2. Pointer Basics

2.1 Declaring Pointers

Syntax:

```
data_type *pointer_name;
```

Examples:

```
int *ptr;        // Pointer to an integer
float *fptr;     // Pointer to a float
char *cptr;      // Pointer to a character
double *dptr;    // Pointer to a double
```

Important Notes:

- The `*` (asterisk) indicates that the variable is a pointer
- The asterisk can be placed next to the type or the variable name
- All three declarations below are equivalent:

  ```
  int *ptr;
  int* ptr;
  int * ptr;
  ```

- Convention: Most C programmers use `int *ptr` style

Multiple Pointer Declaration:

```
int *p1, *p2, *p3;    // Three pointers to int
int *p1, p2, *p3;     // p1 and p3 are pointers, p2 is int
int* p1, p2, p3;      // Only p1 is pointer! p2 and p3 are int
```

2.2 Pointer Operators

There are two main operators for working with pointers:

Operator	Name	Description	Example

`&`	Address-of	Gets the memory address of a variable	`&variable`
`*`	Dereference	Accesses the value at the address stored in pointer	`*pointer`

2.3 Initializing Pointers

Method 1: Initialize with address of existing variable

```
int num = 42;
int *ptr = &num;  // ptr now points to num
```

Method 2: Initialize to NULL

```
int *ptr = NULL;  // Pointer points to nothing (safe initialization)
```

Method 3: Uninitialized (DANGEROUS)

```
int *ptr;  // Contains garbage value - DO NOT USE until initialized!
```

Visual Representation:

```
Memory Layout:

Variable: num = 42
Address:  0x1000
          ┌──────┐
0x1000:   │  42  │  num
          └──────┘

Variable: ptr
Address:  0x2000
          ┌────────┐
0x2000:   │ 0x1000 │  ptr (points to num)
          └────────┘
```

2.4 Using Pointers - The & and * Operators

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

```c
int main() {
    int num = 100;
    int *ptr;

    ptr = &num;  // Store address of num in ptr

    printf("Value of num: %d\n", num);           // Direct access
    printf("Address of num: %p\n", (void*)&num); // Address of num
    printf("Value of ptr: %p\n", (void*)ptr);    // Address stored in ptr
    printf("Value pointed to by ptr: %d\n", *ptr); // Dereference ptr

    // Modify through pointer
    *ptr = 200;

    printf("\nAfter *ptr = 200:\n");
    printf("Value of num: %d\n", num);           // num changed!
    printf("Value pointed to by ptr: %d\n", *ptr);

    return 0;
}

/* Output:
Value of num: 100
Address of num: 0x7ffd5c8e4a3c
Value of ptr: 0x7ffd5c8e4a3c
Value pointed to by ptr: 100

After *ptr = 200:
Value of num: 200
Value pointed to by ptr: 200
*/
```

Understanding the Operations:

```c
int num = 42;
int *ptr = &num;

// These are equivalent:
num = 100;     // Direct modification
*ptr = 100;    // Indirect modification through pointer
```

```
// Both operations change the same memory location
```

Revision #1
Created 2025-10-01 03:25:09 UTC by DS
Updated 2025-10-01 03:25:36 UTC by DS