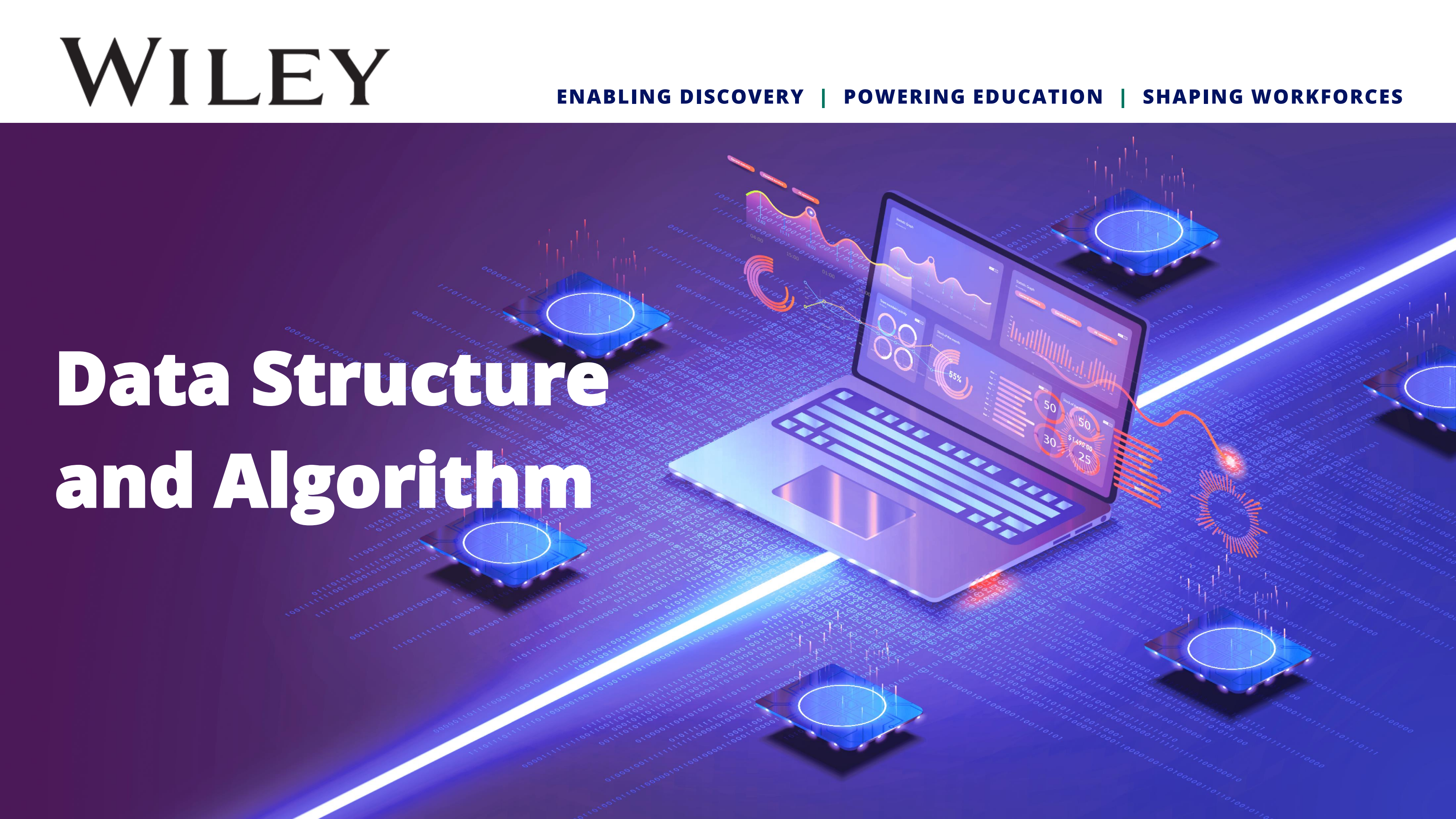


WILEY

ENABLING DISCOVERY | POWERING EDUCATION | SHAPING WORKFORCES

Data Structure and Algorithm

The background of the slide is a vibrant, futuristic digital landscape. It features a central laptop with a screen displaying various data visualizations, including line graphs, bar charts, and circular progress indicators. The laptop is surrounded by several glowing blue squares, each with a white circular outline and a grid pattern, resembling circuit boards or data nodes. These squares are connected by glowing orange lines and dots, suggesting a network or data flow. The entire scene is set against a dark blue background filled with binary code (0s and 1s) and a bright, diagonal light beam.

Objectives

By the end of this module, you will be able to:



Identify the features of linked lists



Implement a singly-linked list



Implement doubly-linked list

Let Us Discuss

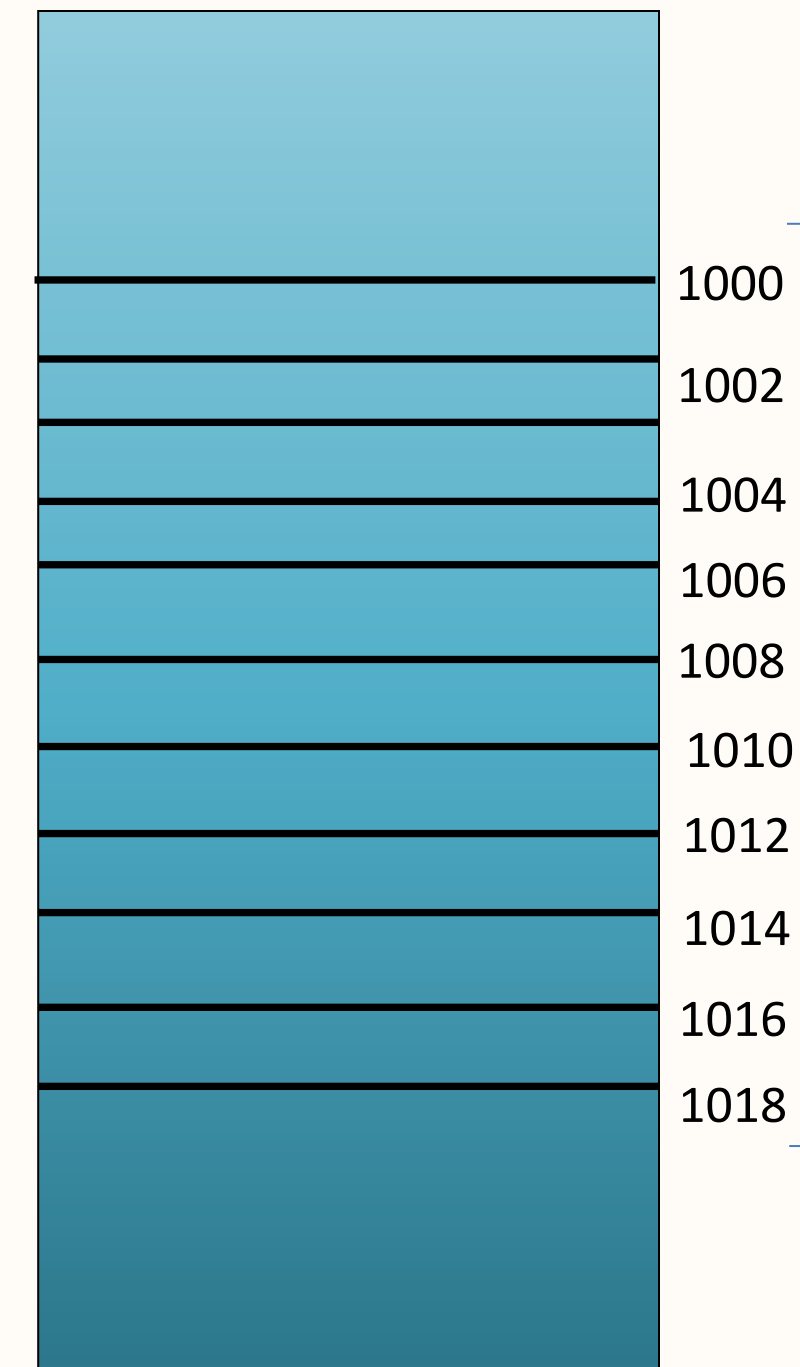
Write an algorithm to generate and store all prime numbers between 1 and 10,00,000 and display them.

