Introduction to C Language

Introduction to the C Language

The C language is a low-level imperative programming language that allows fine-grained memory management.
It is widely used in embedded systems development, operating systems, and applications requiring performance optimization.

1. Variables and Data Types

In C, each variable must be declared before being used. Here are some basic data types:

- **int**: integer (usually 4 bytes).
- **float**: floating-point number (4 bytes).
- **double**: double-precision floating-point number (8 bytes).
- **char**: a single character (1 byte).
- **void**: empty type, used for declaring pointers or functions without a return value.

2. Pointers

A pointer is a variable that stores the memory address of another variable. It is a central concept in C.

Declaring a pointer:

```c
int *ptr;
```

```

Assigning an address to a pointer:

```c
int a = 10;
int *ptr = &a;
```

Dereferencing a pointer:

```c
printf("%d", *ptr);  // Prints the value of 'a'
```

3. Arrays

An array is a data structure that allows storing multiple elements of the same type.

Declaring an array:

```c
int arr[5];  // An array of 5 integers
```

Accessing elements:

```c
arr[0] = 10;
printf("%d", arr[0]);  // Prints 10
```
```

4. Strings

In C, strings are arrays of type `char` terminated by a null character (`\0`).

Declaring a string:

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

5. Structures

Structures allow grouping multiple variables under the same name. They are used to represent more complex objects.

Declaring a structure:

```c
struct Person {
    char name[50];
    int age;
};
```

Accessing members:

```c
struct Person p1;
p1.age = 30;
```

strcpy(p1.name, "John");
```

#### 6. Linked Lists

A linked list is a data structure where each element (or "node") contains a value and a pointer to the next element.

Declaring a simple linked list:
```c
struct Node {
    int data;
    struct Node* next;
};
```

Adding an element to the head of the list:
```c
struct Node* addNode(struct Node* head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    return newNode;
}
```

7. Memory Management
```

In C, memory management is manual. Here are the main functions:

- `malloc`: allocates a block of memory.
- `free`: frees allocated memory.

Example:

```c
int* ptr = (int*)malloc(sizeof(int));
*ptr = 100;
free(ptr);
```

8. Functions

In C, a function must be declared before being called. Here is an example of a simple function that returns the sum of two integers:

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

9. Loops and Conditions

`for` loop:

```c
```

```c
for(int i = 0; i < 10; i++) {
    printf("%d\n", i);
}
```

`while` loop:

```c
int i = 0;
while(i < 10) {
    printf("%d\n", i);
    i++;
}
```

`if` condition:

```c
if(a > b) {
    printf("a is greater than b");
} else {
    printf("b is greater or equal to a");
}
```

The C language is fundamental for understanding low-level programming and manual memory management.

These concepts help develop efficient and optimized software.