CSC 2400: Computer Systems I

Characters in C

Using char for Characters

- Type char can be used for (limited range) arithmetic, but…
- Usually used to store characters – thus the name!
  - Must use a code to map 1-byte numbers to characters
  - Common code: ASCII
- Other ways to represent characters
  - Unicode ("wide" characters) -- 2 bytes
- Format specifier for `printf` is "%c"

The ASCII Code

American Standard Code for Information Interchange

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
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	{	|	}	~	DEL

Lower case: 97-122 and upper case: 65-90
E.g., 'a' is 97 and 'A' is 65 (i.e., 32 apart)

3

char Constants

- C has char constants (sort of) *
- Examples

Constant	Binary Representation (assuming ASCII)	Note
'a'	01100001	letter
'0'	00110000	digit
'\x61'	01100001	hexadecimal form

Use **single** quotes for **char** constant
Use **double** quotes for **string** constant

* Technically 'a' is of type int; automatically truncated to type char when appropriate

4

More char Constants

- Escape characters

Constant	Binary Representation (assuming ASCII)	Note
'\a'	00000111	alert (bell)
'\b'	00001000	backspace
'\f'	00001100	form feed
'\n'	00001010	newline
'\r'	00001101	carriage return
'\t'	00001001	horizontal tab
'\v'	00001011	vertical tab
'\\'	01011100	backslash
'\?'	00111111	question mark
'\''	00100111	single quote
'\"'	00100010	double quote
'\0'	00000000	null

Used often

5

Reading and Writing a Character

- Subset of C I/O functions:

Task	Example Function Calls
Write a char	`int status;` `status = putchar('a');  /* Writes to stdout */`
Read a char	`int c;` `c = getchar();  /* Reads from stdin */`

```
#include <stdio.h>

int main(void) {
   int c;
   c = getchar();
   if (c != EOF) {
      if ((c >= 'a') && (c <= 'z'))
         c = c + 'A' - 'a';
      putchar(c);
   }
}
```

'a' is 97
'A' is 65

EOF is:
CTRL-D in Unix
CTRL-Z in Windows

6

The "End-of-File Character"

- Files do not end with the "EOF character"
  - Because there is no such thing!!!
- EOF is:
  - A special **non-character** value returned by getchar() and related functions to indicate failure
  - #defined in stdio.h; typically as -1

  - In Unix, use CTRL-D to insert EOF into the input stream
  - In Windows, use CTRL-Z to insert EOF into the input stream

7

Using EOF

- Correct code

```
int c;
c = getchar();
while (c != EOF) {
   …
   c = getchar();
}
```

getchar() returns int because:
- int is the computer's natural word size
- getchar() must be able to return all valid chars **and EOF**

- Equivalent idiom

```
int c;
while ((c = getchar()) != EOF) {
   …
}
```

An expression of the form
x = y
assigns to x, and evaluates to the new value of x

- Incorrect code

```
char c;
while ((c = getchar()) != EOF) {
   …
}
```

What if stdin contains the 11111111 (ÿ) character?

8

Practice Now

- Log into your Unix account.
- Create a directory called csc2400
- Change your current directory to csc2400
- Write a C program echo.c that reads characters in a loop until EOF is encountered, and prints them out, one character per line. Sample execution:

Input

```
aB12
c
^Z
```

Output

```
[a]
[B]
[1]
[2]
[
]
[c]
```

9

Brief Introduction to Finite State Automata

(not in the textbook)

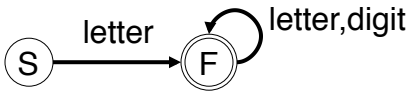
Finite Automata

- Finite State Automaton
  a.k.a. Finite Automaton, Finite State Machine, FSA or FSM
- Used to develop all kinds of software
- Neat way to describe algorithms

- Easy to understand by example

11

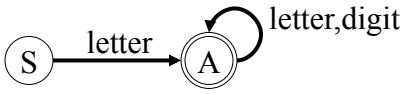
Example FSA



- Nodes are *states*.
- Arcs are *transitions.*
- Labels on arcs tell what causes the transition
  - The single edge labeled "letter“ stands for 52 edges labeled 'a', 'b', ..., 'z', 'A', ..., 'Z'.(Similarly for “digit“)
- S is the *start state* (which is unique).
- F is a *final state*. By convention, final states are drawn using a double circle. There can be more than one final state.

