

Data Structures and Algorithms

Instructor: Tesfaye Guta [M.Sc.]

tesfayeguta@haramaya.edu.et

Haramaya University

Chapter Three: Linked Lists

 This chapter covers:

- Singly linked lists
- Doubly linked lists
- Circular lists
- Skip lists
- Self-organizing lists
- Sparse tables

Data Structure and Applications

☰ In computer Science data can be seen from three different levels:

i) **Application (user) level** – is a level which models real life data in a specific context.

- It answers the question “**why data is needed to be modelled?**”

- e.g. HU Library, HU students

...

ii) **Logical or abstract data type level** – indicates the abstract view of the domain and operations.

- It specifies the type of data stored and the operations supported on them.
- This allows the user to create data types to their own specifications.
- The purpose of the abstract data type is to answer the question “**what data is to be stored?**” and “**what operations are made on the data?**”
- Example: data to be stored (domain of application) – collection books
 - valid operations on this domain
 - *searching books*
 - *Sorting collection of book by title, author,...*
 - *Inserting (adding) new books*
 - *Reserving books*
 - *Selling a book (not valid operation)*
 - *Selling a car (not valid operation)*

iii) **Implementation (Data structure) level** – is a specific representation of the data structure to hold the data items and the coding for operations.

- It answers “**how is data represented?**”.

- Example:

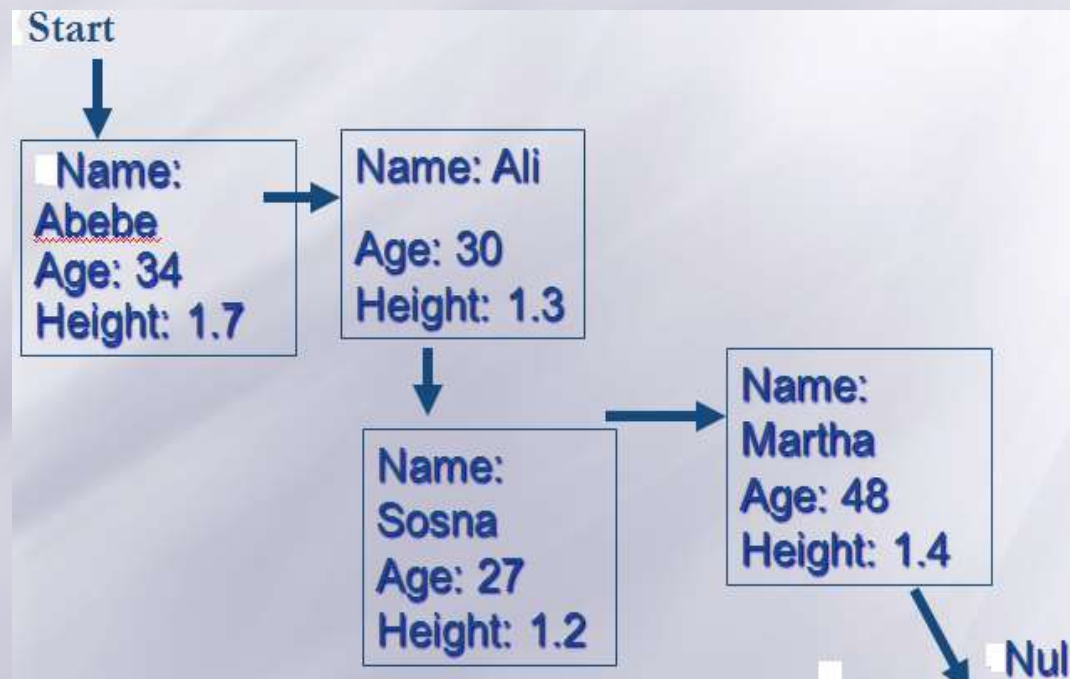
a) `int BookId[1000];`
`bool BooksRes[1000];`
`char BookTitle[1000][15];`

b) `struct HULibBook`
`{`
`int BookId;`
`bool BookRes;`
`char BookTitle[15];`
`};`
`HULibBook Book[1000];`

c) `struct HULibBook`
`{`
`int BookId;`
`bool BookRes;`
`char BookTitle[15];`
`HULibBook *Next;`
`};`
`HULibBook *Head=NULL;`

Creating Linked Lists in C++

- A linked list is a data structure which is built from structures and pointers.
- It forms a chain of "nodes" with pointers representing the links of the chain and holding the entire thing together.
- Graphically,



..

- ☰ This linked list has four nodes in it, each with a link to the next node in the series.
- ☰ The last node has a link to the special value **NULL**, which any pointer (whatever its type) can point to, to show that it is the last link in the chain.
- ☰ There is also another special pointer, called **Start**, which points to the first link in the chain so that we can keep track of it.

Definition and Operations on Data Structures

i) Definition of Data Structures

A) **Array:**

```
struct Node
{
    char name[15];
    int age;
    float height;
};
Node person[1000];
```

...

B) Single Linked List

```
struct node {  
    char name[20]; // Name of up to 20 chars  
    int age; // BirthDate would be better  
    float height; // In meters  
    node *nxt; // Pointer to next node  
};  
node *start_ptr = NULL;  
//start_ptr will permanently point to the start of the list
```

...

- ≡ The key part of a linked list is a structure, which holds the data for each node (the name, address, age or whatever for the items in the list), and, most importantly, a pointer to the next node.
- ≡ The important part of the above structure is the line before the closing curly brackets.
- ≡ This gives a pointer to the next node in the list.
- ≡ This is the only case in C++ where you are allowed to refer to a data type (in this case node) before you have even finished defining it!

ii) Storing and accessing Data files

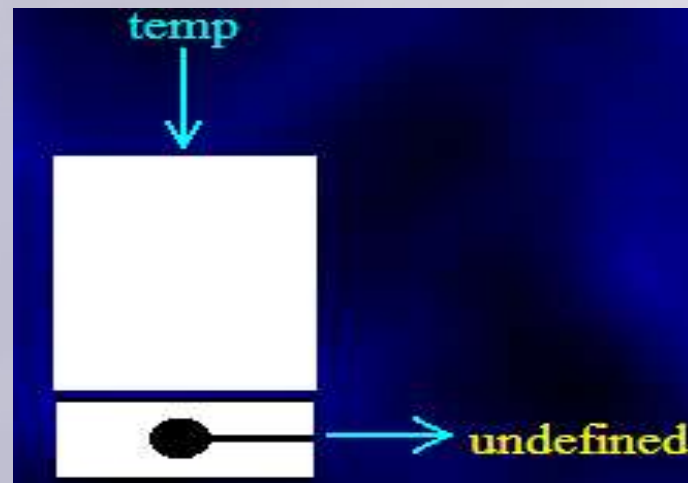
≡ Array:

- `person[0].name = "Abebe"`
- `person[0].age = 20;`
- `person[0].height = 1.6`

≡ Single Linked List

- Firstly, we declare the space for a pointer item and assign a temporary pointer to it. This is done using the new statement as follows:

- `temp = new node;`



☰ Then we will fill details of the person

```
cout << "Please enter the name of the person: ";  
cin >> temp->name;  
cout << "Please enter the age of the person : ";  
cin >> temp->age;  
    cout << "Please enter the height of the person: ";  
cin >> temp->height;  
temp->nxt = NULL;
```

☰ The last line sets the pointer from this node to the next to **NULL**, indicating that this node, when it is inserted in the list, will be the last node.

iii) The full code for adding a node at the end of the list is shown below, in its own little function:

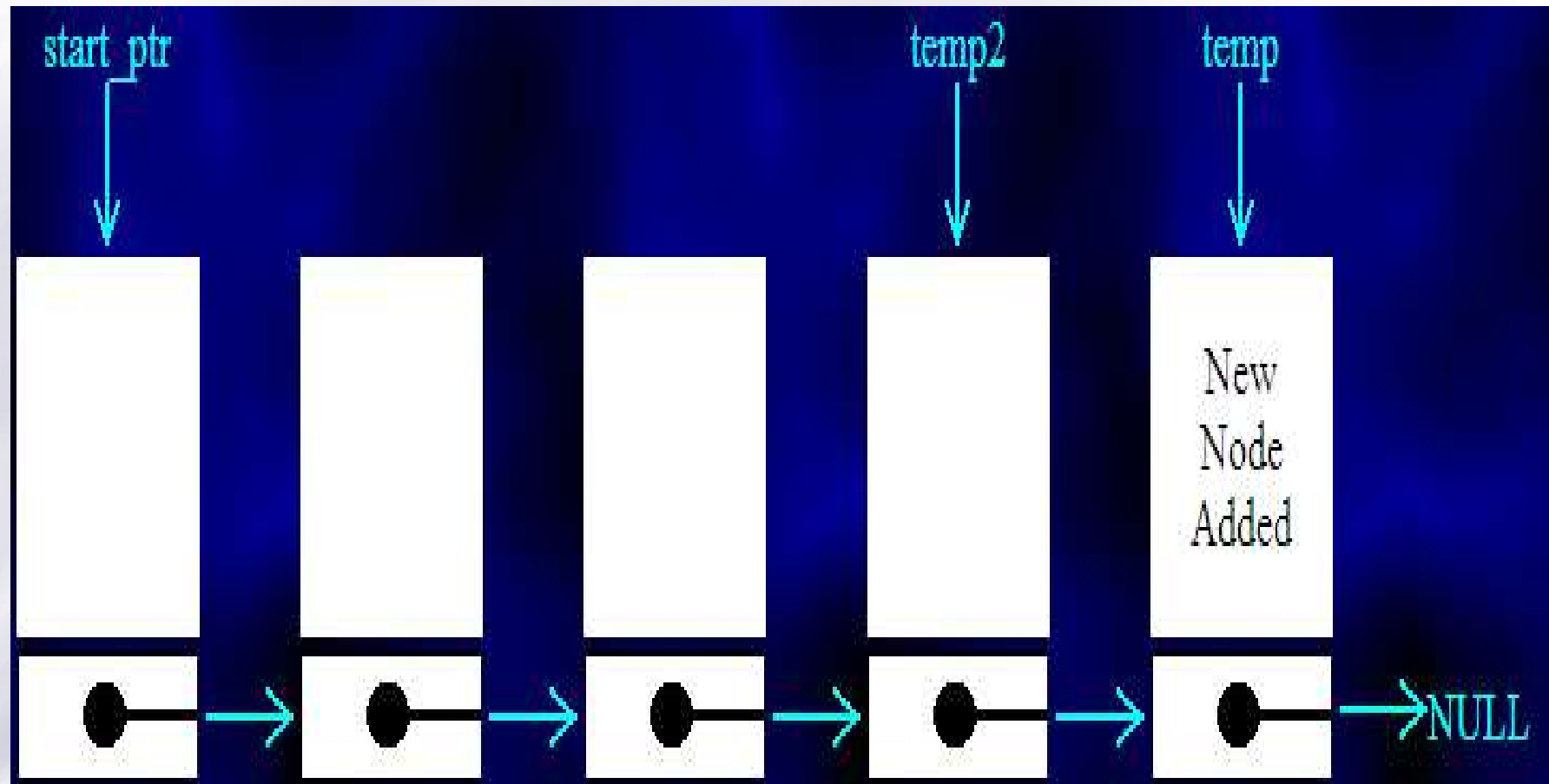
```
void add_node_at_end ()
{   node *temp, *temp2;    // Temporary pointers

    // Reserve space for new node and fill it with data
    temp = new node;
    cout << "Please enter the name of the person: ";
    cin >> temp->name;
    cout << "Please enter the age of the person : ";
    cin >> temp->age;
    cout << "Please enter the height of the person : ";
    cin >> temp->height;
    temp->nxt = NULL;

    // Set up link to this node
    if (start_ptr == NULL)
        start_ptr = temp;
    else
    {   temp2 = start_ptr;
        // We know this is not NULL - list not empty!
        while (temp2->nxt != NULL)
        {   temp2 = temp2->nxt;
            // Move to next link in chain
        }
        temp2->nxt = temp;
    }
}
```

...

Graphically



iv) Traversing (Displaying the list of nodes)

Array:

```
for (i=0;i<=lastperson;i++)  
{  
    cout<<"Person Name:" <<person[i]. name;  
    cout<<"Book Reserved :"<<person[i].age;  
    cout<<"Book Borrowed :"<< person[i]. height;  
}
```

Single Linked List:

Having added one or more nodes, we need to display the list of nodes on the screen.

This is comparatively easy to do. Here is the method:

- Set a temporary pointer to point to the same thing as the start pointer.
- If the pointer points to NULL, display the message "End of list" and stop.
- Otherwise, display the details of the node pointed to by the start pointer.
- Make the temporary pointer point to the same thing as the next pointer of the node it is currently indicating.
- Jump back to step 2.

...

```
temp = start_ptr;
do {
    if (temp == NULL)
        cout << "End of list" << endl;
    else {
        // Display details for what temp points to
        cout << "Name : " << temp->name << endl;
        cout << "Age : " << temp->age << endl;
        cout << "Height : " << temp->height << endl;
        cout << endl; // Blank line
        // Move to next node (if present)
        temp = temp->nxt;
    }
}
while (temp != NULL);
```

v) Deleting a node from the list

☰ When it comes to deleting nodes, we have three choices:

- Delete a node from the start of the list,
- delete one from the end of the list, or
- delete one from somewhere in the middle.