How will you do it and which data structure will you use to store the prime numbers?

Dynamic Memory Allocation

- When you declare an array, a contiguous block of memory is allocated.
- For example, declare an array of size 10 to store the first 10 prime numbers, as shown in the following figure.

One contiguous block of memory allocated for the array to store 10 elements



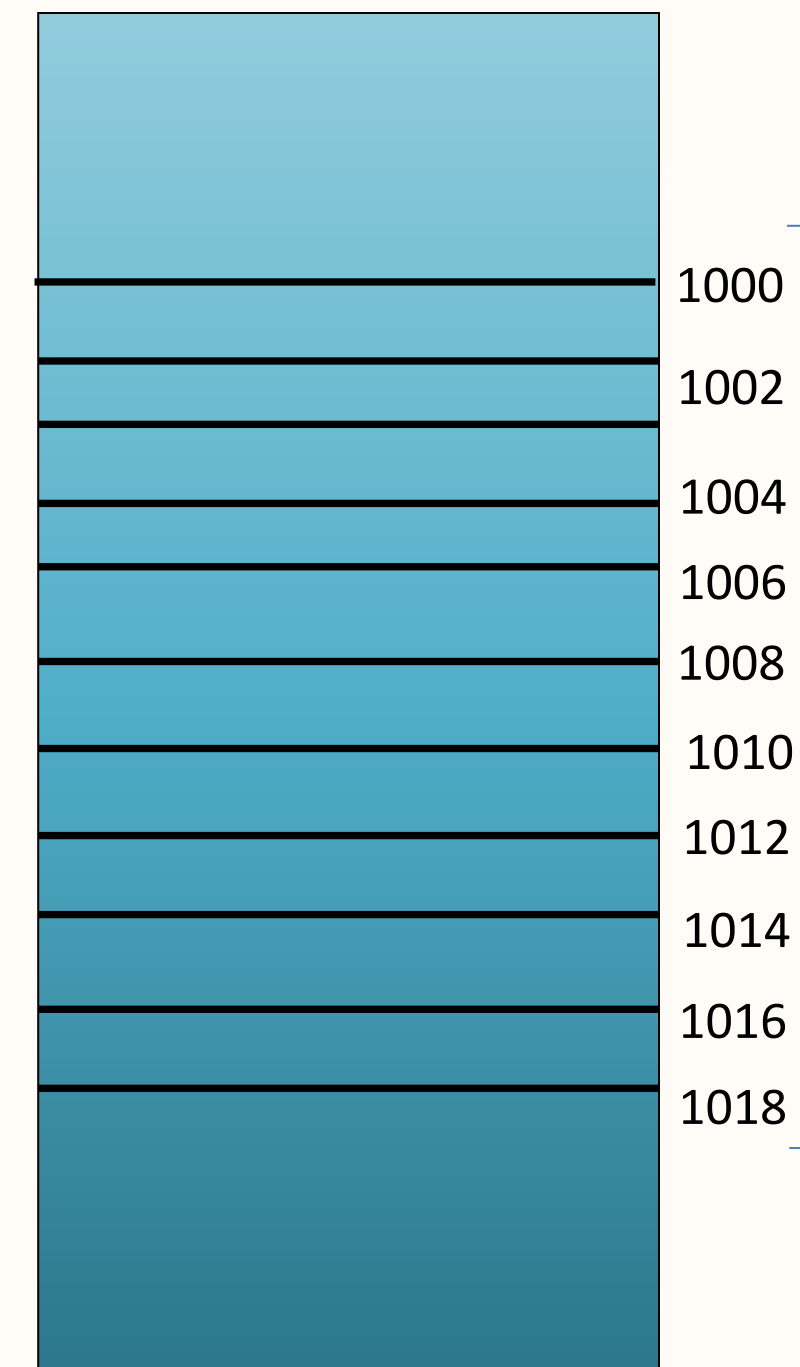
Memory representation

Dynamic Memory Allocation

If you know the address of the first element in the array, you can calculate the address of any other element as shown:

- Address of the first element + (size of the element × index of the element)

One contiguous block of memory allocated for the array to store 10 elements



Memory representation

Dynamic Memory Allocation (cont.)

- When memory is allocated dynamically, a block of memory is assigned arbitrarily from any location in the memory.
- Therefore, unlike arrays, these blocks may not be contiguous and may be spread randomly in the memory.

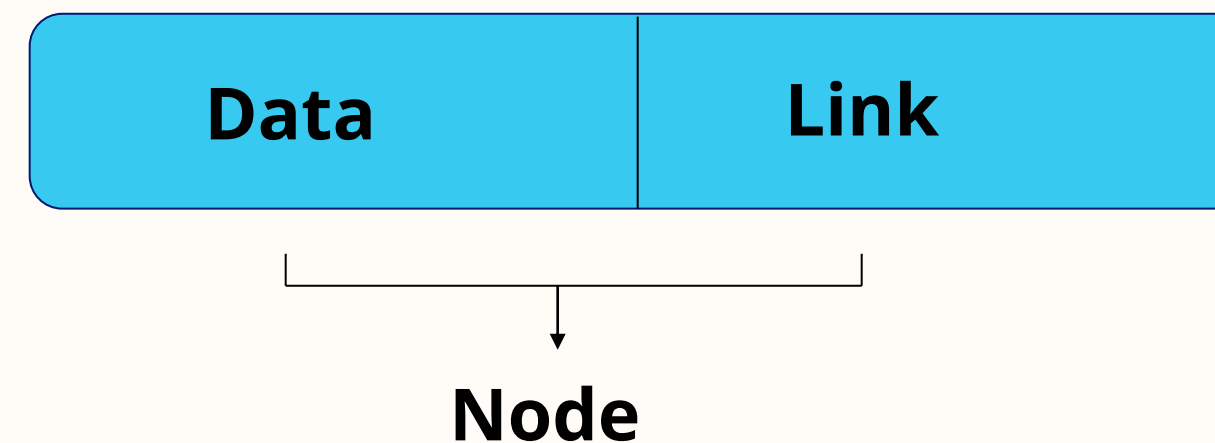
Defining Linked Lists

Linked List:

- Is a dynamic data structure.
- Allows memory to be allocated as and when it is required.
- Consists of a chain of elements, in which each element is referred to as a node.

