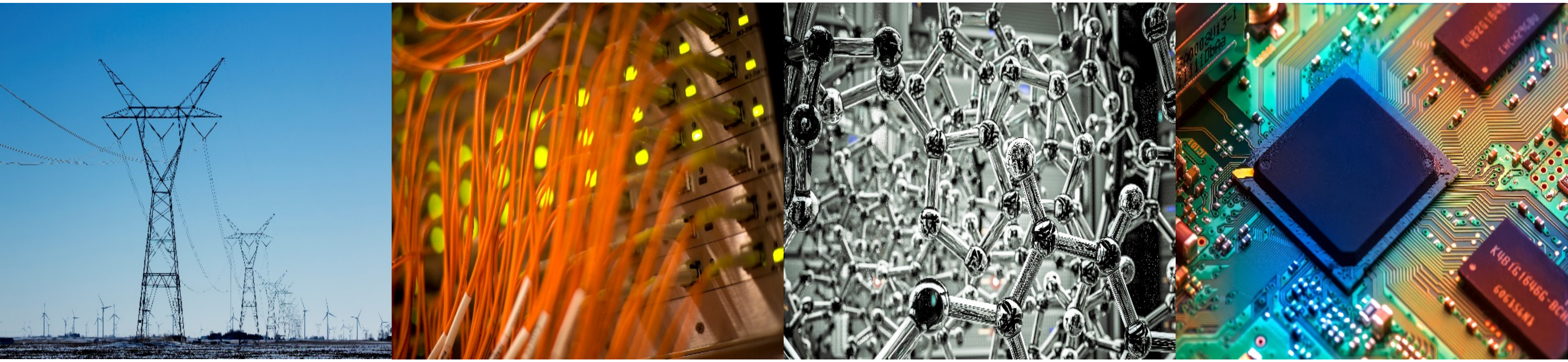
ECE 220 Computer Systems & Programming

Lecture 22 – Recursion with Backtracking, File I/O

July 16, 2020



ILLINOIS

Electrical & Computer Engineering

GRAINGER COLLEGE OF ENGINEERING

- MT2 past exam & practice questions posted
- Informal Early Feedback

Recursion with Backtracking Template

```
bool solve(configuration conf){
   if(no more choices) /*base case*/
      return (conf is goal state);

   for(all available choices){
      try one choice c;
      /*recursively solve after making choice*/
      ok = solve(conf with choice c made);
      if(ok)
         return true;
      else
         unmake choice c;
   }

   return false; /*tried all choices but no solution found*/
}
```

```
int solve_nqueen(int board[N][N], int row){

      /*base case - reach solution, no more rows to place Q*/
      if(row >= N)
            return 1;

      /*recursive case with backtracking*/
      int i
      for(i=0;i<N;i++){ /* try each column */









      }
      return 0;
}
```

Recursion with Backtracking Summary

You are presented with some options to solve a problem; you choose one and then a new set of options emerge. This procedure repeats. If you made a sequence of “good” choices, then eventually you will reach the goal state. If you didn’t, then you need to backtrack to unmake previous choice(s) to reach the goal state.

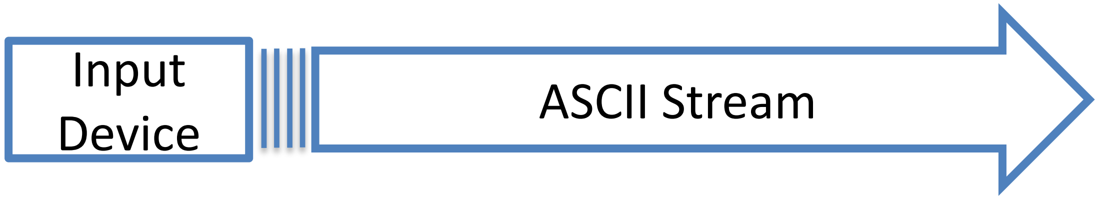
Our goals:

- Looking for a solution
- Looking for all solutions
- Looking for the best solution

Examples:

- Sudoku
- N-Queen
- Permutation
- Maze

Input / Output Streams



```
scanf("%d", &x)
```

I/O Device operates using I/O protocol (such as memory mapped I/O)

In C, we abstract away the I/O details to an I/O function call

Stream Abstraction for I/O

All character-based I/O in C is performed on **text streams**.

A stream is a **sequence of ASCII characters**, such as:

- the sequence of ASCII characters printed to the monitor by a single program
- the sequence of ASCII characters entered by the user during a single program
- the sequence of ASCII characters in a single file

Characters are processed in the order in which they were added to the stream.

- e.g., a program sees input characters in the same order as the user typed them.