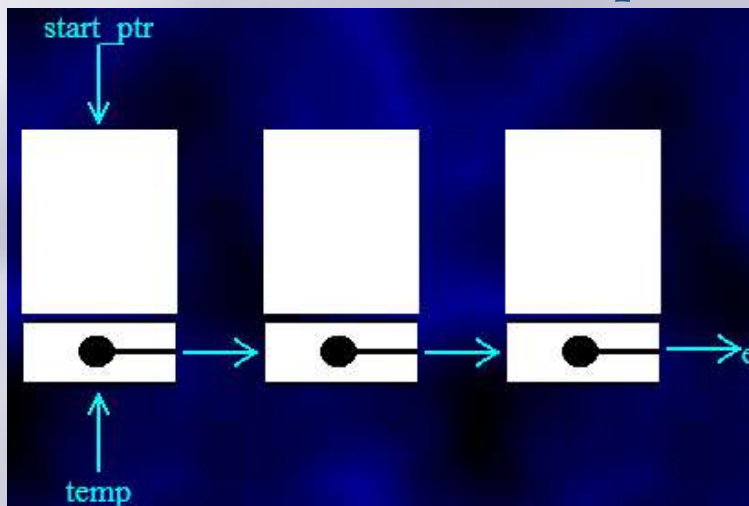
☰ When a node is deleted, the space that it took up should be reclaimed. Otherwise the computer will eventually run out of memory space. This is done with the delete instruction:

- `delete temp; // Release the memory pointed to by temp`

...

- However, we can't just delete the nodes willy-nilly as it would break the chain.
- We need to reassign the pointers and then delete the node at the last moment.
- Here is how we go about deleting the first node in the linked list:

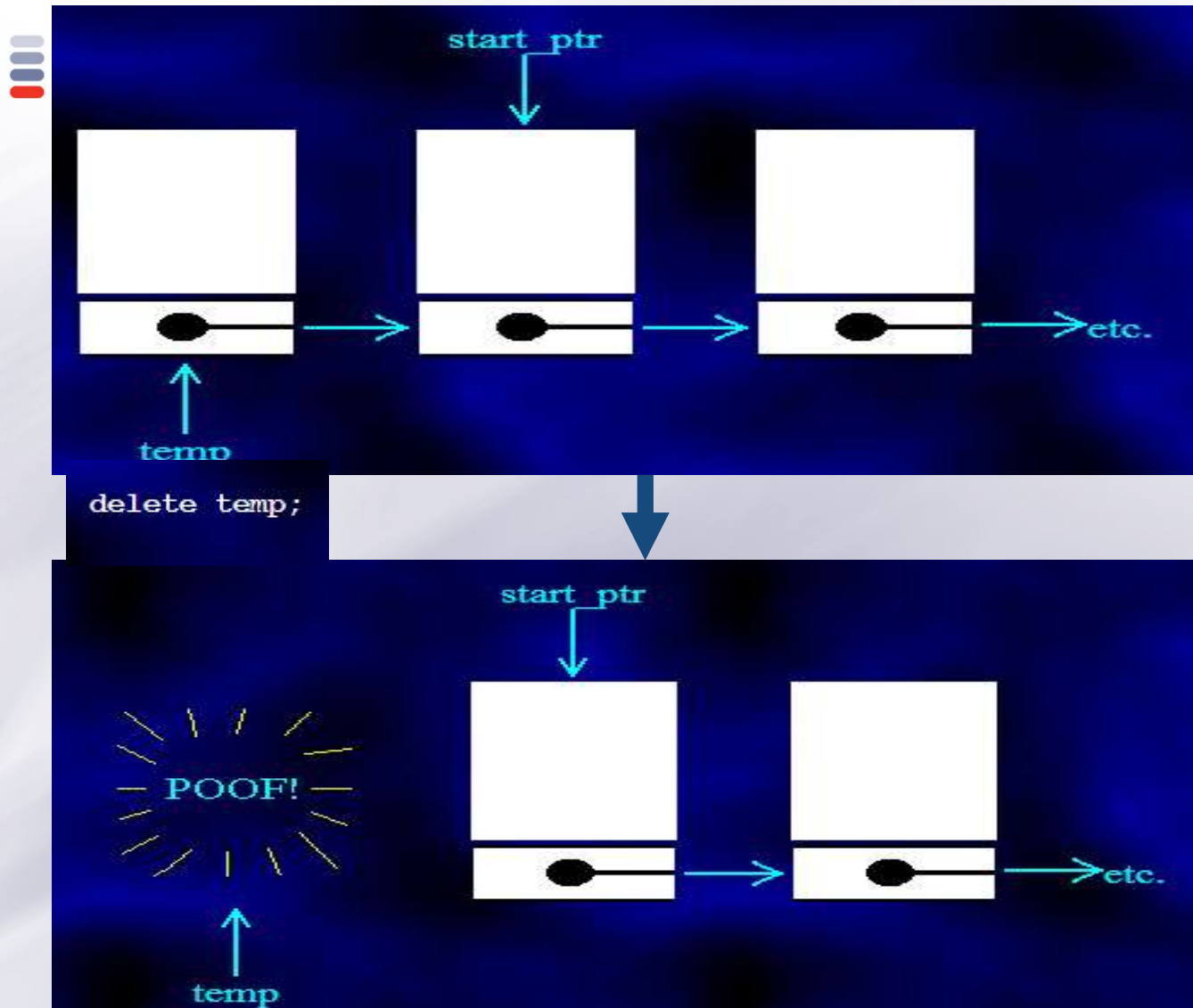
● `temp = start_ptr; // Make the temporary pointer identical to the
// start pointer`



...

- Now that the first node has been safely tagged (so that we can refer to it even when the start pointer has been reassigned), we can move the start pointer to the next node in the chain:
- `start_ptr = start_ptr->nxt; // Second node in chain.`

...



