Some notes on pointers in C++.

(i) int *p = new int(4)

new; returns pointer to memory address,

⇒ p stores address of contiguous block of memory that will store 4 integers.

The values at these 4 locations can be assigned/obtained in 2 ways:

(i) p[0], p[1], p[2], p[3]

(ii) *(p+0) *(p+1) *(p+2) *(p+3)

(ii) vector <int> *p = new vector<int>{1, 2, 3, 4};

p; a pointer will point to address/contain address of a variable of type vector<int>.

The values can be assigned/obtained in the following way:

(*p)[0], (*p)[1], (*p)[2], (*p)[3].

(*p) is vector, then traverse vector as [0], – [3].

Instead of above, we cannot do

*((*p) + i) ie,

When we had int *p = new int(4):

p[i] = *(p + i)

<u>But now</u>: (*p)[i] ≠ *((*p) + i)

B/c p in second case

vector <int> *p = new vector<int>{1,2,3,4}

is pointer to a vector, not its values.

So $\underset{\text{vector address}}{((*p)} + i)$, <u>not make sense</u>

(iii) Usage of vector<int*> p. :

ie, p an array (vector) that contains values that point to an integer

<u>eg:</u> int *a = new int(4)

Let a[0] or *(a+0) = 11

a[1] or *(a+1) = 12

a[2] or *(a+2) = 13

a[3] or *(a+3) = 14.

Now, declare:

vector<int*> p = {a, a+1, a+2, a+3}

p: now a vector. <u>NOT</u> a pointer to a vector.

∴ To output 11, 12, 13, 14:

(i=0; i<4; i++)
    cout << *(p[i]) << " ";

<u>Cannot</u> assume that p is a pointer to a vector and do stuff like:

(i=0; i<4; i++)

*((*p)[i])

i.e. ------

{a, a+1, ..a+3}

and then

<u>WRONG</u>.