LAB 3

COMMON MISTAKES & RECOMMENDATIONS FROM QUIZ 1-2

TYPEDEF

- If we are given a struct as such:

```
struct _node  {
        int data;
        struct _node * next;
};
```

- We can add:

```
typdef struct _node Node;
```

- Alternatively,

```
typdef struct _node {
        int data;
        struct _node * next;
} Node;
```

- When we do this, our valid types are `struct _node` and `Node`.
- `struct Node` is invalid because `Node` expands to `struct _node`

EASIER WAY OF READING FILE

- feof(inputFile) Prone to reading file too far, but also gives us a little more access to error handling
- fgets(str, STR_COUNT, inputFile);
  - Strtok
  - Custom parsing function
- We can use it as such:

```
while (fgets(str, STR_COUNT, inputFile)) { … }
```

- Or:

```
char * str;
str = fgets(str, STR_COUNT, inputFile);
while (str) { … ; str = fgets(str, STR_COUNT, inputFile); }
```

WHEN WE ARE READING IN DATA WITH **KNOWN** FORMAT (CSV)

```
char * token;
while (fgets(…)) {
      token = strtok(str, “,”); // or a strsep variant for C89/99 Standard
      // handle token
      token = strtok(NULL, “,”); // or a strsep variant for C89/99 Standard
      // handle token
      // Repeat last two lines until all fields are read.
}
```

WHEN WE ARE READING IN DATA WITH **UNKNOWN** FORMAT (CSV) [1/3]

```
typdef struct _headerField {
	char * name;
	size_t index;
} Header;

typdef struct _headerNode {
	Header * header;
	struct _headerNode * pNext;
	struct _headerNode * pLast;
} Node;
```

WHEN WE ARE READING IN DATA WITH **UNKNOWN** FORMAT (CSV) [2/3]

```
Node * getHeaders(FILE * infile) {
	Node * headers = NULL;
	char * string = (char*)malloc(100000);
	fgets(string, 100000, infile);
	char * token = strsep(string, “,”);
	while (token != NULL) {
		if (!headers) headers = makeNode(token, 0);
		else insertAtEnd(&headers, token);
		token = strsep(NULL, “,”);
	}
	return headers;
}
```

WHEN WE ARE READING IN DATA WITH **UNKNOWN** FORMAT (CSV) [3/3]

```
Node * headers = getHeaders(infile);
Matrix * matrix = initMatrix();
Row * row = initRow();
char * str = (char *) malloc(100000);
char * token;
while (fgets(str, 100000, infile)) {
	token = strsep(token, “,”);
	while (token != NULL) {
		row.insert(token); // This will insert the token into the row
		token = strsep(NULL, “,”);
	}
	insertRow(&matrix, row); // This will insert row into matrix
	resetRow(&row); // This will reset row back to NULL;
}
```

IN BOTH CASES, WE DO NOT USE A SERIES OF IF STATEMENTS OR SWITCH STATEMENT

- Note that we should not use an if statement or switch statement to insert the new data in a for/while loop
- If we use if statements in a while or for loop, then we waste a lot of resources
  - If we have 100 cases, then we 100 checks
    - When we insert at the last position, we check 100 conditions, a total of 5,050 cmps
    - We could complete this in a total of 199 operations instead
    - This could cut as many as 300 lines (depending on your coding style)
- This works, but it is not efficent

EXAM REVIEW

POSSIBLE CODING PROBLEMS

- Be sure you can:
  - Copy stacks
  - Copy lists
  - Make and destroy lists stacks
  - Perform operations on lists and stacks
    - max, min, avg, longest string, shortest string, avg string length
    - insertAtEnd, removeAtN, removeAtEnd, insertAtBeginning, removeAtBeginning, …
  - Understand dynamic memory
  - Understand when to apply linked lists and stacks

[1 PT] WHAT IS WRONG WITH THIS CODE?

```
char pop(Stack * s) {
	char tmp_val;

	tmp_val = s->pHead->data;

	Node * tmp_ptr = s->pHead->pNext;

	s->pHead = tmp_ptr;

	return tmp_val;
}
```

```
typdef struct node
{
	char data;
	struct node *pNext;
} Node;
```

[1 PT] WHAT IS WRONG WITH THIS CODE? [SOLN]

```
char pop(Stack * s) {
        char tmp_val;

        tmp_val = s->pHead->data;

        Node * tmp_ptr = s->pHead->pNext;

        free(s->pHead);

        s->pHead = tmp_ptr;

        return tmp_val;
}
```

```
typdef struct node
{
        char data;
        struct node *pNext;
} Node;
```

[1 PT] WHAT IS WRONG WITH THIS CODE?

```
int insertAtEnd (struct node **pStart, char *pNewData) {
        Node * newNode = malloc(sizeof(Node));
        if (newNode != NULL) { return 0; }
        int size = (strlen(pNewData) + 1);
        newNode->pData = calloc(sizeof(char) * size);
        if (newNode->pData == NULL) { return 0; }
        strncpy(newNode->pData, pNewData, sizeof(Node));
        Node * cur = *pStart;
        for (;cur->pNext != NULL;cur=cur->pNext);
        newNode->pPrev = cur;
        cur->pNext = newNode;
        newNode->pNext = NULL;
        return 0;
}
```

```
typdef struct node
{
    char data;
    struct node *pNext;
    struct node *pPrev;
} Node;
```

[1 PT] WHAT IS WRONG WITH THIS CODE? [SOLN]

```
int insertAtEnd (struct node **pStart, char *pNewData) {
        Node * newNode = malloc(sizeof(Node));
        if (newNode == NULL) { return 0; }
        int size = (strlen(pNewData) + 1);
        newNode->pData = malloc(sizeof(char) * size);
        if (newNode->pData == NULL) { return 0; }
        strncpy(newNode->pData, pNewData, size);
        Node * cur = *pStart;
        for (;cur->pNext != NULL;cur=cur->pNext);
        newNode->pPrev = cur;
        cur->pNext = newNode;
        newNode->pNext = NULL;
        return 0;
}
```

```
typdef struct node
{
    char data;
    struct node *pNext;
    struct node *pPrev;
} Node;
```

LAB 3

VERY SIMILAR TO LAST LAB

- Now, we have a doubly linked list
  - You need pNext and pLast
  - Copy/paste from last week!
  - You can finish this quickly if you use last week's code
- Write test functions this time
- Be sure to use whiteboards or scratch paper!
- I am not going to help *as much* as last time since you are expected to this on Friday

UPCOMING ASSIGNMENTS

- PA 2 Due tomorrow
  - Submit what you have – even if it is empty
    - Any submission = 100%

- Midterm Exam 1
  - Friday
  - Normal Lec Time – don’t be late