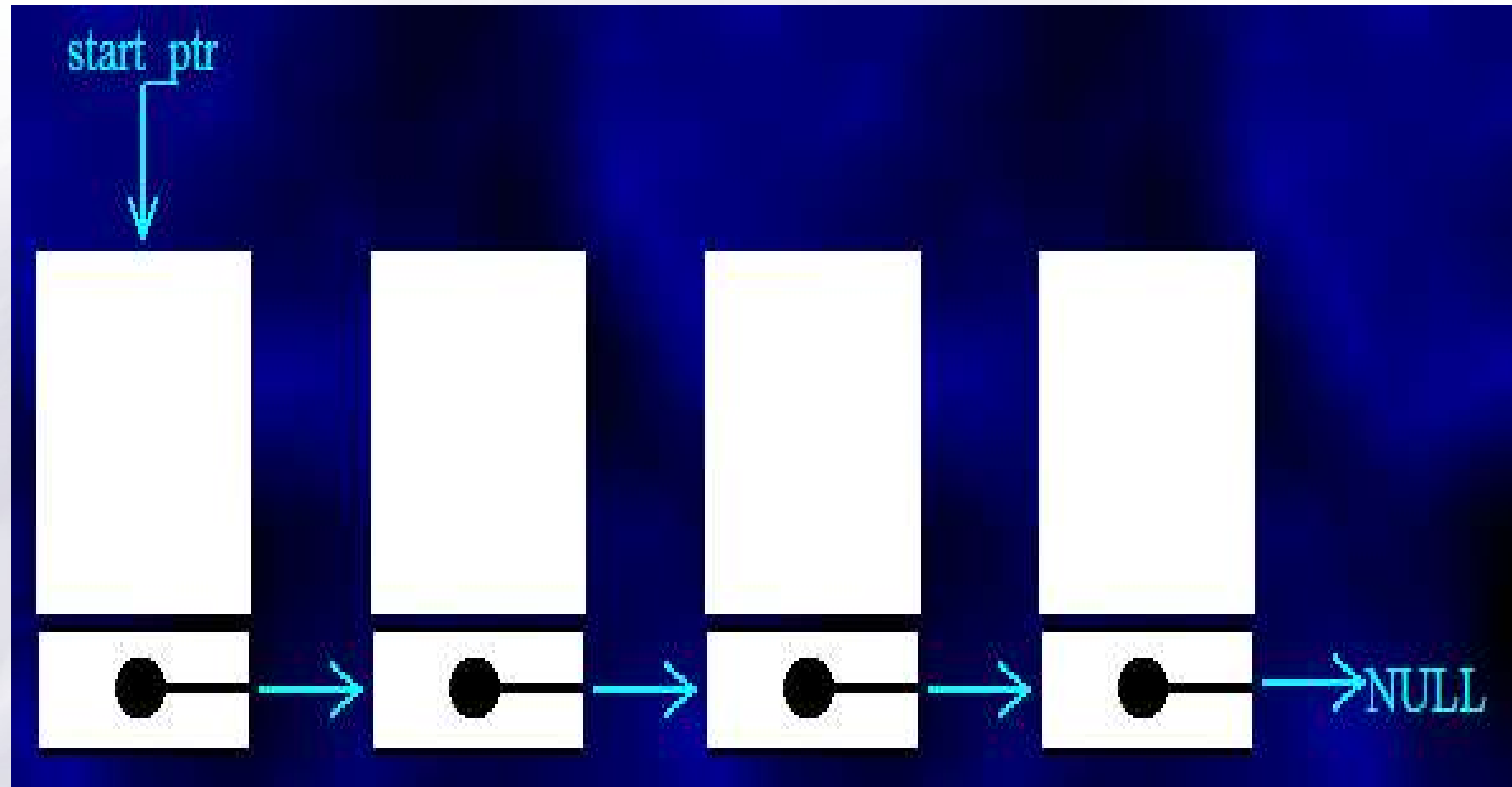
Here is the function that deletes a node from the start:

```
 void delete_start_node()
{
    node *temp;
    temp = start_ptr;
    start_ptr = start_ptr->nxt;
    delete temp;
}
```

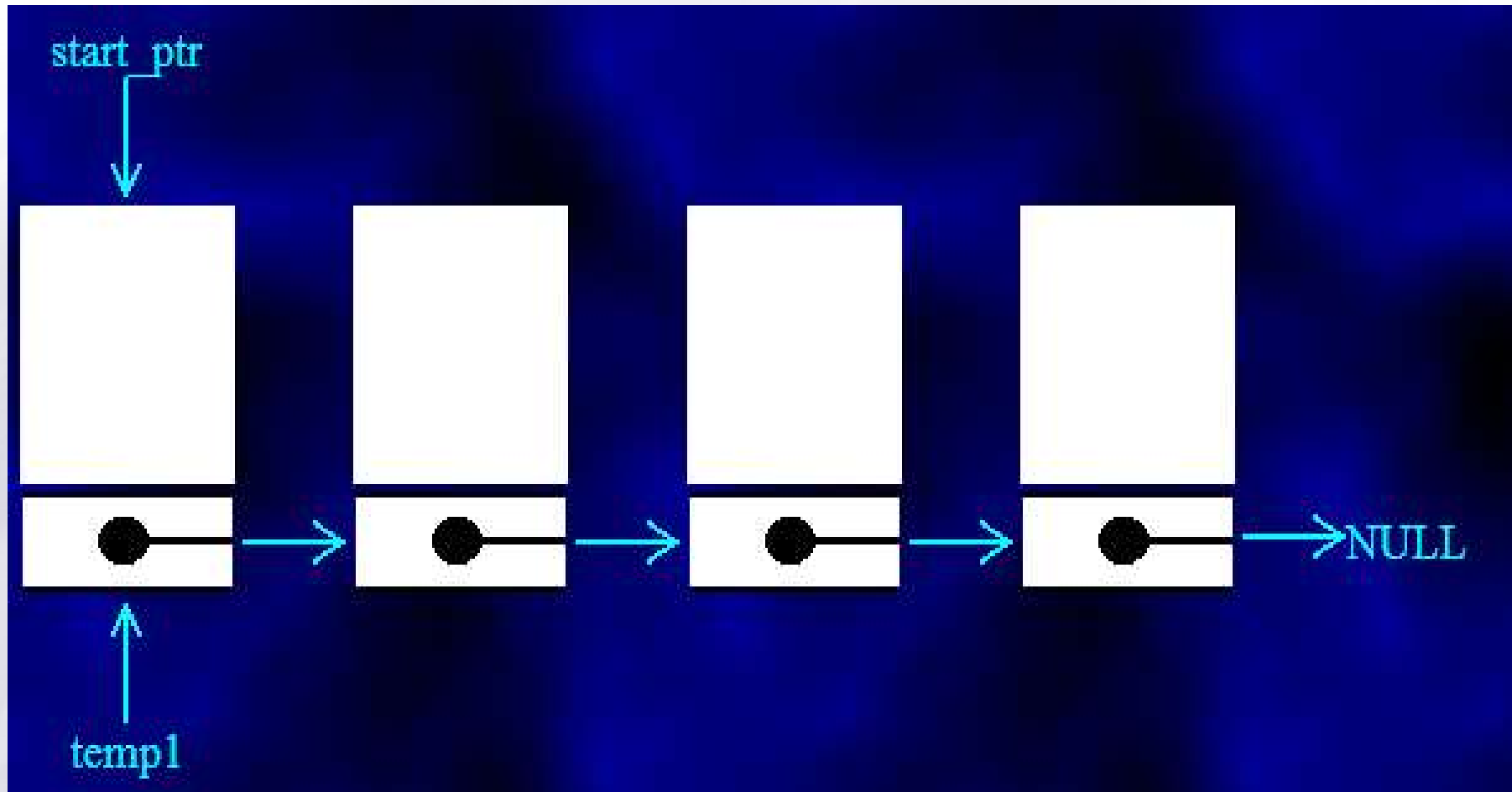
Deleting a node from the end

1. Look at the start pointer. If it is NULL, then the list is empty, so print out a "No nodes to delete" message.
2. Make temp1 point to whatever the start pointer is pointing to.
3. If the nxt pointer of what temp1 indicates is NULL, then we've found the last node of the list, so jump to step 7.
4. Make another pointer, temp2, point to the current node in the list.
5. Make temp1 point to the next item in the list.
6. Go to step 3.
7. If you get this far, then the temporary pointer, temp1, should point to the last item in the list and the other temporary pointer, temp2, should point to the last-but-one item.
8. Delete the node pointed to by temp1.
9. Mark the nxt pointer of the node pointed to by temp2 as NULL - it is the new last node.

