FINAL EXAM PROFESSIONAL ELECTIVE 2
VISUALIZING LINKED LISTS

NAME: TAPNIO, XYMON D.
SECTION: UCOS 4-1

INSTRUCTIONS

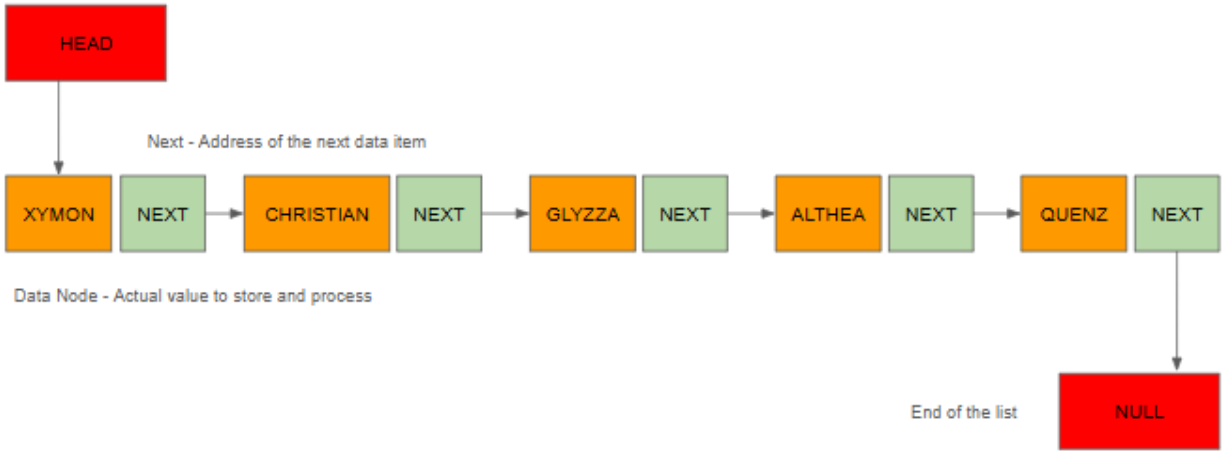
- The goal of this final exam project is to **VISUALIZE HOW LINKED LISTS WORK**. Write first a Linked List implementation using Python OOP with insertion, deletion, updating, and displaying methods. The linked list you'll show in the diagram should contain a minimum of **FIVE ELEMENTS**. You are tasked to **CREATE DIAGRAMS** showing the following:
    - **INSERTION** of a new element to linked list.
    - **DELETION** of a given element inside a linked list.
    - **UPDATING** of an element from the linked list.
    - **DISPLAYING** of all elements from the linked list.

- Please follow the format of this document and **REFER TO THE CRITERIA** on the last page of this document.

ELEMENTS TO ADD IN LINKED LIST: Xymon, Christian, Glyzza, Althea, Quenz

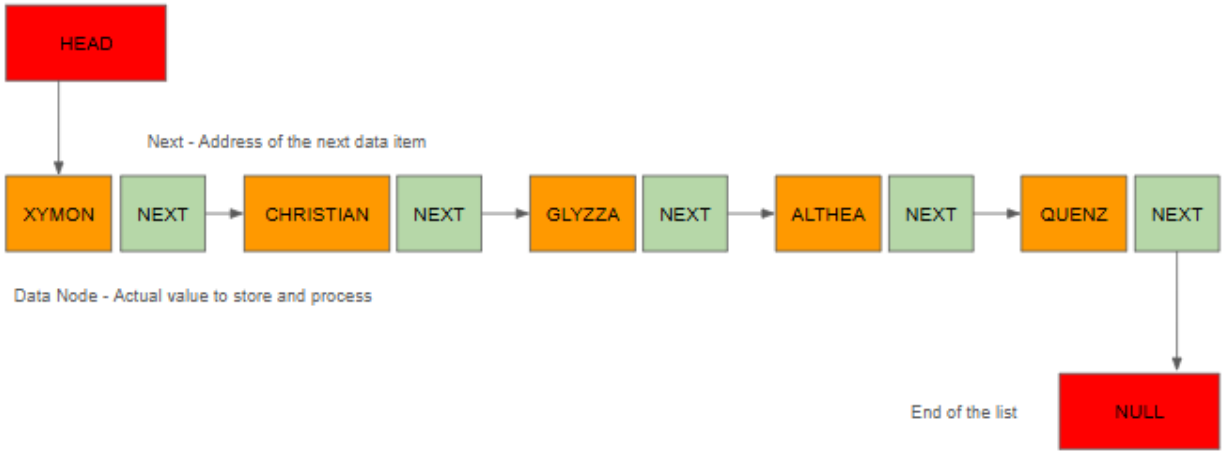
INSERTION OF ELEMENT:

Initial Linked List:
Head -> [Xymon] -> [Christian] -> [Glyzza] -> [Althea] -> [Quenz] -> None

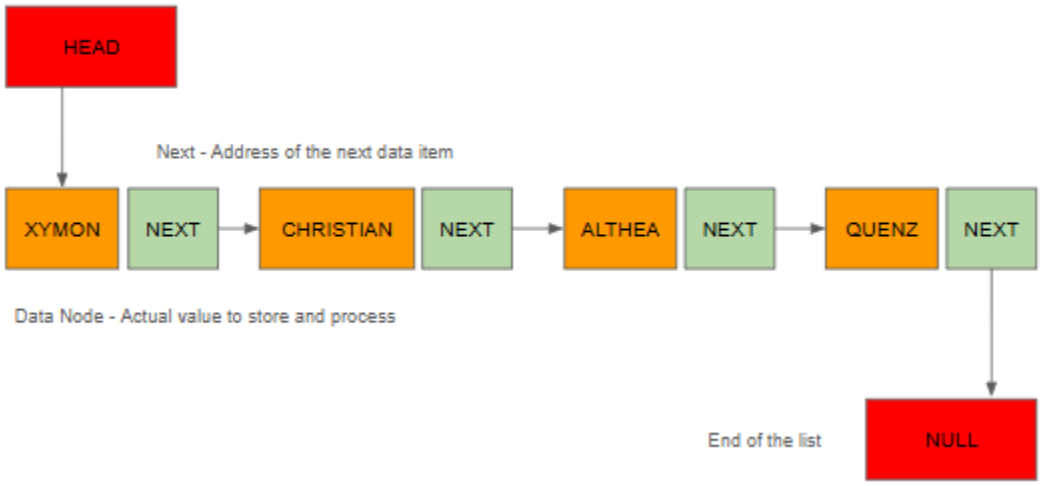


DELETION OF ELEMENT:

Before Deletion of 'Glyzza':
Head -> [Xymon] -> [Christian] -> [Glyzza] -> [Althea] -> [Quenz] -> None



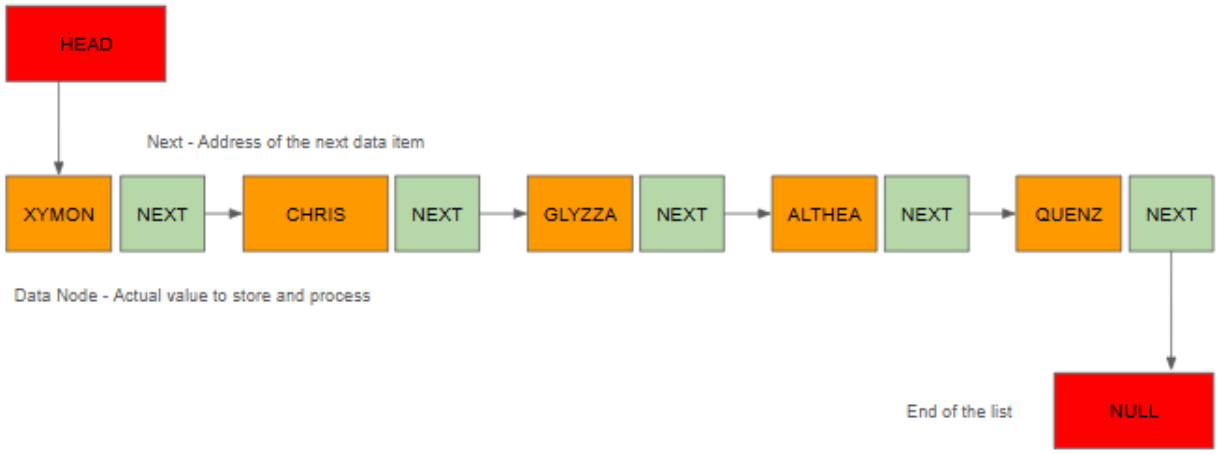
After Deletion of 'Glyzza':
Head -> [Xymon] -> [Christian] -> [Althea] -> [Quenz] -> None