12

Applying FSA to Input



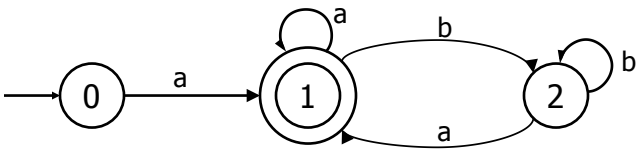
aA23
Z2aFop

~~7AbeR6~~
~~B343P?~~

- The FSA starts in its start state.
- If there is a edge out of the current state whose label matches the current input character, then the FSA moves to the state pointed to by that edge, and "consumes" that character; otherwise, it gets stuck.
- The finite-state automata stops when it gets stuck or when it has consumed all of the input characters.
- An input string is *accepted* by a FSA if:
  - The entire string is consumed (the machine did not get stuck) AND
  - the machine ends in a final state.

13

FSA Example 1

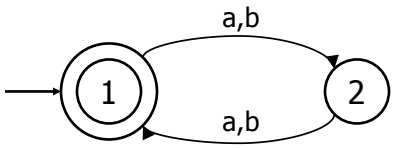


- What types of strings does this FSA recognize?

Input String	Result
abaaabbbaaaba	Accept
aabbaaabbaabbb	Reject
aabba	Accept

14

FSA Example 2



- What types of strings does this FSA recognize?

Input String	Result
aabbaaba	Accept
aba	Reject
abaa	Accept

15

Exercise 1

- Write a finite-state automata that accepts Java identifiers (one or more letters, digits, underscores, dollar signs; not starting with a digit).

16

Exercise 2

- Write a finite-state automata that accepts Java identifiers that do not end with an underscore.

17

DFA to Code

current_state=S (start state)

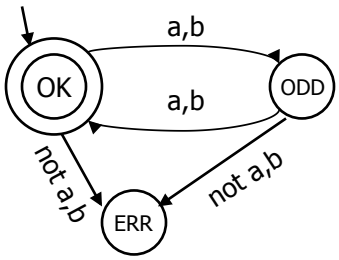
Repeat:
  read next character
  use DFA to update current_state
Until
  machine gets stuck (reject) or entire input is read

If current_state == one of final states Accept
Else Reject

18

DFA to C Code

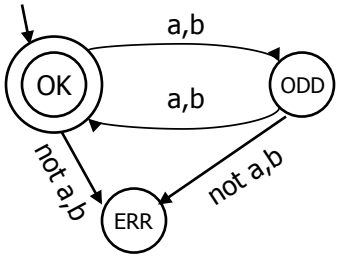
- Step 1: Define the states, initialize current state.




```
#define OK 1

_____________________


_____________________

int state = OK;
```

DFA to C Code

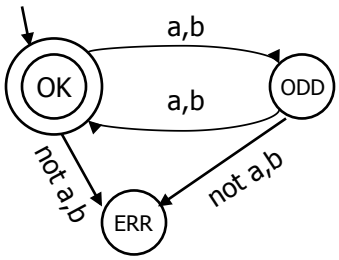
- Step 2: Read characters one by one until EOF




```
int main()
{
  int c = getchar();

  while ((c != EOF) && (state != ERR))
  {
    /* Process character */




    c = getchar();
  }
}
```

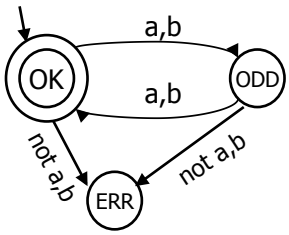
DFA to C Code

- Step 3: Implement state transitions




```
int main()
{
  int c = getchar();
  while ((c != EOF) && (state != ERR))
  {
    switch(state) {
      case OK:  /* update state */


                break;
      case ODD: /* update state */


                break;
      case ERR: break;
    }
    c = getchar();
  }
}
```

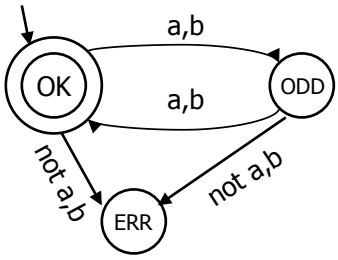
DFA to C Code




```
#define OK 1
#define ODD 2
#define ERR 3

int state = OK;

int main()
{
  int c = getchar();

  while ((c != EOF) && (state != ERR)) {
    switch(state) {
      case OK:  if((c == 'a') || (c == 'b'))
                  state = ODD;
                else state = ERR;
                break;
      case ODD: if((c == 'a') || (c == 'b'))
                  state = OK;
                else state = ERR;
                break;
    }
    c = getchar();
  }
  /* Input consumed */
  if (state == ERR) printf("Rejected \n");
  if (state == OK)  printf("Accepted \n");
  return 0;
}
```

Practice Now

- Log into your Unix account.
- Change your current directory to csc2400
- Write a C program fsa.c that implements this DFA:



- Print out "String accepted" if the input string is valid, and "String rejected" otherwise.

```
printf("String accepted\n");
```

23

Required Reading

- *Textbook, Sections 4.1 – 4.3, Chapter 4*

24