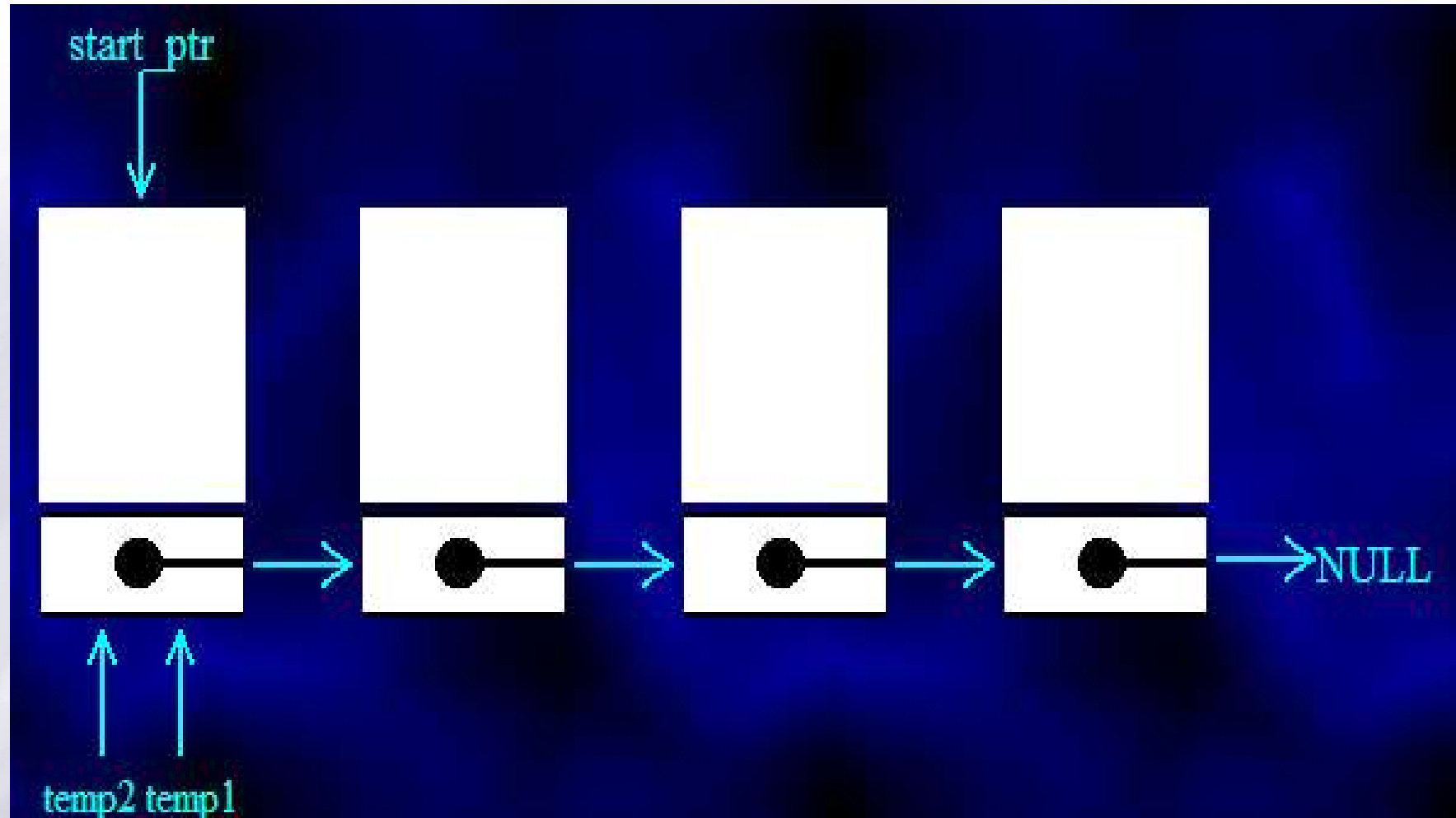
Suppose we want to delete the last node from
this list:



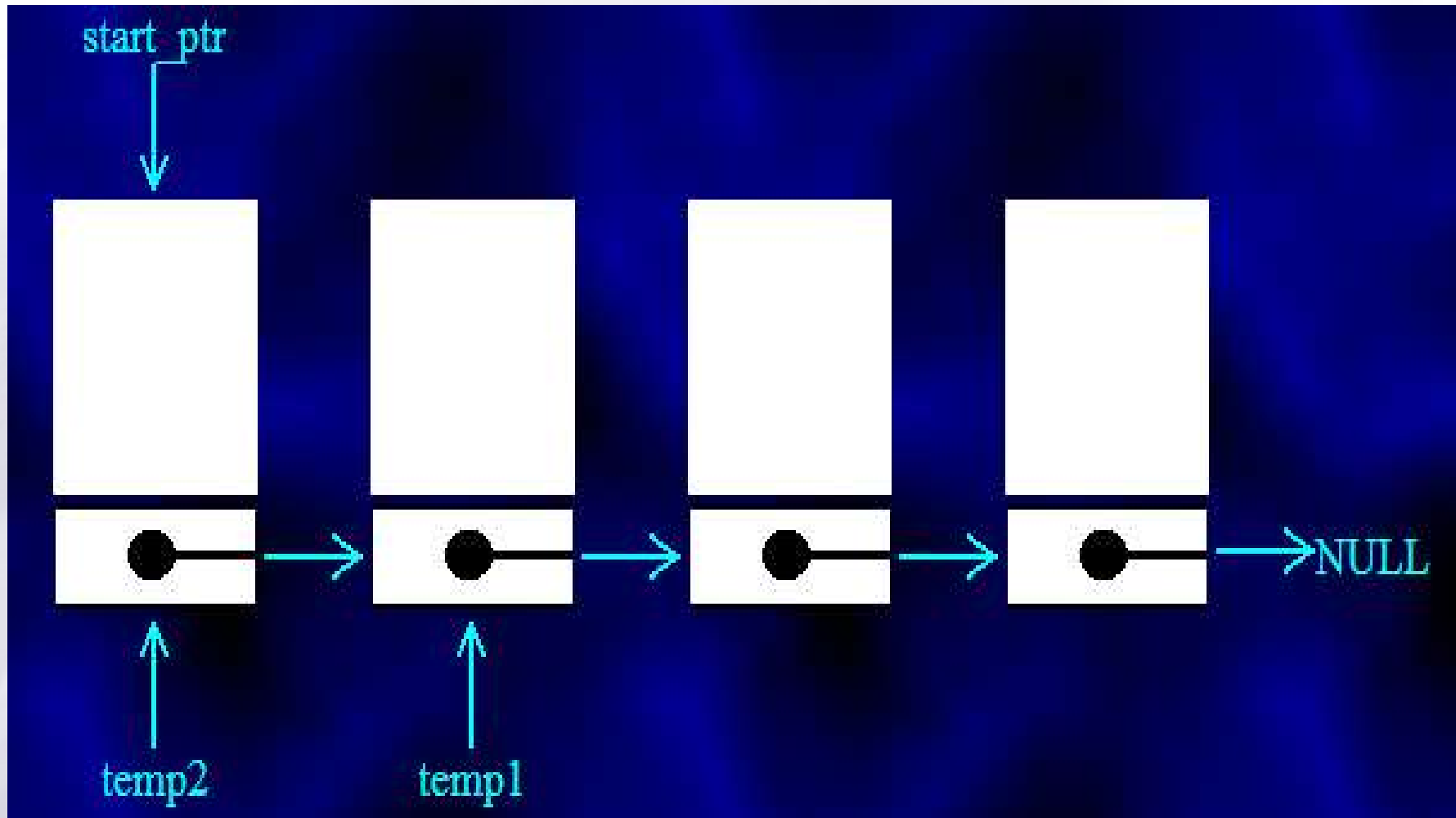
Let's get straight on with step2 - set the pointer temp1 to the same as the start pointer:



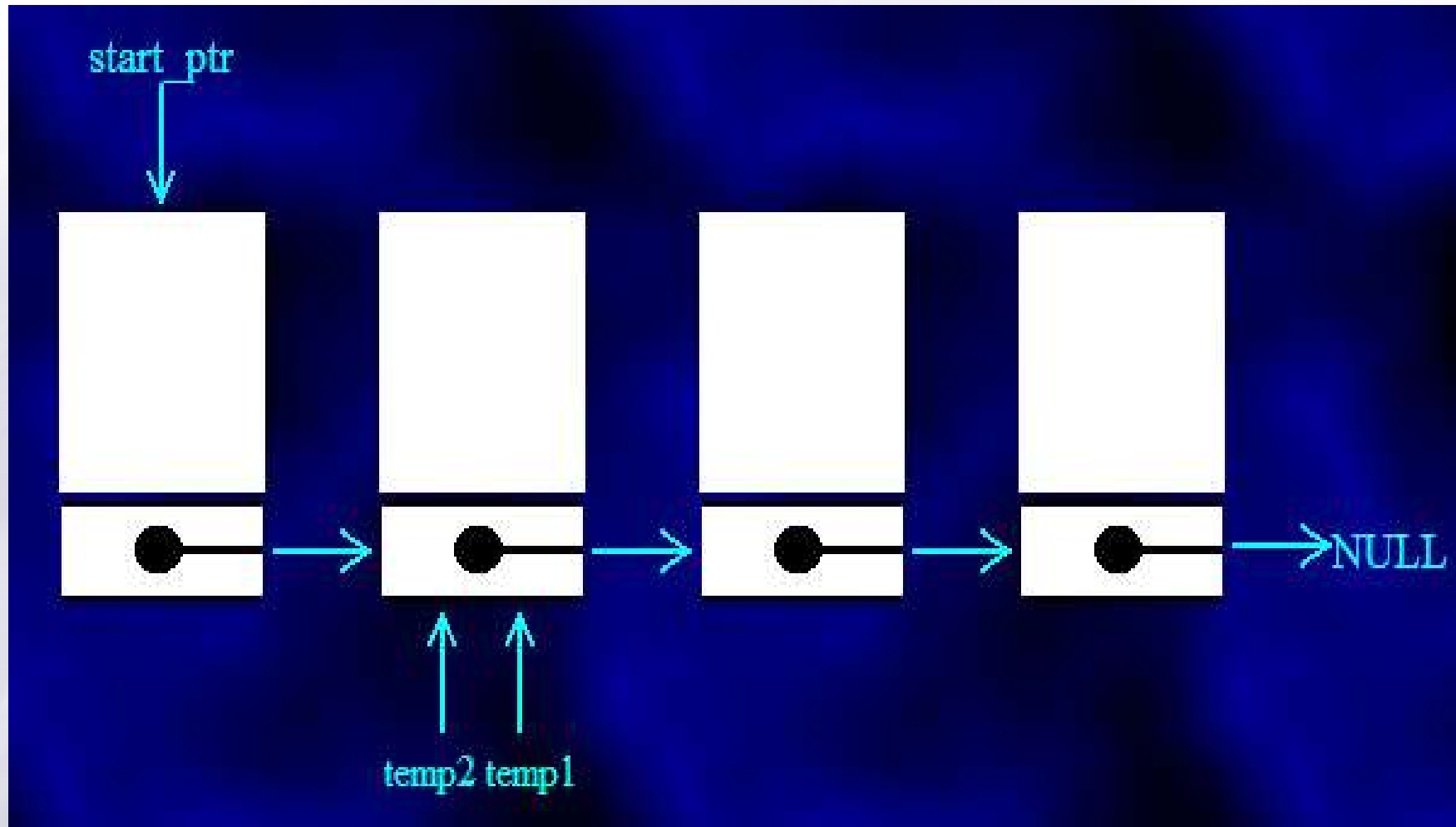
The next pointer from this node isn't NULL, so we haven't found the end node. Instead, we set the pointer temp2 to the same node as temp1



