To store an address of a memory location we use pointer variables. The declaration of such a variable and how it can be used is given below:

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Assigning an address</th>
<th>Accessing x through p</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Store an Integer</td>
<td>int x;</td>
<td>p = &amp;x;</td>
</tr>
<tr>
<td>To Store an Address</td>
<td>int *p;</td>
<td></td>
</tr>
</tbody>
</table>

- Data type is int
- Name of the variable is x
- Data type is int *
- Name of the variable is p
- Address of x is stored in p.
- Creates another name for x.
- The alias for x is *p
- Accessing x using *p.
- 4 is stored in x

To understand the concept of a pointer variable p, think of it as a box where you have the address of a house. So, it is just a box, big enough to hold an address. The variable x is the house itself. To place the address of x in p, you write p=&x. To put somebody (say a person called 9) in the house x, you can write x=9. Remember a person cannot live in p but can live in x. You cannot put the person 9 in the box p; it can hold only an address of a house. You can go to x after picking up the address in p; that is called de-referencing and gives another name (a nick-name, an alias) for the house x. It is *p. So, to put the person 9 in the house, you can also write *p=9. A pictorial representation is given here to illustrate the relationship between x and p.

Since p can hold an address of a house, you can change the address you are placing in it. If you want to store the address of another house y, you write p=&y. Now, if you pick up the address in p, you will go to y and not to x. So, *p becomes another name for the house y.

**Example:**

<table>
<thead>
<tr>
<th>C++ Statements</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 int x, y, z; int *p, *q;</td>
<td>x y z p q</td>
</tr>
</tbody>
</table>
Addresses of Array Locations

- **Given the declaration:** `int x[10];`

- **A Pictorial Representation of the Array:**

  ![Diagram of array locations]

  Since `x` has the address of the first location `x[0]` of the array, it is a pointer variable and can be used to access the first location using de-referencing. So, another name of the first location is `*x`. We can use that address to determine the addresses of other array locations.

  Since array locations are consecutive memory locations starting with `x[0]`
  - `(x+1)` is the address of the location `x[1]`,
  - `(x+2)` is the address of the location `x[2]` and so on.
  In general,
  - `(x+k)` is the address of the location `x[k].`

  Using de-referencing
  - `*(x+1)` is another name for `x[1]`,
  - `*(x+2)` is another name for `x[2]`, and so on.
  In general,
  - `*(x+k)` is another name for `x[k].`

  We can use a pointer variable `int *p`, to store the address of an array location. When we do that we create two more sets of names for the array locations.

  Suppose `p = &x[3];`
  Since `p` has the address of `x[3]`, `*p` is another name for `x[3]`. Hence
  - `*(p+1)` is another name for the next location `x[4].`
  - `*(p+2)` is another name for the location `x[5]` and so on.

  Also, since `p` has an address of an array location, `p[0]` is the name of that location. That is, `p[0]` is another name for `x[3]`. Hence,
  - `p[1]` is another name for `x[4]`,
  - `p[2]` is another name for `x[5]` and so on.

  So, using `x` and `p`, the array location `x[4]` has 4 names:
  
  (1) `x[4]`  (2) `*(x+4)`  (3) `*(p+1)`  (4) `p[1]`

**Note:** Even though `x` is a pointer variable it is a constant. That means, you can use its value but you cannot change its value.
Dynamical Allocations

In the declaration: \( \text{int } x; \) the variable \( x \) is a static variable. Memory allocation for \( x \) is done during compilation of a program. We can create dynamic variables using a pointer variable. Memory allocation for a dynamic variable is done during an execution of a program. Statements given below show how to create and use such dynamic variables.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Explanation</th>
<th>Pictorial Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *p;</td>
<td>( p ) is a pointer variable, pointer to an integer. It is a static variable.</td>
<td><img src="image" alt="Diagram of p" /></td>
</tr>
<tr>
<td>( p = \text{new int}; )</td>
<td>( \text{new} ) creates a dynamic variable and returns its address; ( p ) has that address.</td>
<td><img src="image" alt="Diagram of new int" /></td>
</tr>
<tr>
<td>( *p = 4; )</td>
<td>( *p ) is the name of the dynamic variable and 4 is stored in that variable.</td>
<td><img src="image" alt="Diagram of *p = 4" /></td>
</tr>
<tr>
<td>char *p;</td>
<td>( p ) is a pointer variable, pointer to a character. It is a static variable.</td>
<td><img src="image" alt="Diagram of char *p" /></td>
</tr>
<tr>
<td>( p = \text{new char}; )</td>
<td>( \text{new} ) creates a dynamic variable and returns its address; ( p ) has that address.</td>
<td><img src="image" alt="Diagram of new char" /></td>
</tr>
<tr>
<td>( *p = 't'; )</td>
<td>( *p ) is the name of the dynamic variable and ‘t’ is stored in that variable.</td>
<td><img src="image" alt="Diagram of *p = 't'" /></td>
</tr>
</tbody>
</table>

Remember: The name of a dynamic variable is \( *(\text{whoever has its address}) \)

Example: Define necessary data types and write statements to create static and dynamic variables indicated in the diagram and complete the assignments.

In the diagram, there is only one static variable \( x \). All other variables are dynamic variables. The three colored boxes are unknown data types. So, we need to define them using struct. Let us name the data types as GreenType, BlueType and PinkType. The definitions are as follows:

```c
struct BlueType {
    char Ba;
    float *Bb;
};
struct PinkType {
    float Pa;
    int Pb;
};
struct GreenType {
    BlueType *Ga;
    int Gb;
    PinkType *Gc;
};
```

GreenType *x; // Declaration of the static variable \( x \); data type of \( x \) is “pointer to GreenType”
x = new GreenType; // \( x \) is pointing to a dynamic variable of GreenType. Its name is \( *(x) \)
\((*x).Ga\) = new BlueType; // \( Ga \) in \( *(x) \) is pointing to a dynamic variable of BlueType.
\((*x).Gb\) = 45; // \( Gb \) in \( *(x) \) has the value 45.
\((*x).Gc\) = new PinkType; // \( Gc \) in \( *(x) \) is pointing to a dynamic variable of PinkType.

\(/ / \text{The name of the dynamic variable of BlueType is } \ast \left( \ast (x) . Ga \right)\)  \(// \text{The name of the dynamic variable of PinkType is } \ast \left( \ast (x) . Gc \right)\)

\(\left( \ast \left( \ast (x) . Ga \right) \right) . Ba = 't' ;\)
\(\left( \ast \left( \ast (x) . Ga \right) \right) . Bb = \text{new float;}\)
\(* \left( \ast \left( \ast (x) . Ga \right) \right) . Bb \) = 3.4;
\(\left( \ast \left( \ast (x) . Gc \right) \right) . Pa = 65.5;\)
\(\left( \ast \left( \ast (x) . Gc \right) \right) . Pb = 100;\)