1)

A.

```
int m = 5, n = 7; // m =5, n = 7
int *p; // m = 5, n = 7
p = &n; // *p = n = 7, m = 5
*p = 2 * n + m; // *p = 2 * 7 + 5 = 19 = n, m = 5
*p = 2 * n + m; // *p = 2 * 19 + 5 = 43 = n, m = 5
p = &m; // *p = m = 5, n = 43
*p = 2 * m + n; // *p = 2 * 5 + 43 = 53 = m, n = 43
*p = 2 * m + n; // *p = 2 * 53 + 43 = 149 = m, n = 43
cout << "m = " << m << ", "; // m = 149
cout << "n = " << n << ", "; // n = 43
cout << "*p = " << *p << endl; // *p = 149
```

B.

```
int x; // 3173315904
int *ptr = &x; // &ptr = &x = 3173315904
ptr++; // &ptr = 3173315905
```

C.

The graph is not properly initialized

D.

```
int arr[5] = {1,2,3,4,5} ;
int *ptr = arr ;
cout << *(ptr + 2) << endl ;
```

Output:

```
3
5
```

2)

A.

```
leo.Eats();
```

B.

```
lionPtr->Sleep();
```

C.

It's a variable in a class

D.

Leo eats meat.

3)

A.

waBegin and waEnd return iterators pointing to the first and last elements of the array

B.

```
WideArray::waIterator::operator++() {
        m_ptr+=2;
}
```

C.

```
WideArray::waIterator waEnd() {
    return waIterator(m_data+2*m_size)
}
```

4)

A.

Segmentation fault because there isn't a declared variable at this address

B.

The old data array is not deallocated. I would use valgrind to detect it.