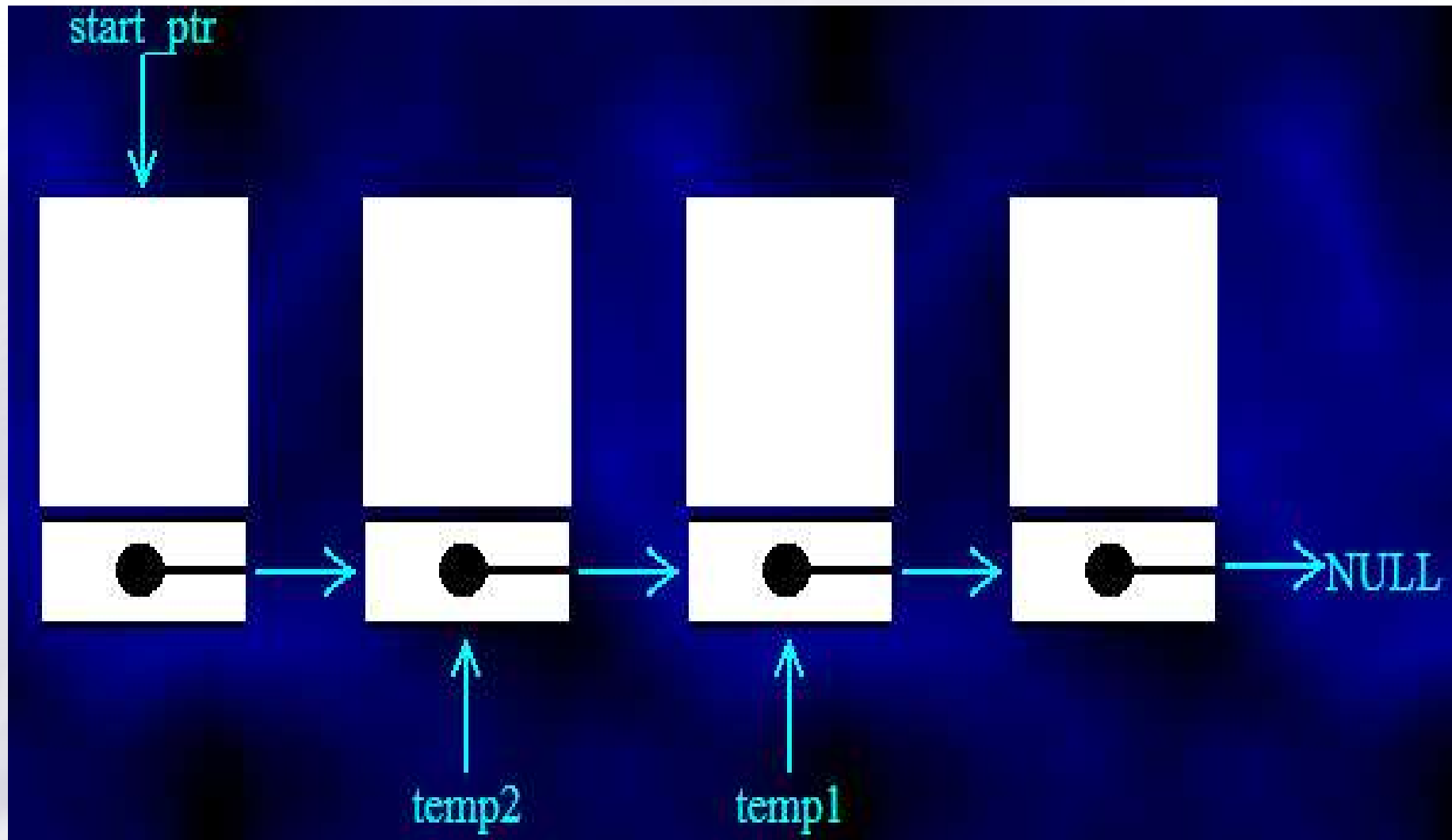
and then move temp1 to the next node in the list:



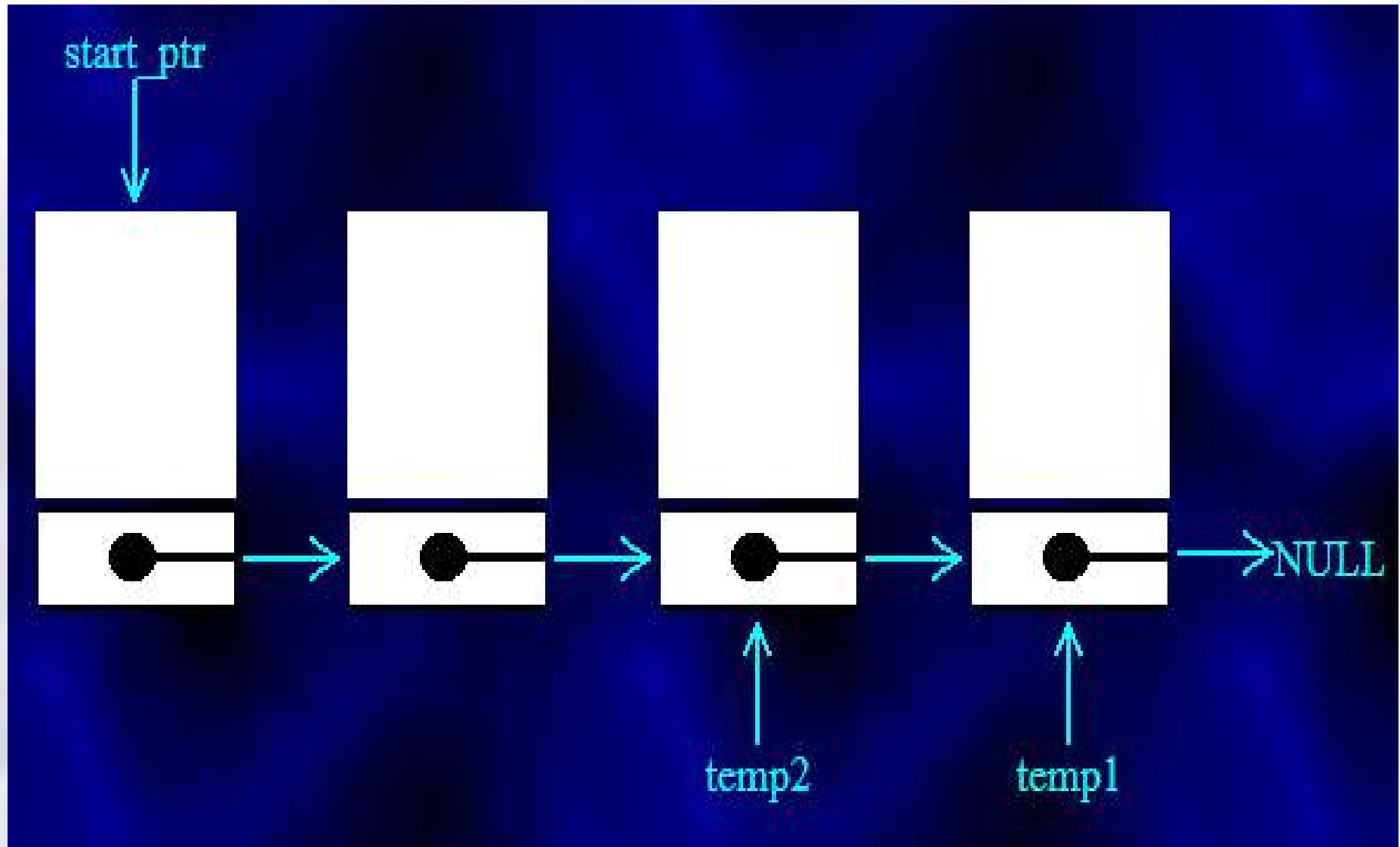
Going back to step 3, we see that temp1 still doesn't point to the last node in the list, so we make temp2 point to what temp1 points to



