Introduction to C

Data Structures and Algorithms
CSE 102

Agenda

Goal: Provide foundational C programming knowledge for understanding DSA

- Basics of C
- Control Structures
- Functions
- Pointers
- Arrays
- Structs and Memory Layout
- Dynamic Memory Allocation

Why C for DSA?

- Low-level memory access (pointers)
- Control over system resources
- Widely used for algorithm design and system programming
- C as the foundation for many modern languages.

The Building Blocks of C

- Components
  - Keywords
  - Variables and Data Types (int, float, char, etc.)
  - Basic Input/Output (e.g., printf and scanf)
- Example Code

```c
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("You entered: %d\n", num);
    return 0;
}
```

Compilation and Execution

- Compile code with GCC in a terminal window

```
gcc -g program.c -o program
```

- Run code

```
./program
```

Command Line Arguments

- The main() function accepts command line arguments
  - int argc: no. of arguments
  - char **argv: list of arguments
  - argv[0] always contains the program name with path
- Example

```c
#include <stdio.h>
int main(int argc, char **argv) {
    printf("No.  of arguments: %d\n", argc);
    for(int i=0; i<argc; i++)
    {
        printf("Arg %d: %s\n", i, argv[i]);
    }
    return 0;
 }
```

Control Structures

- Conditional Statements
  - if, else if, else

```c
int num = 3;
if (num > 0) {
    printf("The number is positive.\n");
} else if (num < 0) {
    printf("The number is negative.\n");
} else {
    printf("The number is zero.\n");
}
```

- Loops: for, while, do-while

```c
int sum = 0;
for (int i = 1; i <= N; i++) {
    sum += i;
}
```

Functions

- Why Functions?
  - Reusability, modularity, readability.
- Syntax

```
return_type function_name(parameters) {
    // code
    return value;
}
```

- Example: Factorial using recursion

```
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

Pointers

- What are pointers?
  - Variables that store memory addresses

- Syntax

```
int a = 10;
int *p = &a;  // p stores the address of a
```

- Importance in DSA:
  - Used in dynamic memory allocation, linked lists, trees, etc.