```
#include <iostream>
using namespace std;

int main()
{
    cout<<"1d int array"<<endl;
    int num_1d[] = {1, 2, 3, 4, 5};
    cout<<num_1d<<endl;  //address for first element(pointer to array)
    cout<<&num_1d<<endl;   //address for first element(pointer to pointer to array)
    cout<<*num_1d<<endl;   //1

    cout<<(unsigned long long)(num_1d)<<endl;
    cout<<(unsigned long long)(num_1d+1)<<endl; // diff of 4 bytes

    cout<<(unsigned long long)(&num_1d)<<endl;
    cout<<(unsigned long long)(&num_1d+1)<<endl<<endl; // diff of 20 bytes bcz it's
a pointer to pointer and adding 1 will inc by size * type_size of array: 5*4 = 20

    cout<<"2d int array"<<endl;
    int num_2d[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    cout<<num_2d<<endl; //address of first element (pointer to first row but stores
address of first element)
    cout<<&num_2d<<endl; //address of first element (pointer to pointer to first
row but stores address of first element)
    cout<<*num_2d<<endl; //address of first element (pointer to first element)
    cout<<**num_2d<<endl; //1

    cout<<(unsigned long long)(num_2d)<<endl;
    cout<<(unsigned long long)(num_2d + 1)<<endl; //diff of 12 bytes because we are
adding 1 to a pointer to the first row and that row has 3 elements of in type

    cout<<(unsigned long long)(&num_2d)<<endl;
    cout<<(unsigned long long)(&num_2d + 1)<<endl; //diff of 36 bytes because we
are adding 1 to a pointer to the whole 2d array and that array has 9 elements of in
type

    cout<<(unsigned long long)(*num_2d + 1)<<endl; //first dereferencing and then
adding traversed that specific row meaning this will return the address of second
element of first row
    cout<<*(*num_2d + 1)<<endl; //dreferencing the above gives 2
    //adding a number without dereferencing the array first traverses among rows
and dereferencing first traverses among columns of that specific row

    cout<<((num_2d + 1) + 2); // moves 1 row over and then moves 2 columns over so
it should output 6
}
```

Program Breakdown:

1. 1D Array

cpp

Copy code

```
int num_1d[] = {1, 2, 3, 4, 5};
```

- This declares a 1D array num_1d of integers, containing the values {1, 2, 3, 4, 5}.

Pointer Behavior in 1D Array:

- num_1d is a pointer to the first element of the array. In C++, when you use the name of an array, it decays to a pointer to its first element. This is why num_1d holds the address of the first element of the array, which is the address of num_1d[0].

cpp

Copy code

```
cout<<num_1d<<endl; // Prints the address of the first element

cout<<&num_1d<<endl; // Prints the address of the entire array (pointer to the array)

cout<<*num_1d<<endl; // Dereferencing the pointer gives the value of the first element: 1
```

- num_1d is a pointer to the first element, so it prints the address of num_1d[0].
- &num_1d prints the address of the entire array, which is a pointer to the array.
- Dereferencing num_1d (*num_1d) gives the value stored at the first element, which is 1.

2. Pointer Arithmetic for 1D Array

cpp

Copy code

```
cout<<(unsigned long long)(num_1d)<<endl;

cout<<(unsigned long long)(num_1d+1)<<endl; // diff of 4 bytes
```

- num_1d is a pointer to the first element.
- num_1d+1 gives the address of the second element. Since each element is of type int (which typically takes 4 bytes), adding 1 to the pointer will advance by 4 bytes.

cpp

Copy code

```
cout<<(unsigned long long)(&num_1d)<<endl;

cout<<(unsigned long long)(&num_1d+1)<<endl; // diff of 20 bytes
```

- &num_1d gives the address of the entire array.

- &num_1d+1 moves the pointer by 20 bytes, because the size of the array is 5 * sizeof(int) (which is 5 * 4 = 20 bytes).

3. 2D Array

cpp

Copy code

```
int num_2d[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

- This declares a 2D array num_2d with 3 rows and 3 columns, initialized with values.

Pointer Behavior in 2D Array:

- num_2d is a pointer to the first row of the array, i.e., the pointer to num_2d[0].

cpp

Copy code

```
cout<<num_2d<<endl; // Address of first row (pointer to the first row)

cout<<&num_2d<<endl; // Address of first row (pointer to pointer to the first row)

cout<<*num_2d<<endl; // Address of the first element in the first row

cout<<**num_2d<<endl; // Dereferencing twice gives the value of the first element: 1
```

- num_2d is a pointer to the first row.
- &num_2d is a pointer to the entire 2D array.
- Dereferencing num_2d gives the address of the first row.
- Dereferencing *num_2d gives the address of num_2d[0], which is the first element of the first row.
- Dereferencing **num_2d gives the value at num_2d[0][0], which is 1.

4. Pointer Arithmetic for 2D Array

cpp

Copy code

```
cout<<(unsigned long long)(num_2d)<<endl;

cout<<(unsigned long long)(num_2d + 1)<<endl; // diff of 12 bytes
```

- num_2d is a pointer to the first row of the 2D array.
- num_2d+1 moves the pointer by the size of one row, i.e., 3 * sizeof(int) (which is 12 bytes, because each row has 3 elements of type int).

cpp

Copy code

```
cout<<(unsigned long long)(&num_2d)<<endl;

cout<<(unsigned long long)(&num_2d + 1)<<endl; // diff of 36 bytes
```

- &num_2d gives the address of the entire 2D array.
- &num_2d + 1 moves the pointer by 36 bytes, as the entire 2D array has 3 * 3 * sizeof(int) (which is 36 bytes, because the array has 9 elements).

5. Accessing Specific Elements in 2D Array via Pointer Arithmetic

cpp

Copy code

```
cout<<(unsigned long long)(*num_2d + 1)<<endl; // address of second element of first row

cout<<*(*num_2d + 1)<<endl; // dereferencing the above gives 2
```

- *num_2d + 1 gives the address of the second element of the first row (num_2d[0][1]).
- Dereferencing *(*num_2d + 1) gives the value 2.

cpp

Copy code

```
cout<<((num_2d + 1) + 2); // Moves 1 row over and then moves 2 columns over to output 6
```

- num_2d + 1 moves the pointer to the second row, and then adding 2 moves it 2 columns over, landing on num_2d[1][2], which is 6.

Key Concepts:

1. **Arrays and Pointers:**
   - In C++, arrays are closely related to pointers. The name of the array, such as num_1d or num_2d, is treated as a pointer to the first element (or row in the case of a 2D array).
2. **Pointer Arithmetic:**
   - When you add an integer to a pointer, the pointer advances by the number of bytes corresponding to the type it points to. For example, adding 1 to a pointer to an int advances it by sizeof(int) (typically 4 bytes).
3. **Array of Pointers:**
   - In a multi-dimensional array, each element (row) is a pointer to the first element in that row, and pointer arithmetic can be used to navigate between rows and columns.
4. **Pointer Dereferencing:**
   - Dereferencing a pointer allows you to access the value stored at the address it points to. For example, *num_2d gives the address of the first element in the first row.