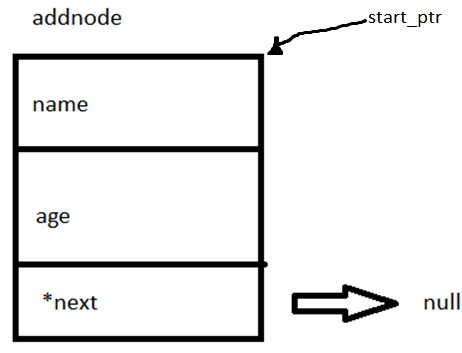
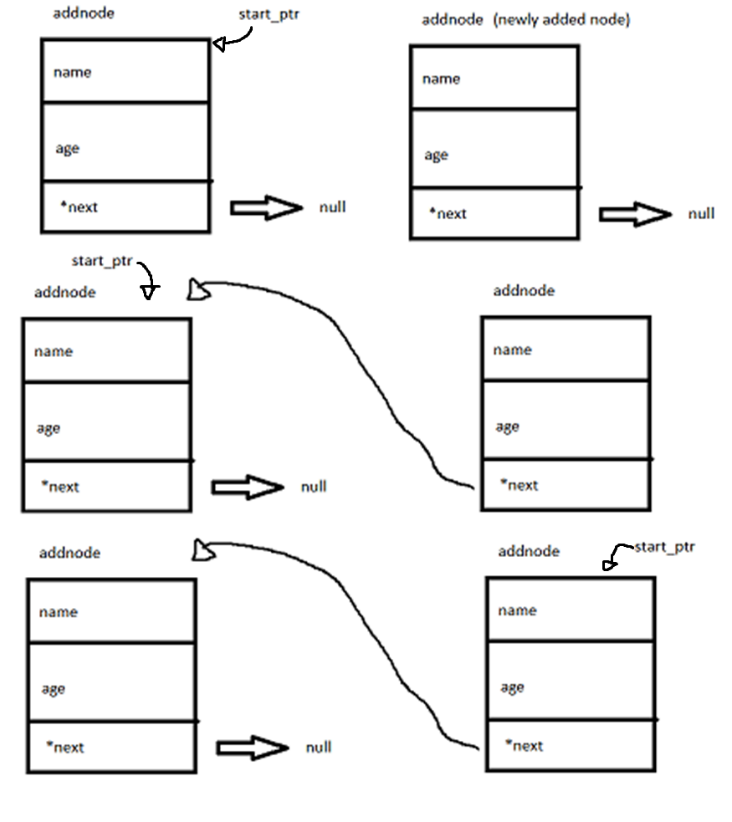
1. Compile and run exercise1.cpp and answer the questions that follow.
   1. What does addNode1() do? (Answer must be in detail; obviously as the name suggests, it add nodes; how does it function?). Use illustration to help you explain

- first it declares addnode pointer then right after declaring that, it asks for a cin.getline then assigns it to addnode->name and also same as for the addnode->age. Then addnode->next is set to null.

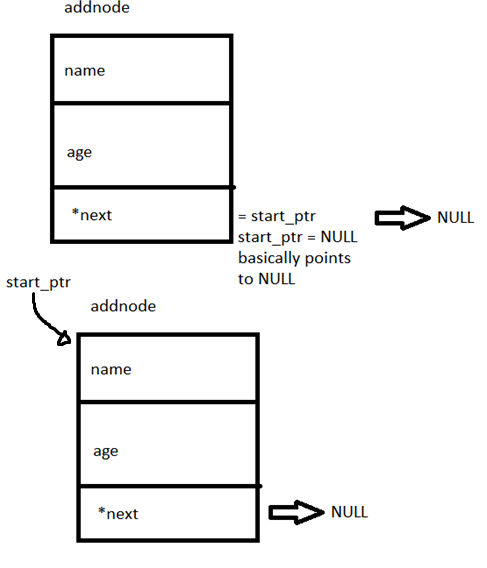
Right after that, the code checks if there is an existing node beforehand, if there are none the start\_ptr become the newly added node. Else, if there is one node existing. The previous node will be pointed by the newly added node. Then the newly added node will become the start\_ptr. Basically newer nodes becomes the start\_ptr and pushes back the older nodes. Newer nodes are added in the beginning of the list.

* 1. Add another addnode1() and displayNode() lines in the main function. This will allow you to create 3 nodes in the list. What happens when the condition in line50 is changed to *temp->next!=NULL*? Why did this happen? How did this change affect the program execution?