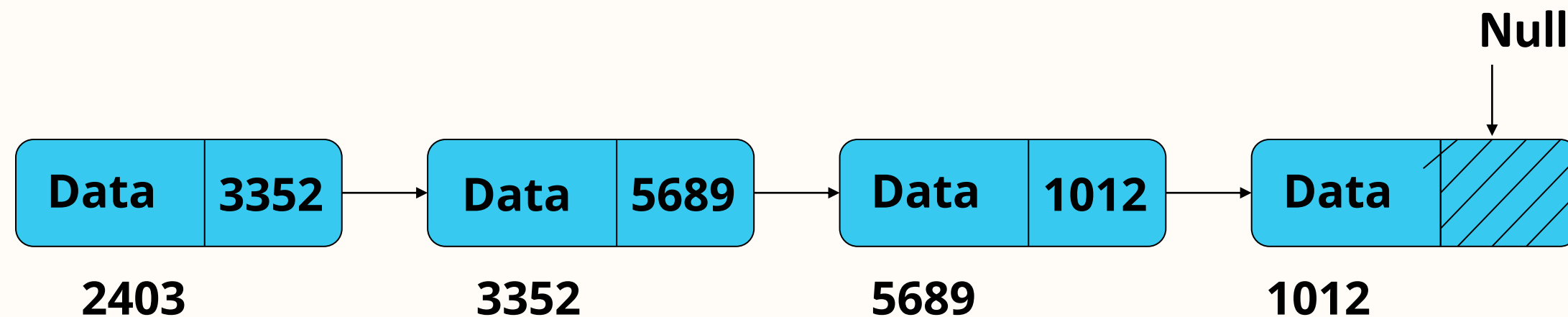
Defining Linked Lists

- A node is the basic building block of a linked list.
- A node consists of two parts:
 - Data: Refers to the information held by the node.
 - Link: Holds the address of the next node in the list.



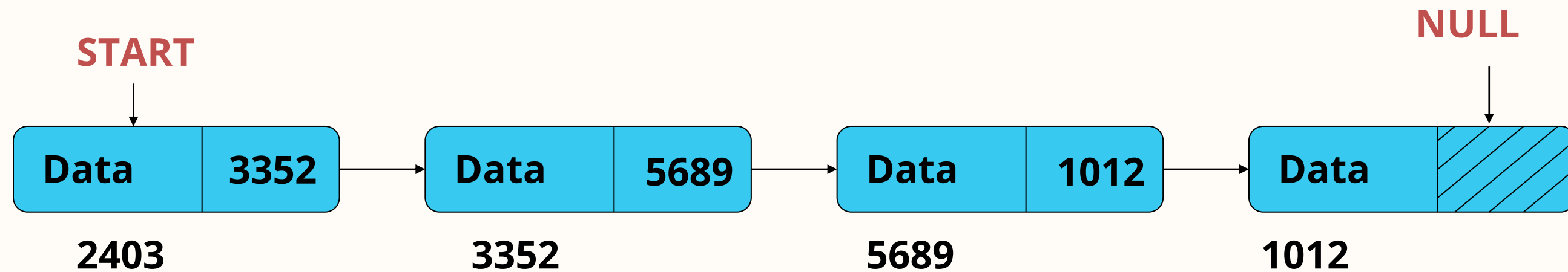
Defining Linked Lists (cont.)

- All the nodes in a linked list are present at arbitrary memory locations.
- Therefore, every node in a linked list has a link field that stores the address of the next node in sequence.
- The last node in a linked list does not point to any other node. Therefore, it points to NULL.



Defining Linked Lists (cont.)

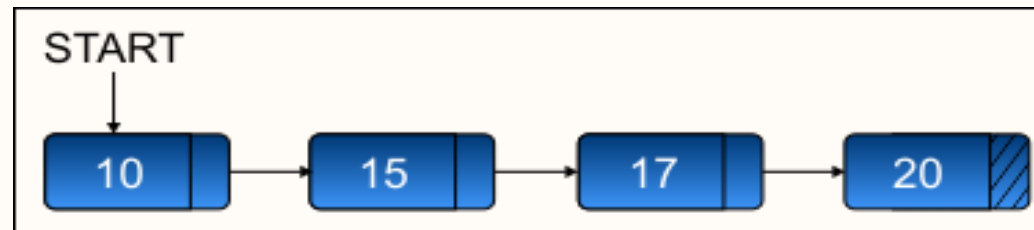
- To keep track of the first node, declare a variable/pointer, **START**, which always points to the first node.
- When the list is empty, **START** contains, **NULL**.



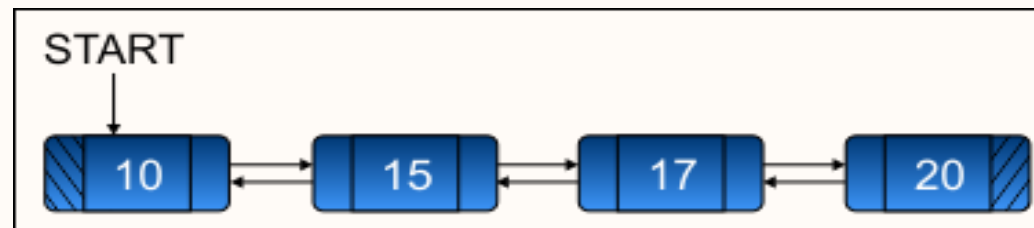
Identifying Different Types of Linked Lists

- To keep track of the first node, declare a variable/pointer, START, which always points to the first node.
- When the list is empty, START contains, NULL.
- Linked lists can be of the following types:

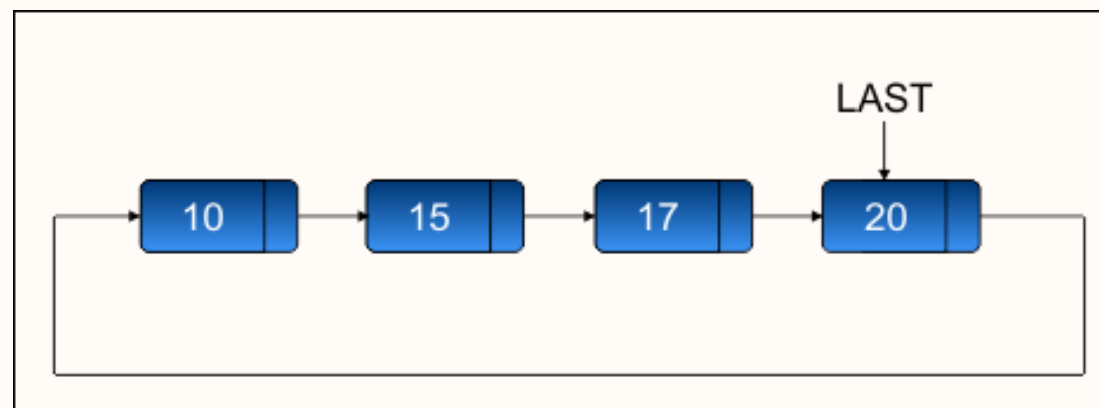
- Singly-linked list



- Doubly-linked list



- Circular-linked list



Question

Which one of the following linked lists has neither any beginning nor any end?

- A. Singly-linked list
- B. Doubly-linked list
- C. Circular-linked list



Singly Linked List

A singly-linked list is represented in a program by defining two classes:

- Node class: This class contains the data members which represent the data to be stored and the reference of the next node in the sequence.

```
class Node:
    def __init__(self,
dataval=None):
        self.dataval = dataval
        self.nextval = None

class SLinkedList:
    def __init__(self):
        self.headval = None

list1 = SLinkedList()
list1.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")
# Link first Node to second node
list1.headval.nextval = e2

# Link second Node to third node
e2.nextval = e3
```

Singly Linked List

Let us solve the given problem of storing prime numbers using a linked list:

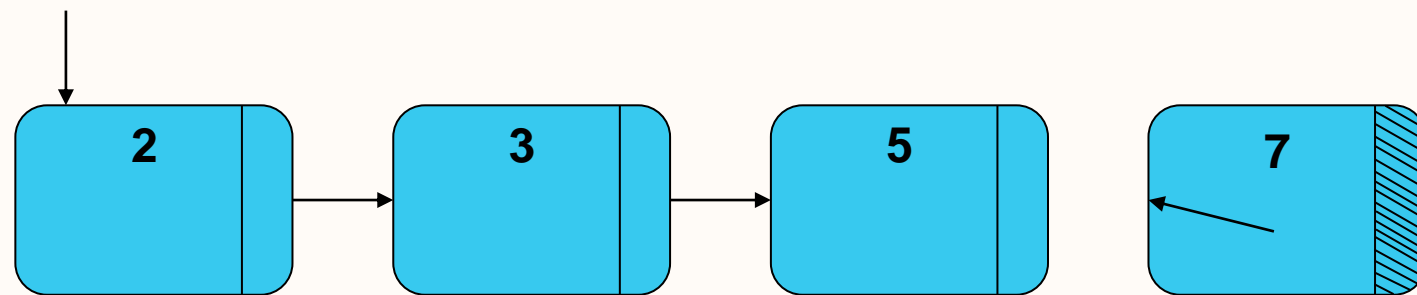
1. Repeat step 2 varying n from 2 to 1000000.
2. If n is a prime number, insert it in the linked list:
 - a) Allocate memory for a new node
 - b) Store the prime number in the new node
 - c) Attach the new node in the linked list
3. Display the prime numbers stored in the linked list.

Traversing a Singly-Linked List

Algorithm for traversing a singly-linked list:

1. Make currentNode point to the first node in the list.
2. Repeat the steps 3 and 4, until currentNode becomes NULL.
3. Display the information contained in the node marked as currentNode.
4. Make currentNode point to the next node in the sequence.

START



Inserting a Node in a Singly-Linked List

Insertion refers to the process of adding a new node in the list.

The following algorithm depicts the logic to insert a node in an empty linked list:

1. Allocate memory for the new node.
2. Assign a value to the data field of the new node.
3. Make the next field of the new node point to NULL.
4. Make START point to the new node.

If the linked list is not empty, you can insert a node at:

- Beginning of the list
- Between two nodes in the list
- End of the list

Inserting a Node in a Singly-Linked List (cont.)

The following algorithm depicts the logic to insert a node at the beginning of the linked list:

1. Allocate memory for the new node.
2. Assign a value to the data field of the new node.
3. Make the next field of the new node point to START
(that is the first node in the list).
4. Make START point to the new node.

Inserting a Node in a Singly-Linked List (cont.)

The following algorithm depicts the logic to insert a node at the end of the linked list:

5. Allocate memory for the new node.
6. Assign a value to the data field of the new node.
7. If START is NULL (if the list is empty), then:
 - I. Make START point to the new node.
 - II. Make LAST point to the new node.
 - III. Go to the step 6.
8. Make the next field of LAST point to the new node.
9. Mark the new node as LAST.
10. Make the next field of the new node point to NULL.

Deleting a Node from a Singly-Linked List

Delete operation in a linked list refers to the process of removing a specified node from the list.

You can delete a node from the following places in a linked list:

- Beginning of the list
- Between two nodes in the list
- End of the list

Deleting a Node from a Singly-Linked List (cont.)

The following algorithm depicts the logic to delete a node from the beginning of a list:

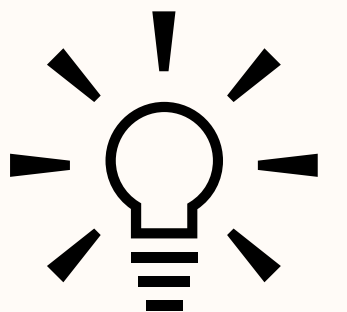
- Mark the first node in the list as current.
- Make START point to the next node in the sequence.
- Release the memory for the node marked as current.

Activity

Write a program to implement the insert, search, delete, and traverse operations on a singly-linked list, which stores the records of the students in a class.

Each record holds the following information:

- Roll number of the student
- Name of the student



Why Do We Need a Double-Linked List?

Consider a sorted list of 1,00,000 numbers stored in a linked list.

If you want to display these numbers in ascending order, what will you do?