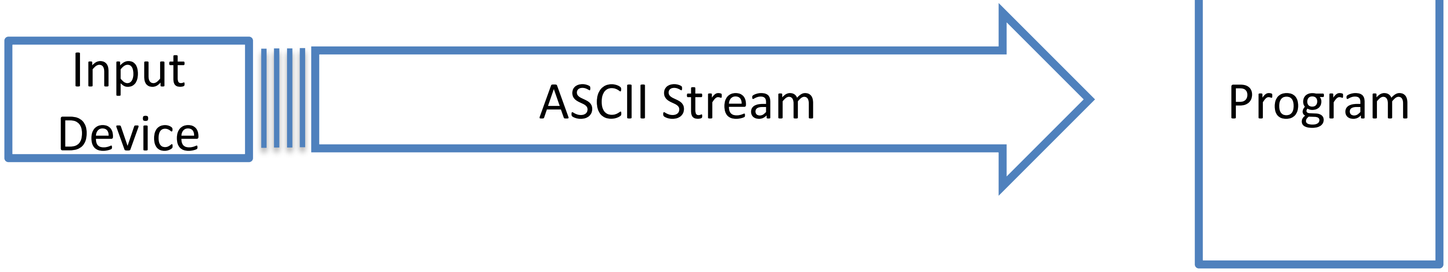
Standard Streams:

Input (keyboard) is called **stdin**.

Output (monitor) is called **stdout**.

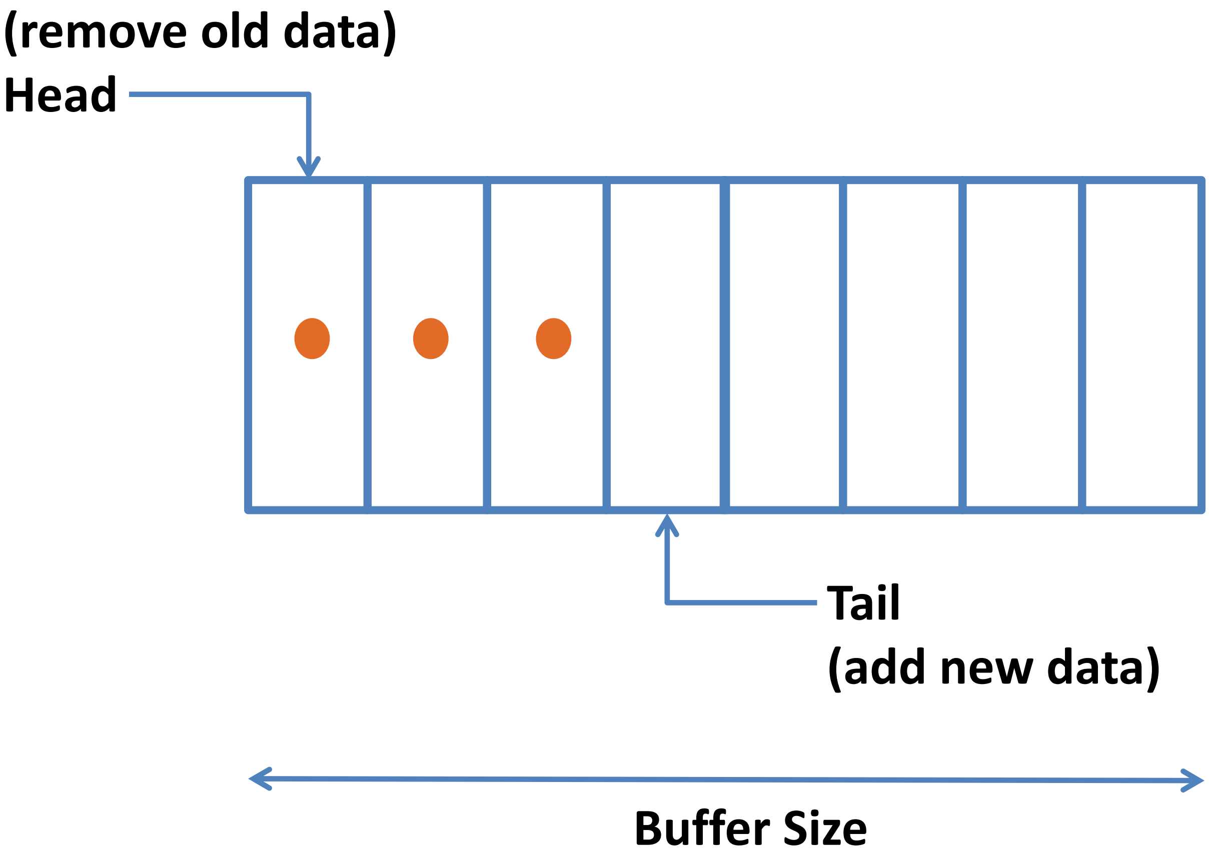
Error (monitor) is called **stderr**.

