
  movl $0, %eax
call printf
  movl $0, %eax
  addq $4, %rsp
  ret

• 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

The result:

• 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

The result:

• Machine language
• Contains definitions of getchar() and printf()

Complete! Executable!

Running “charcount”

Command to run:
• charcount < somefile

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

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

Why int instead of char?
# 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;
}

EOF is a special non-char value that getchar() returns to indicate failure.

• Computer calls getchar() again, and repeats

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

Normal execution ⇒ return 0 or EXIT_SUCCESS
Abnormal execution ⇒ return EXIT_FAILURE
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: == !=

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

“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.
```

ASCII

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
<td>106</td>
<td>107</td>
<td>108</td>
<td>109</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>

Partial map

Note: Lower case and upper case letters are 32 apart

EBCDIC

Extended Binary Coded Decimal Interchange Code

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
<td>106</td>
<td>107</td>
<td>108</td>
<td>109</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>

Partial map

Note: Lower case not contiguous; same for upper case
```
#include <stdio.h>
int main(void)
{  int c;
    while ((c = getchar()) != EOF)
        {  if ((c >= 'a') && (c <= 'z'))
            c += 'A' - 'a';
            putchar(c);
        }
    return 0;
}
```

**What’s wrong?**

```
#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;
}
```

**What’s wrong?**

```
#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;
}
```

**What’s wrong?**

```
#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          stdout

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          stdout

upper1

“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"

“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.
```

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

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

**Problem:**
- The program works, but...
- States should have names

**Solution:**
- Define your own named constants
  ```c
  enum Statetype {NORMAL, INWORD};
  ```
- Define an enumeration type
- Define a variable of that type

```c
#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 = NORMAL;
                }
                else
                {
                    putchar(c); state = INWORD;
                }
                break;
        }
    }
    return 0;
}
```

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

**Solution:**
- Handle each state in a separate function

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

**Problem:**
- The program works, but...
- No comments

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

```c
#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)
{
    enum Statetype state = NORMAL;
    while ((c = getchar()) != EOF)
    {
        switch (state)
        {
            case NORMAL:
                state = handleNormalState(c);
                break;
            case INWORD:
                state = handleInwordState(c);
                break;
        }
    }
    return 0;
}
```
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
/*------------------------------------------------------------*/
/* upper1.c                                                   */
/* Author: Bob Dondero                                        */
/*------------------------------------------------------------*/
#include <stdio.h>
#include <ctype.h>
enum Statetype {NORMAL, INWORD};

/*----------------------------------------------------------*/
/* 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;
}

/*----------------------------------------------------------*/
/* 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;
}

/*----------------------------------------------------------*/
/* 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. Write c or its uppercase equivalent to stdout, as specified by the DFA. Return the next 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;
}
```
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
- 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!

Another DFA Example

Does the string have “nano” in it?
- “banano” => yes
- “nnnnnnnanofff” => yes
- “bananananana” => yes
- “bananananasashana” => no

Double circle is accepting state
Single circle is rejecting state

Appendix:

Additional DFA Examples

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”
- “999.99e99”

Invalid literals
- “abc”
- “-e9”
- “1e”
- “+”
- “17.9A”
- “0.38+”
- “=”
- “38.38f9”