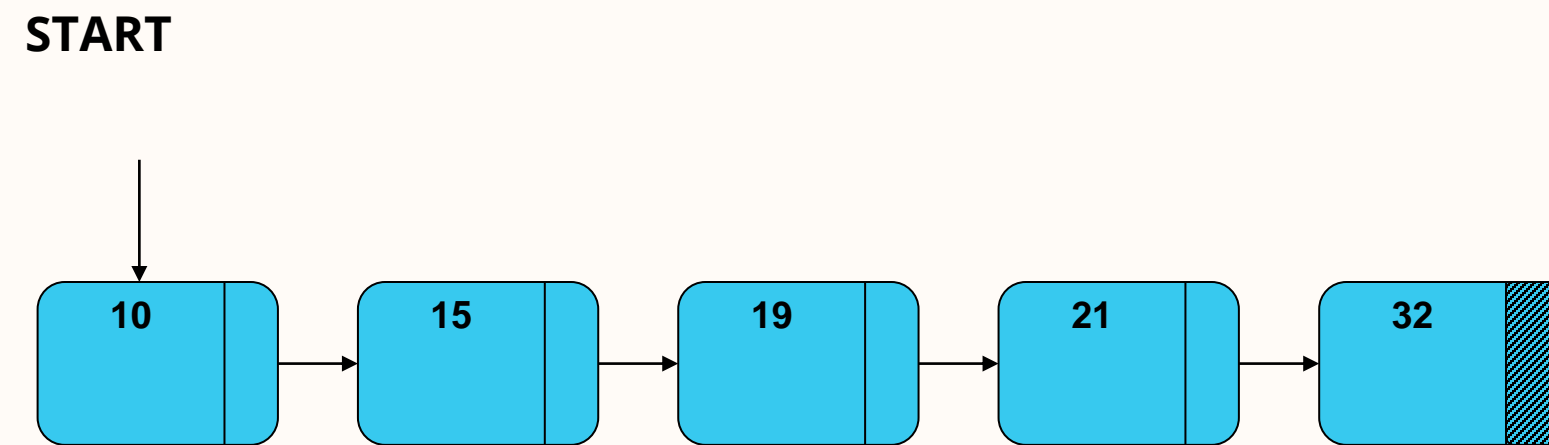
What is a Double-Linked List?

A Doubly linked list is a type of linked list in which each node apart from storing its data has two links:

- The first link points to the previous node in the list and the second link points to the next node in the list.
- The first node of the list has its previous link pointing to NULL similarly the last node of the list has its next node pointing to NULL.

Implementing a Doubly-Linked List

Algorithm to reverse a singly-linked list:

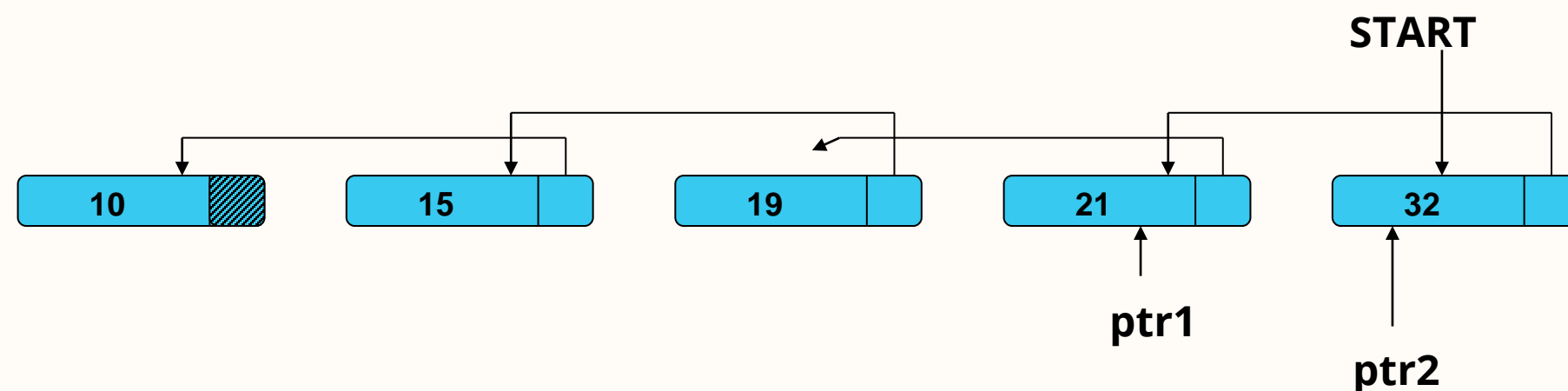
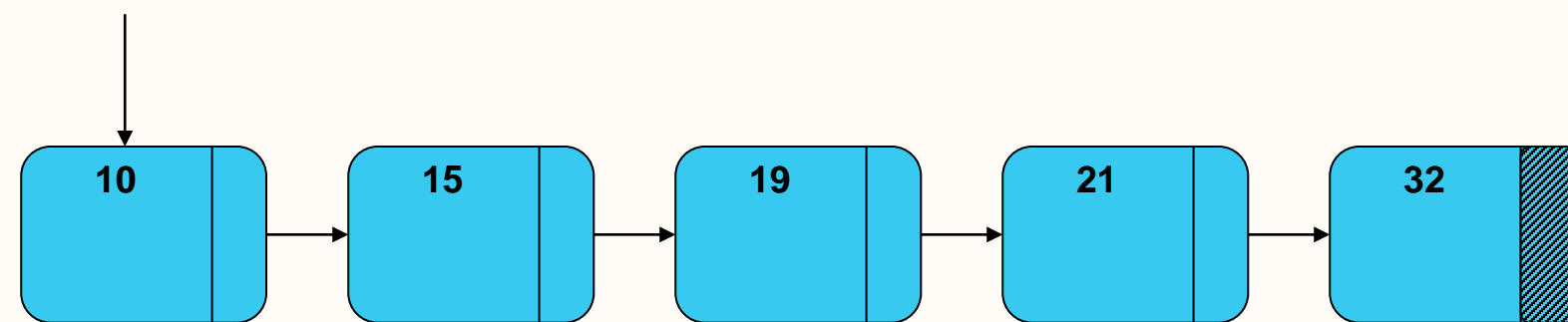


1. Declare three variables/pointers ptr1, ptr2, and ptr3.
2. If there is only one node in the list: Exit.
3. Mark the first node in the list as ptr1.
4. Mark the successor of ptr1 as ptr2.
5. Mark the successor of ptr2 as ptr3.
6. Make the next field of ptr1 point to NULL.
7. Make the next field of ptr2 point to ptr1.
8. Repeat until ptr3 becomes NULL:
 - I. Set ptr1 = ptr2.
 - II. Set ptr2 = ptr3.
 - III. Make ptr3 point to the next node in its sequence.
 - IV. Make the next field of ptr2 point to ptr1.
9. Make START point to ptr2.

Implementing a Doubly-Linked List (cont.)

