New and Delete in C++

Dynamic memory allocation in C/C++ refers to performing memory allocation manually by a programmer.

One use of dynamically allocated memory is to allocate memory of variable size, which is not possible with compiler allocated memory

For dynamically allocated memory like "int *p = new int[10]", it is the programmer's responsibility to deallocate memory when no longer needed. If the programmer doesn't deallocate memory, it causes a memory leak

New and Delete in C++

new operator

The new operator denotes a request for memory allocation on the Free Store. If sufficient memory is available, a new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

```
pointer-variable = new data-type;
```

```
// Pointer initialized with NULL
// Then request memory for the variable

int *p = NULL;
p = new int;

OR

// Combine declaration of pointer
// and their assignment

int *p = new int;
```

New and Delete in C++

new operator

For custom data types, a constructor is required (with the data type as input) for initializing the value.

```
pointer-variable = new data-type(value);
```

```
int* p = new int(25);
float* q = new float(75.25);
```

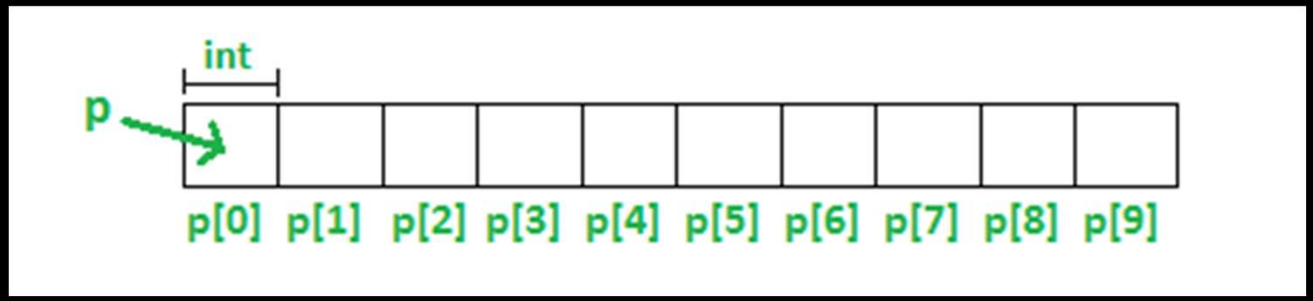
New and Delete in C++

new operator

Allocate a block of memory: a new operator is also used to allocate a block(an array) of memory of type *data type*.

```
pointer-variable = new data-type[size];
```

```
int* p = new int[10];
```



New and Delete in C++

new operator

If enough memory is not available.

```
int *p = new int;

if (!p)
{
   cout << "Memory allocation failed\n";
}
```

New and Delete in C++

delete operator

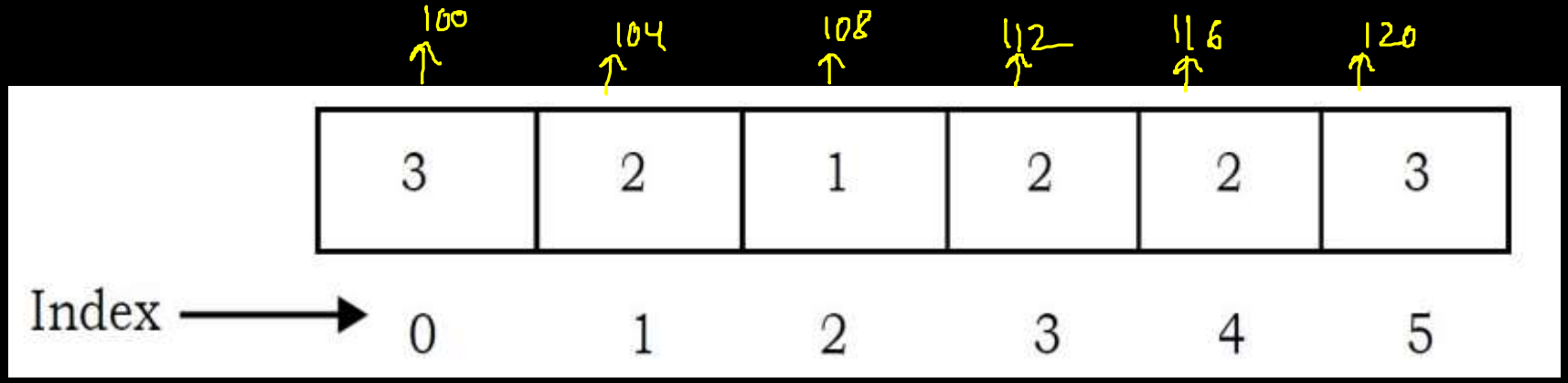
It is the programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator in C++ language.

```
delete pointer_variable;\
```

```
delete  p;
delete [ ] p;
```

Array

One memory block is allocated for the entire array to hold the elements of the array. The array elements can be accessed in constant time by using the index of the element as the subscript.



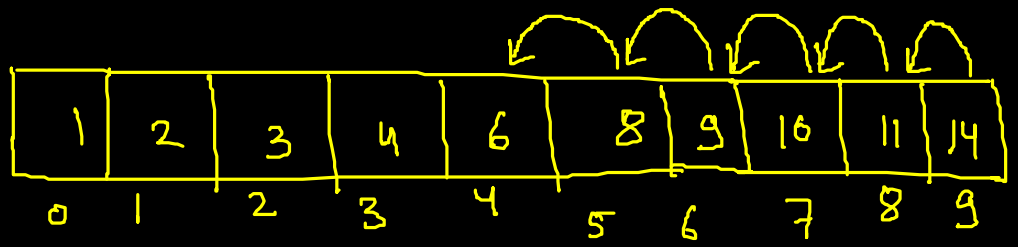
int a[10];

a[5] = 100 + 5 x 4 = 120

O(1) ← [Address = (Base Address) + (Index) x (Size of each block)]

Search , Insert and Delete in an Array

1) `int a[10];`



insertion :-

```
for(i=0; i<10;i++)
{  cin >> a[i];
}
```

→ O(1) ⇒ Time complexity

```
a[5] = 10;
```

Search :-

```
for ( i=0 ; i<10 ;i++)
{
    if ( k == a[i])
        cout<<"Element found";
}
```

→ O(n)

3) deletion :- O(n)

```
for(i=index ; i<9;i++)
{
   a[i] = a[i+1];
}
  a[last] =0;
```

Advantages and Disadvantages of Array

1) Constant time → memory access

2) Disadvantage:- fixed size.

Dynamic Array

- ✓ Dynamic array (also called as growable array, resizable array, dynamic table, or array list) is a random access, variable-size list data structure that allows elements to be added or removed.

- ✓ One simple way of implementing dynamic arrays is to initially start with some fixed size array. As soon as that array becomes full, create the new array double the size of the original array. Similarly, reduce the array size to half if the elements in the array are less than half.

Dynamic Array

Factors impacting performance of Dynamic Arrays-

The array's initial size and its growth factor determine its performance.

1. If an array has a small size and a small growth factor, it will keep on reallocating memory more often. This will reduce the performance of the array.

2. If an array has a large size and a large growth factor, it will have a huge chunk of unused memory. Due to this, resize operations may take longer. This will reduce the performance of the array.

Dynamic Array

Array Initialization –

```
int *array { new int[5] { 10, 7, 15, 3, 11 } };
```

```
int *a { new int[10]{ } };  // Initialized with zero
```

int a[R][C]; a[i][j];

Address:

= Base + [(size of each block) * C] * i + (size of each block) * j