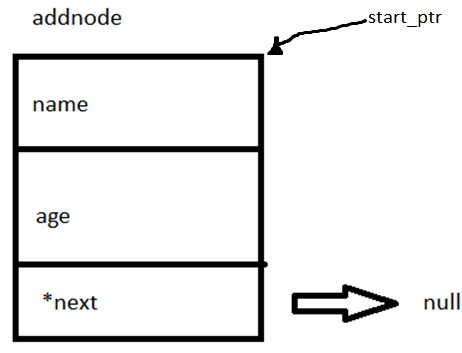
- The first set of input for the node is not being printed in the command line. Since the first node->next always points to NULL, it will be never be printed since the condition it will only print the current if temp->next is not NULL.

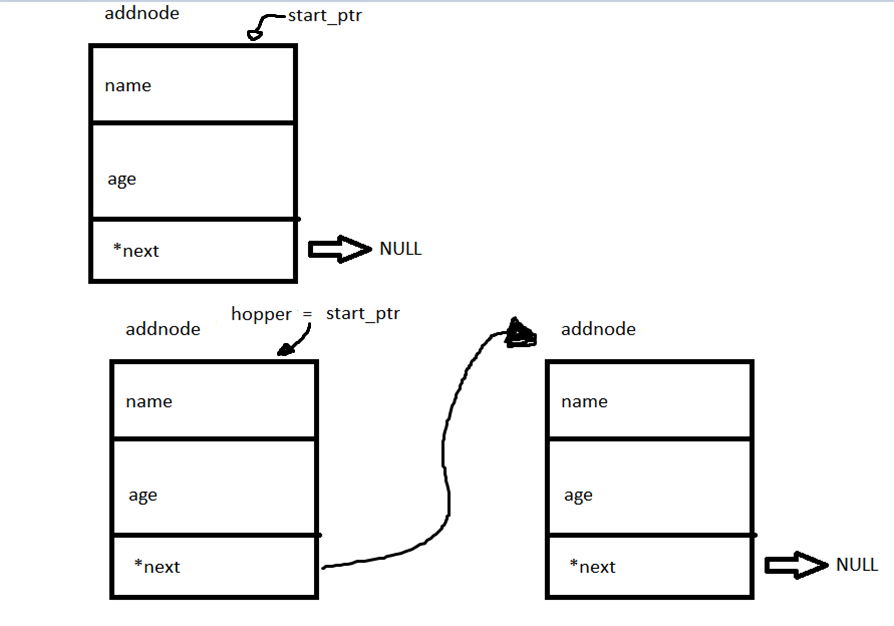
* 1. What does line36 check for? If this condition is removed what happens?

- Not much really happens but there is a difference. For the first node. This happens:

The addnode->next points to start\_ptr which is NULL. Then start\_ptr is assigned with addnode. If the steps are interchanged, it will create an infinite loop in the displayNode.

1. Compile and run exercise2.cpp and answer the questions that follow.
   1. What does addNode2() do? (Again, you are required to answer in detail.). Use illustration to help you explain.

- first it declares a node \*hopper equal to start\_ptr and addnode pointer. Then right after declaring that, it asks for a cin.getline then assigns it to addnode->name and also same as for the addnode->age. Then start\_ptr is equal to current addnode if the start\_ptr is NULL.

As soon as we get to the second inputs for the node. Hopper->next will point to the newly added node. Which is \*hopper is the start\_ptr.

* 1. How different are the codes in addNode2() from addNode1() of exercise1.cpp? What was/were changed?

- very similar to the first one but instead of newer nodes becomes the start\_ptr. The newer nodes are added in the last. The exercise1 pushes older nodes into the linked list while exercise2 adds new nodes at the end of the list.

* 1. What is the pointer *hopper* used for?

- hopper is used to traverse through the linked list. It starts at start\_ptr. Then moves to the next node if conditions are met.

* 1. What happens if the condition in line 42 is changed to *hopper != NULL*? How did this change affect the program execution?

- it produces an infinite loop and displays. It produces an infinite loop since hopper will never be null since it is pointing to start\_ptr.

* 1. Discuss what each line does:
     1. Line 42 : *while(hopper->next!= NULL)*- checks if the current position of node that the program is into is NULL or not. If the hopper->next is NULL, the loop ends and signifies that the current node is actually the last node.
     2. Line 44 : *hopper=hopper->next;*- hops or traverses to next node linked. it is very similar to i++ of arrays.
     3. Line 47 : *hopper->next=addnode;*

- This line will be executed when the last node is reached. The addnode is linked to the list.