Algorithm to reverse a singly-linked list:

START



ptr3 = NULL

1. Declare three variables/pointers ptr1, ptr2, and ptr3.
2. If there is only one node in the list: Exit.
3. Mark the first node in the list as ptr1.
4. Mark the successor of ptr1 as ptr2.
5. Mark the successor of ptr2 as ptr3.
6. Make the next field of ptr1 point to NULL.
7. Make the next field of ptr2 point to ptr1.
8. Repeat until ptr3 becomes NULL:
 - I. Set ptr1 = ptr2.
 - II. Set ptr2 = ptr3.
 - III. Make ptr3 point to the next node in its sequence.
 - IV. Make the next field of ptr2 point to ptr1.
9. Make START point to ptr2.

Implementing a Doubly-Linked List (cont.)

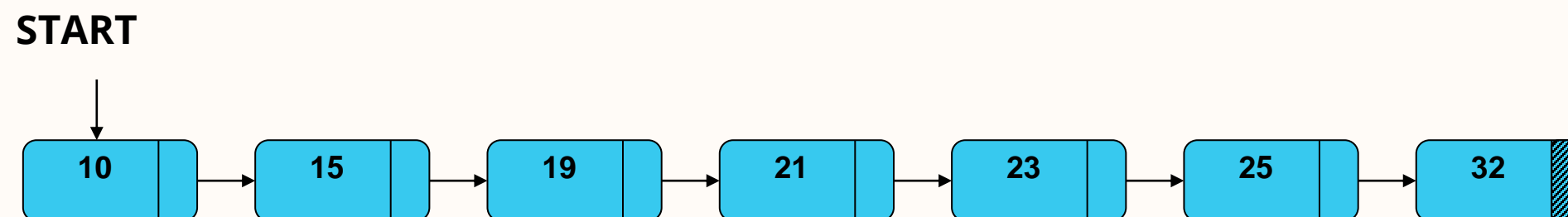
What is the problem with the previous algorithm?

- You need to adjust the links for all three variables whenever you visit the next node.
- This approach is inefficient and time-consuming for large lists.

How can you solve this problem?

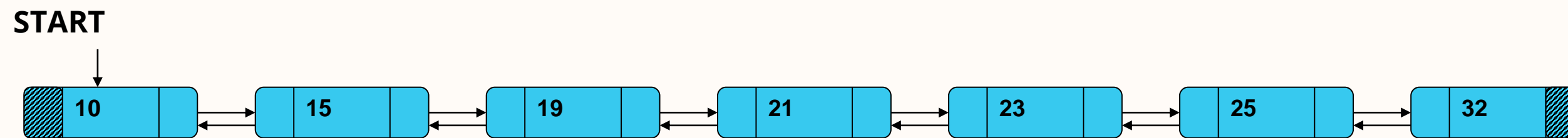
This problem can be solved if each node in the list holds the reference of its preceding node in addition to its next node in the sequence.

Consider the following list:



Implementing a Doubly-Linked List (cont.)

- You can introduce an additional field in each node of a singly-linked list, which would hold the address of its previous node.
- Such a type of list is known as a doubly-linked list.



Structure of a doubly-linked list

Representing a Doubly-Linked List

A doubly-linked list is represented in a program by defining two classes.

Node class: In a doubly-linked list, each node needs to store:

- The information
- The address of the next node in sequence
- The address of the previous node

Representing a Doubly-Linked List

```
def insert_at_end(self, data):
    if self.start_node is None:
        new_node = Node(data)
        self.start_node = new_node
        return
    n = self.start_node
    while n.nref is not None:
        n = n.nref
    new_node = Node(data)
    n.nref = new_node
    new_node.pref = n
```

Inserting Items at the End

```
def insert_at_start(self, data):
    if self.start_node is None:
        new_node = Node(data)
        self.start_node = new_node
        print("node inserted")
        return
    new_node = Node(data)
    new_node.nref =
self.start_node
self.start_node.pref =
new_node
self.start_node = new_node
```

Inserting Items at the Start

Deleting Nodes from a Doubly-Linked List

Deleting Nodes from a Doubly-Linked List

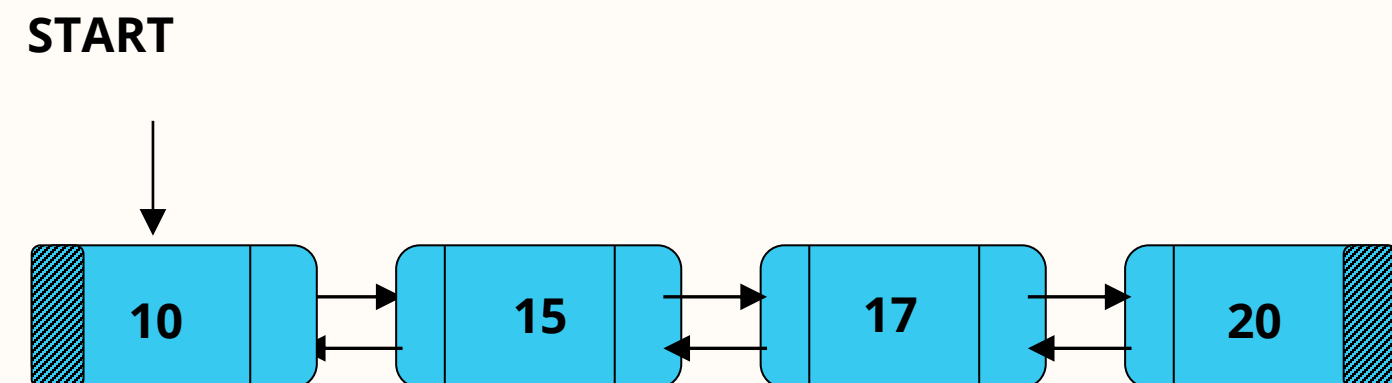
You can delete a node from one of the following places in a doubly-linked list:

- Beginning of the list
- Between two nodes in the list
- End of the list

Deleting Nodes from a Doubly-Linked List (cont.)

