```
address of n1 in main(): 0x7ffea2e1b8d4
address of n1 in square1(): 0x7ffea2e1b8cc
Square of n1: 64
No change in n1: 8

address of n2 in main(): 0x7ffea2e1b8d0
address of n2 in square2(): 0x7ffea2e1b8d0
Square of n2: 64
Change reflected in n2: 64

address of n3 in main(): 0x7ffea2e1b8cc
address of n3 in square3(): 0x7ffea2e1b8cc
Square of n3: 64
Change reflected in n3: 64
```

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

// Pass-by-Value
int square1(int n) {
    cout << "address of n1 in square1(): " << &n << "\n";
    n *= n;
    return n;
}

// Pass-by-Reference with Pointer Arguments
void square2(int* n) {
    cout << "address of n2 in square2(): " << n << "\n";
    *n *= *n;
}

// Pass-by-Reference with Reference Arguments
void square3(int& n) {
    cout << "address of n3 in square3(): " << &n << "\n";
    n *= n;
}

void example() {
    // Call-by-Value
    int n1 = 8;
    cout << "address of n1 in main(): " << &n1 << "\n";
    cout << "Square of n1: " << square1(n1) << "\n";
    cout << "No change in n1: " << n1 << "\n";

    // Call-by-Reference with Pointer Arguments
    int n2 = 8;
    cout << "address of n2 in main(): " << &n2 << "\n";
    square2(&n2);
    cout << "Square of n2: " << n2 << "\n";
    cout << "Change reflected in n2: " << n2 << "\n";

    // Call-by-Reference with Reference Arguments
    int n3 = 8;
    cout << "address of n3 in main(): " << &n3 << "\n";
    square3(n3);
    cout << "Square of n3: " << n3 << "\n";
    cout << "Change reflected in n3: " << n3 << "\n";
}

// Driver program
int main() {
    example();
    return 0;
}
```

```
#include <iostream>

using namespace std;

class Base                    ANS + : {
    int a;
    int d;
    char ch;
};
int main()
{
  Base b;
  std::cout << "Size of class base is :
"<<sizeof(b) << std::endl;
    return 0;
}
```

Memory alignment ensures that data is stored in memory at addresses that are multiples of the data's size. This can improve the efficiency of data access.`int a` (4 bytes) starts at address 0.

- `int d` (4 bytes) starts at address 4.
- `char ch` (1 byte) starts at address 8.
- 3 bytes of padding are added after `char ch` to make the total size a multiple of 4 bytes (the size of the largest member).

Total Size:

- 4 bytes (for `int a`) + 4 bytes (for `int d`) + 1 byte (for `char ch`) + 3 bytes (padding) = 12 bytes.