2. Node Structure

2.1 Defining a Node

In C, a node is typically defined using a **structure**:

```
// Definition of a node
struct Node {
    int data;           // Data part
    struct Node *next;  // Pointer to next node
};
```

Memory Layout:

```
Single Node in Memory:

┌─────────┬─────────┐
│  data   │  next   │
│ (int)   │(pointer)│
└─────────┴─────────┘
   4 bytes    8 bytes (on 64-bit system)
```

2.2 Creating a Node

Method 1: Static Allocation (Limited)

```
struct Node node1;
node1.data = 10;
node1.next = NULL;
```

Method 2: Dynamic Allocation (Recommended)

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
```

```
    struct Node *next;
};

int main() {
    // Allocate memory for new node
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));

    // Check if allocation successful
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    // Initialize node
    newNode->data = 10;
    newNode->next = NULL;

    printf("Node created with data: %d\n", newNode->data);

    // Free memory when done
    free(newNode);

    return 0;
}
```

2.3 The Arrow Operator (->)

When working with pointers to structures, use the arrow operator:

```
struct Node *ptr;

// These are equivalent:
ptr->data = 10;      // Arrow operator (preferred)
(*ptr).data = 10;    // Dereference then dot operator
```

Why use ->?

- More readable
- Less typing
- Standard convention in C

Revision #1
Created 2025-10-27 05:00:59 UTC by DS
Updated 2025-10-27 05:01:24 UTC by DS