CSC 2400: Computer Systems

Preprocessor
and
Finite State Automata

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"

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

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

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?

Practice Now

- Log into your Unix account.
- 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]
```

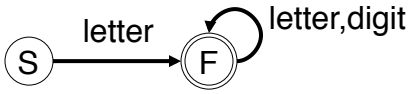
Brief Introduction to Finite State Automata

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

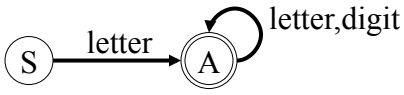
- Easy to understand by example

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.

Applying FSA to Input

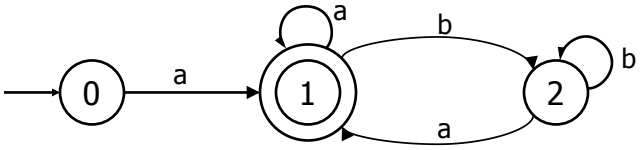


aA23
Z2aFop

~~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.

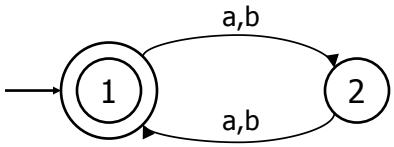
FSA Example 1



- What types of strings does this FSA recognize?

Input String	Result
abaaabbbaaaba	Accept
aabbaaabbaabbb	Reject
aabba	Accept

FSA Example 2



- What types of strings does this FSA recognize?

Input String	Result
aabbaaba	Accept
aba	Reject
abaa	Accept

Exercise 1

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

Exercise 2

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

DFA to Code

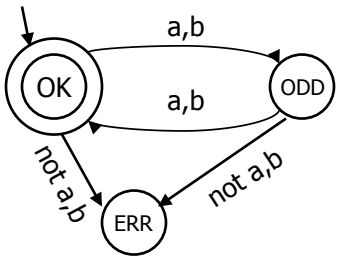
current_state = S (start state)

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

DFA to C Code

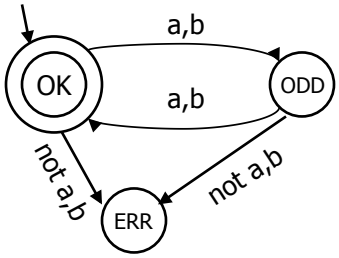
- Step 1: Define the states, initialize current state.




```
#define OK 1

_____________________


_____________________

int state = OK;
```

DFA to C Code

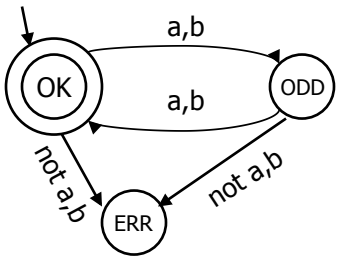
- Step 2: Read characters one by one until EOF




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

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




    c = getchar();
  }
}
```

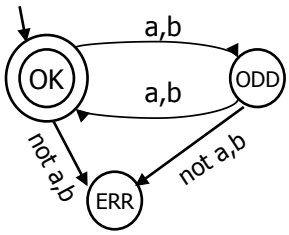
DFA to C Code

- 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();
  }
}
```

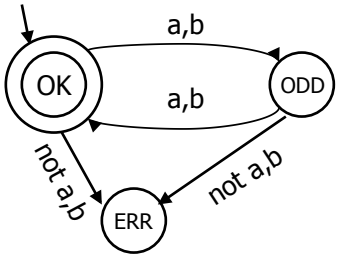
DFA to C Code




```
#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 != OK) 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");
```

Preprocessor:

First stage of the compiler

Example: “Circle” Program

File circle.c:

```
#include <stdio.h>

int radius;

int main()

/* Read a circle's radius from stdin, and compute and write its
   diameter and circumference to stdout.  Return 0. */

{
   int diam;
   double circum;
   printf("Enter the circle's radius:\n");
   scanf("%d", &radius);
   diam = 2 * radius;
   circum = 3.14159 * (double)diam;
   printf(”Circle with radius %d, diameter %d\n”, radius, diam);
   printf("and circumference %f.\n", circum);
   return 0;
}
```

The Preprocessor's View

File circle.c

Preprocessor directive
Preprocessor replaces with contents of file /usr/include/stdio.h

```
#include <stdio.h>
int radius;
int main()
/* Read a circle's radius from stdin, and compute and write its
   diameter and circumference to stdout.  Return 0. */
{
   int diam;
   double circum;
   printf("Enter the circle's radius:\n");
   scanf("%d", &radius);
   diam = 2 * radius;
   circum = 3.14159 * (double)diam;
   printf("Circle with radius %d, diameter %d\n", radius, diam);
   printf("and circumference %f.\n", circum);
   return 0;
}
```

Comment
Preprocessor removes

Results of Preprocessing

Declarations of printf(), scanf(), and other functions; compiler will have enough information to check subsequent function calls

```
int printf(char*, …);
int scanf(char*, …);
…
int radius;
int main()
{
   int diam;
   double circum;
   printf("Enter the circle's radius:\n");
   scanf("%d", &radius);
   diam = 2 * radius;
   circum = 3.14159 * (double)diam;
   printf("Circle
   printf("and ci
   return 0;
}
```

Where are the **DEFINITIONS** of printf() and scanf() ?

What is Wrong?

```
#include <stdio.h>

int main()
{
   int result;

   result = SomeFunction(5);

   return 0;
}

int SomeFunction(int x)
{
   return x * x;
}
```