CSE 3100: Systems Programming

Complete Lecture Notes

Lecture 1: Course Introduction and First C Program

1. Systems Programming Overview

- **Definition**: Writing low-level software that interacts with hardware or operating systems.
- **Topics Covered**:
  - C Programming
  - Process Management
  - Concurrency
  - Memory Management

2. First C Program

```
#include <stdio.h>

int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

- `#include <stdio.h>` : Includes standard input/output functions.
- `main()` : Entry point of the program.
- `printf()` : Prints text to the console.
- `return 0;` : Indicates successful execution.

3. Compilation and Execution

```
cc hello.c -o hello  # Compile
./hello              # Run
```

- **Compilation steps**:
  1. **Preprocessing** (`#include` handling)
  2. **Compilation** (translating to assembly)
  3. **Linking** (combining with libraries)

Lecture 2: Expressions and Basic Data Types

1. Data Types

Type	Size (bytes)	Description
`char`	1	Single character
`int`	4	Integer value
`float`	4	Floating-point number
`double`	8	High-precision float

2. Operators

Category	Operators
Arithmetic	`+`, `-`, `*`, `/`, `%`
Comparison	`==`, `!=`, `<`, `>`
Logical	`&&`,`
Assignment	`=`, `+=`, `-=`, `*=`, `/=`

3. Increment and Decrement

```
int x = 5;
int y = ++x; // Pre-increment (x = 6, y = 6)
int z = x--; // Post-decrement (x = 5, z = 6)
```

Lecture 3: Control Flow

1. If-Else Statements

```
if (x > 0) {
    printf("Positive");
} else {
    printf("Negative");
}
```

2. Loops

While Loop

```
while (x < 5) {
    printf("%d", x);
    x++;
}
```

For Loop

```
for (int i = 0; i < 5; i++) {
    printf("%d", i);
}
```

Lecture 4: Functions and Global Variables

1. Function Basics

```
int add(int a, int b) {
    return a + b;
}
```

2. Function Prototypes

```
int square(int x);
```

3. Static Variables

```
void counter() {
    static int count = 0;
    count++;
    printf("%d", count);
}
```

Lecture 5: Arrays and Pointer Basics

1. Arrays

```
int arr[5] = {1, 2, 3, 4, 5};
```

2. Strings

```
char str[] = "Hello";
```

3. Pointers

```
int x = 10;
int *p = &x;  // Pointer stores address of x
printf("%d", *p); // Dereference pointer
```

Lecture 6: Memory Allocation

1. Dynamic Memory Allocation

```
int *p = (int *)malloc(10 * sizeof(int));
free(p);
```

2. Resizing Memory

```
p = realloc(p, 20 * sizeof(int));
```

Lecture 7: More on Memory, Pointers, and Structures

1. Structures

```
typedef struct {
    char name[50];
    int id;
    float gpa;
} Student;
```

2. Structure Pointers

```
Student s = {"Alice", 101, 3.5};
Student *p = &s;
printf("%s", p->name);
```

Lecture 8: Linked Lists, Enums, and Function Pointers

1. Linked List Basics

```
typedef struct Node {
    int data;
    struct Node* next;
} Node;
```

2. Function Pointers

```
int add(int a, int b) { return a + b; }
int (*funcPtr)(int, int) = add;
```

3. `qsort()` Example

```
int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}
```

Lecture 9: Input/Output and Files

1. File Handling

```
FILE *fp = fopen("file.txt", "r");
fclose(fp);
```

2. Reading a File

```
char buffer[100];
fgets(buffer, 100, fp);
```

3. Writing to a File

```
fprintf(fp, "Hello, World!");
```

4. Moving File Pointer

```
fseek(fp, 0, SEEK_END);
long size = ftell(fp);
```

Final Notes

- Understand pointers and memory allocation.
- Practice function pointers and linked lists.
- Master file handling for real-world applications.

 Mastering these concepts will make you a strong C programmer!