Stream Buffering



- Input device is the producer; Program is the consumer
- We want producer and consumer to be operating independently
- Why??? Think Netflix over spotty internet connection
- We can accomplish that via buffering

Simple Buffer



- Producer adds data at Tail
- Consumer removes data from Head
- Buffer Full?
- Buffer Empty?
- Concept of circular buffer
- Also called First in, First Out (FIFO) or Queue

I/O Functions in C

The standard I/O functions are declared in the <stdio.h> header file.

Function	***Description***
putchar	Displays an ASCII character to the screen.
getchar	Reads an ASCII character from the keyboard.
printf	Displays a formatted string.
scanf	Reads a formatted string.
fopen	Open/create a file for I/O.
fclose	Close a file for I/O.
fprintf	Writes a formatted string to a file.
fscanf	Reads a formatted string from a file.
fgetc	Reads next ASCII character from stream.
fputc	Writes an ASCII character to stream.
fgets	Reads a string (line) from stream.
fputs	Writes a string (line) to stream.
EOF & feof	End of file

How to use these I/O functions

```
/* Open/create a file for I/O */
FILE* fopen(char* filename, char* mode)          /* mode: “r”, “w”, “a”, … */
            success-> returns a pointer to FILE
            failure-> returns NULL

/* Close a file for I/O */
int fclose(FILE* stream)
            success-> returns 0
            failure-> returns EOF (Note: EOF is a macro, commonly -1)

/* Writes a formatted string to a file */
int fprintf(FILE* stream, const char* format, …)
            success-> returns the number of characters written
            failure-> returns a negative number

/* Reads a formatted string from a file */
int fscanf(FILE* stream, consta char* format, …)
            success-> returns the number of items read; 0, if pattern doesn’t match
            failure-> returns EOF
```

/* Reads next ASCII character from stream */
int fgetc(FILE* stream)

success-> returns the character read
failure-> returns EOF and sets end-of-file indicator

/* Writes an ASCII character to stream */
int fputc(int char, FILE* stream)

success-> write the character to file and returns the character written
failure-> returns EOF and sets end-of-file indicator

/* Reads a string (line) from stream */
char* fgets(char* string, int num, FILE* stream)

success-> returns a pointer to string
failure-> returns NULL

/* Writes a string (line) to stream */
int fputs(const char* string, FILE* stream)

success-> writes string to file and returns a non-negative value
failure-> returns EOF and sets the end-of-file indicator

/* checks end-of-file indicator */
int feof(FILE* stream)

if at the end of file-> returns a non-zero value
if not -> returns 0

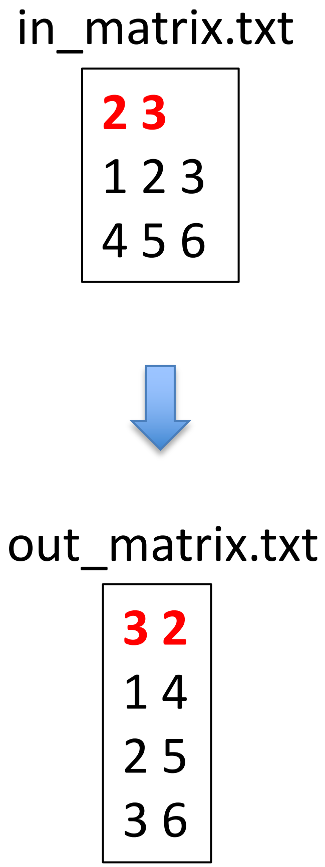
Exercise: *Read an mxn matrix from file in_matrix.txt and write its transpose to file out_matrix.txt.* ***The first row of the file specifies the size of the matrix.***

Hint: *use fscanf to read from a file and use fprintf to write to a file.*

```
#include <stdio.h>
int main(){
    FILE *in_file;
    FILE *out_file;

    /* opening in_matrix.txt for read */
    in_file = fopen("in_matrix.txt", "r");
    if(in_file == NULL)
        return -1;

    /* reading matrix dimensions from file */
    int m, n;
    fscanf(in_file, "%d %d", &m, &n);
    int matrix[m][n];
```



in_matrix.txt

```
2 3
1 2 3
4 5 6
```

out_matrix.txt

```
3 2
1 4
2 5
3 6
```

```
    /* open out_matrix.txt file for write*/
    out_file = fopen("out_matrix.txt", "w");
    if(out_file == NULL)
        return -1;

    /* writing transposed matrix dimensions to file */
    fprintf(out_file, "%d %d\n", n, m);




    return 0;
}
```