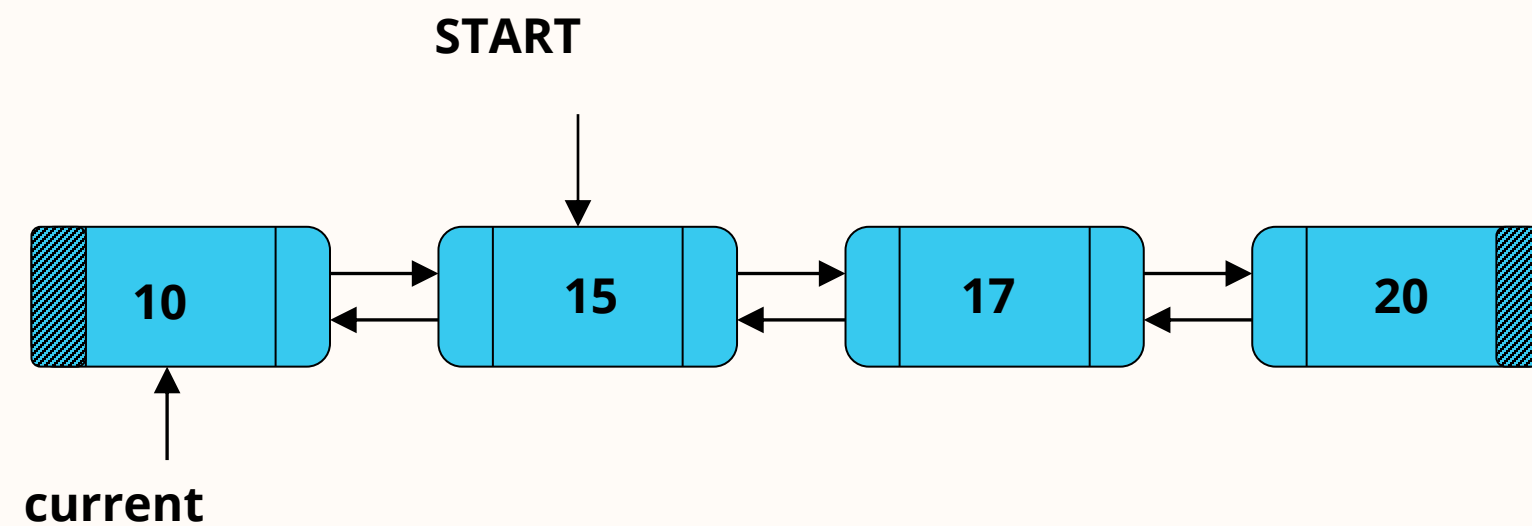
Algorithm to delete a node from the beginning of the doubly-linked list:

1. Mark the first node in the list as current.



Deleting Nodes from a Doubly-Linked List (cont.)

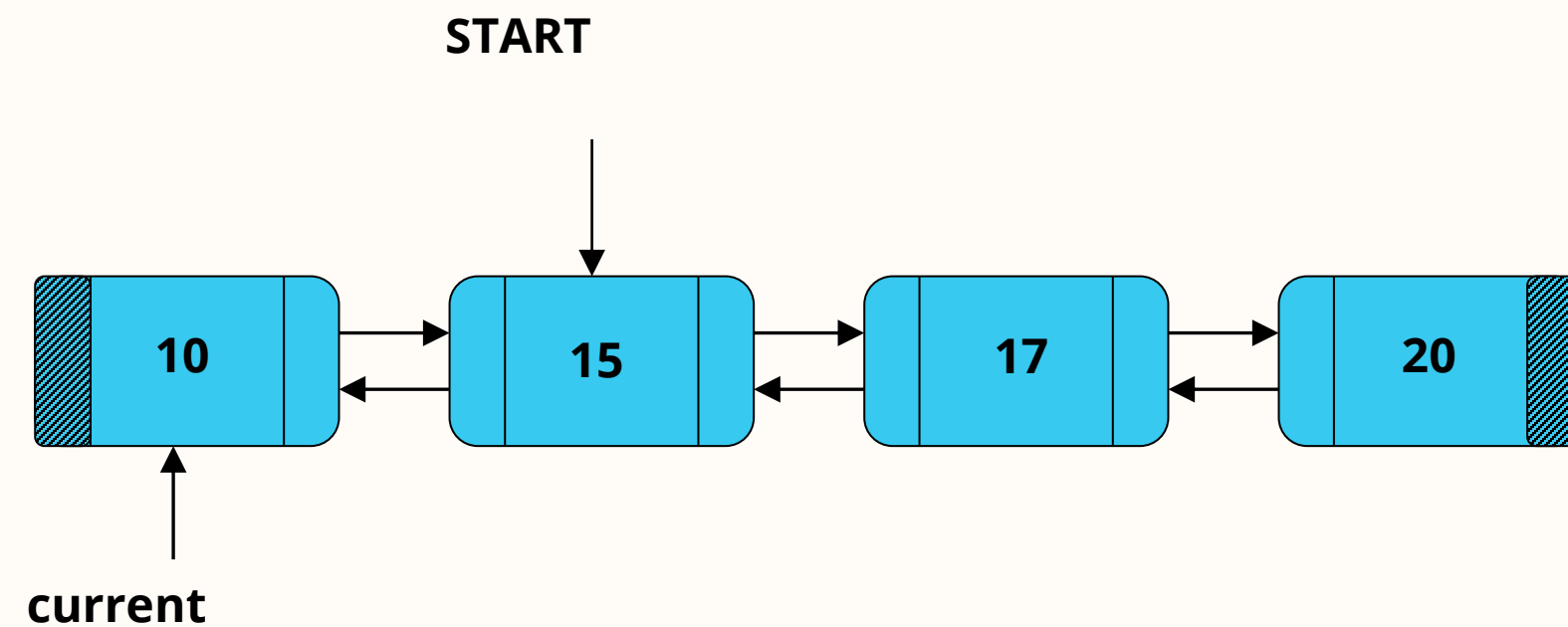
2. Make START point to the next node in the sequence.



START = START.next

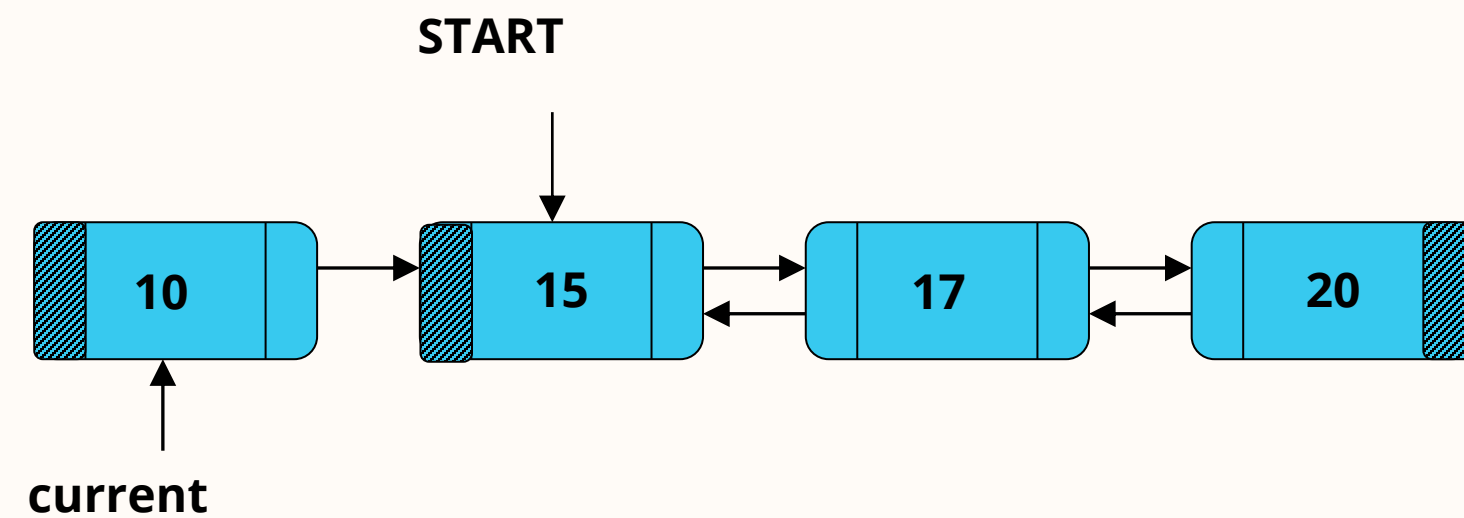
Deleting Nodes from a Doubly-Linked List (cont.)