UPDATING OF ELEMENT:

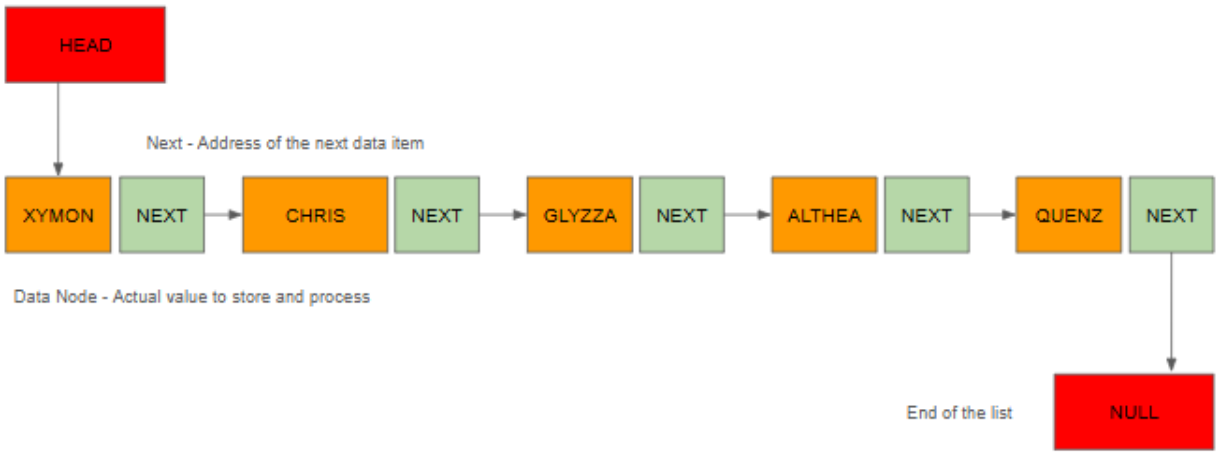
Before Updating 'Christian' to 'Chris':
Head -> [Xymon] -> [Christian] -> [Althea] -> [Quenz] -> None

After Updating 'Christian' to 'Chris':
Head -> [Xymon] -> [Chris] -> [Althea] -> [Quenz] -> None



DISPLAYING OF ELEMENTS:

Displaying Linked List:
[Xymon, Chris, Glyzza, Althea, Quenz]



LINKED LIST IMPLEMENTATION USING PYTHON OOP (INSERTION, DELETION, UPDATING, AND DISPLAYING)

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node

    def delete(self, key):
        temp = self.head
        if temp and temp.data == key:
```

```
            self.head = temp.next
            temp = None
            return
        prev = None
        while temp and temp.data != key:
            prev = temp
            temp = temp.next
        if temp is None:
            return
        prev.next = temp.next
        temp = None

    def update(self, old_data, new_data):
        current = self.head
        while current:
            if current.data == old_data:
                current.data = new_data
                return
            current = current.next

    def display(self):
        current = self.head
        elements = []
        while current:
            elements.append(current.data)
            current = current.next
        return elements

# Example usage
linked_list = LinkedList()
elements = ["Xymon", "Christian", "Glyzza", "Althea", "Quenz"]
for element in elements:
    linked_list.insert(element)

print("Initial Linked List:", linked_list.display())

# Deleting an element
linked_list.delete("Glyzza")
print("After Deletion of 'Glyzza':", linked_list.display())

# Updating an element
linked_list.update("Christian", "Chris")
print("After Updating 'Christian' to 'Chris':", linked_list.display())
below.
```

CRITERIA:

CATEGORY	**PERCENTAGE**
Technical concepts about Linked Lists are made easier because of the diagram. Diagram clearly represents what takes place in the operations stated.	40%
The diagram created is based on the linked list implementation source code that you provided.	30%
Labels were shown to communicate ideas.	20%
The diagram is done neatly. Color combination is considered carefully.	10%
TOTAL	**100%**