3. If START is not NULL: /* If the node to be deleted is not the only node in the list */



Deleting Nodes from a Doubly-Linked List (cont.)

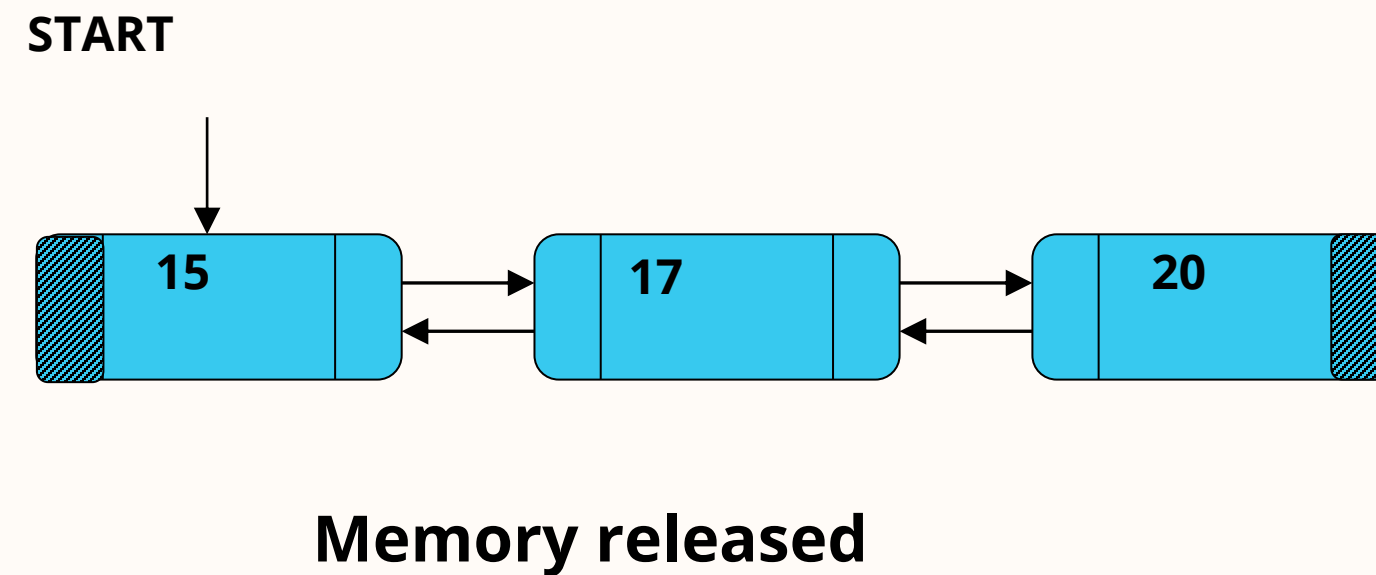
4. Assign NULL to the prev field of the node marked as START.



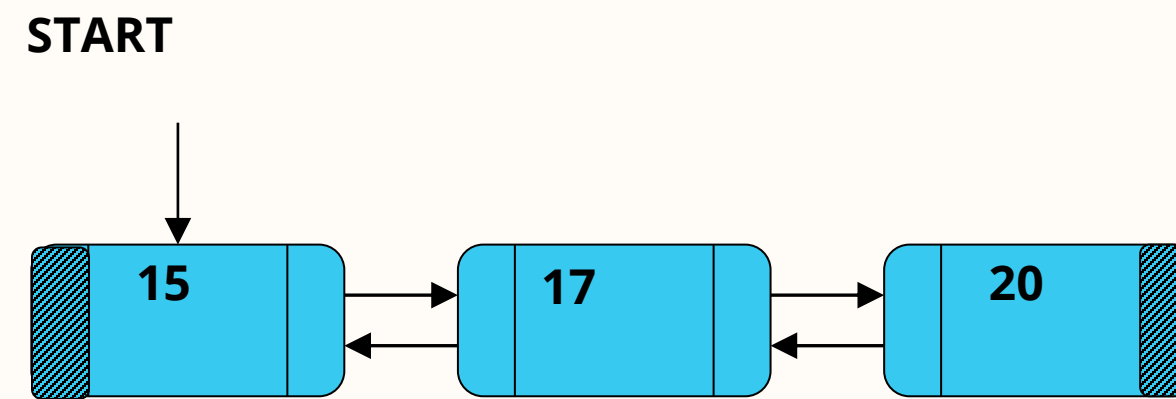
START. prev = NULL

Deleting Nodes from a Doubly-Linked List (cont.)

5. Release the memory of the node marked as current.



Deleting Nodes from a Doubly-Linked List (cont.)



Delete operation complete

Deleting Nodes from a Doubly-Linked List (cont.)

The following algorithm deletes a node between two nodes and from the end of the doubly-linked list:

Mark the node to be deleted as current and its predecessor as previous. To locate previous and current, execute the following steps:

1. Make previous point to NULL.
2. Make current point to the first node in the linked list(that is, Set current = START).
3. Repeat the steps, d and e, until either the value of current is same as the value to be deleted or current becomes NULL.
4. Make previous point to current.
5. Make current point to the next node in the sequence.

If current is NULL:

1. Display "Value to be deleted not found in the list".
2. Exit.

Deleting Nodes from a Doubly-Linked List (cont.)

- If previous is not NULL:
Make the next field of previous point to the successor of current.
- If previous is NULL:
Make START point to its successor.
- If the successor of current exists:
Make the prev field of the successor of current point to previous.
- Release the memory of the node marked as current.

Question

To which one of the following options does the next pointer of the last node in a doubly-linked list point?

- A. NULL
- B. First node
- C. START





Questions?

Summary

In this session, you learned:

- A singly-linked list is a simple data structure that acts as a base for the various other data structures.
- A singly-linked list is represented in a program by defining the following classes:
- A class that represents a node in a linked list
- A class that represents a linked list
- A doubly-linked list in a program can be represented by declaring the following classes:
- A class that represents a node in a doubly-linked list
- A class that represents a doubly-linked list