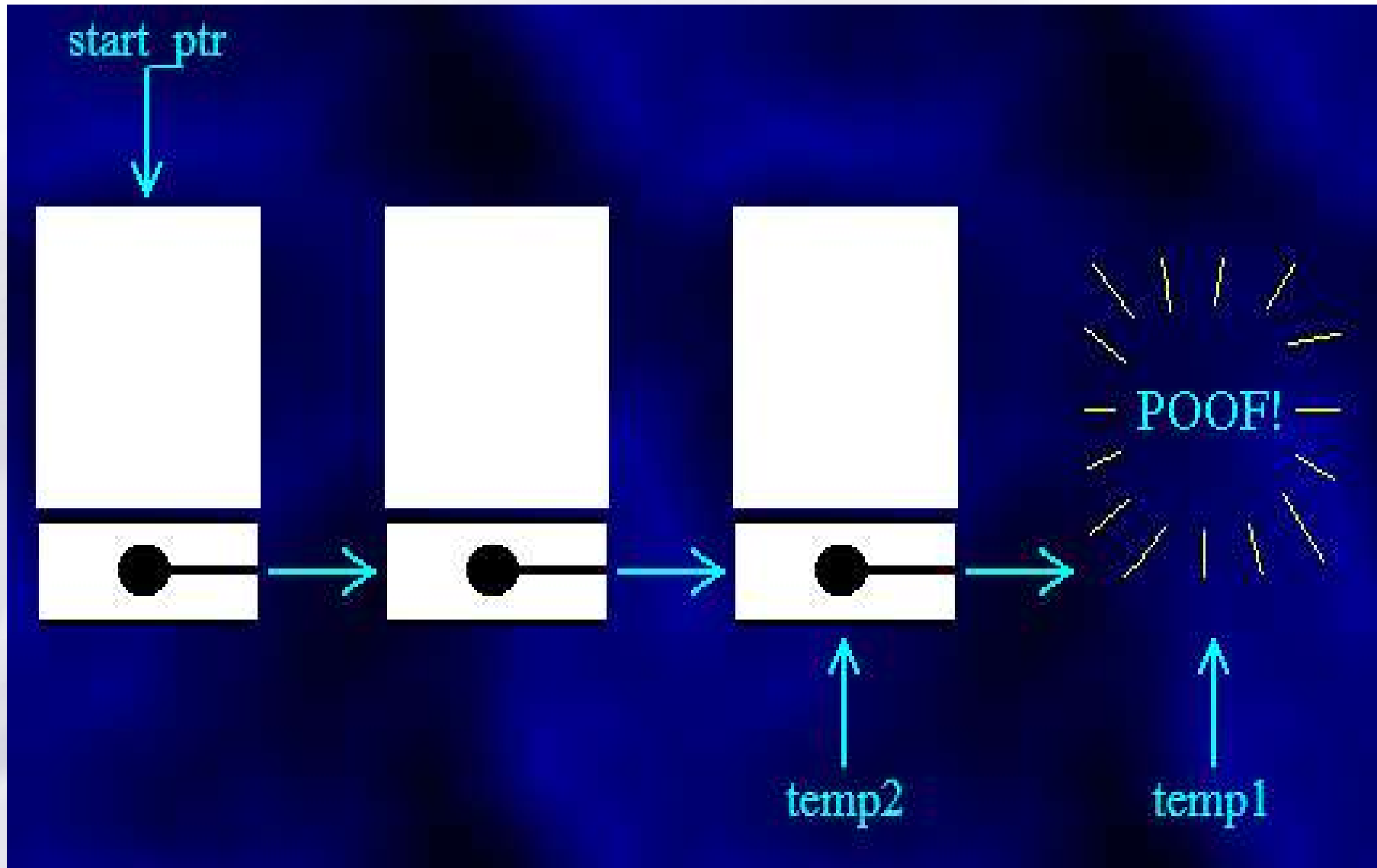
and temp1 is made to point to the next node along:



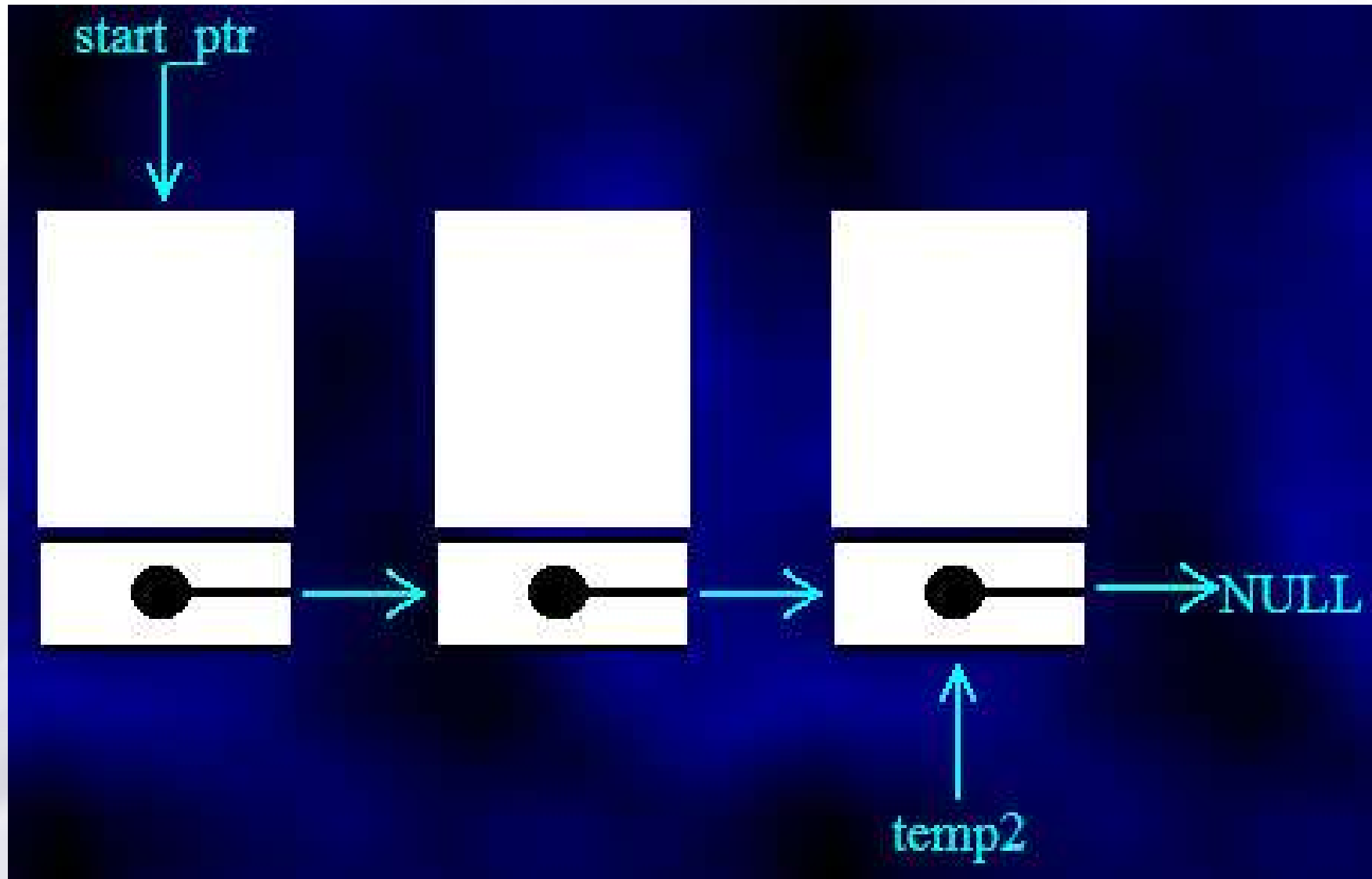
Eventually, this goes on until temp1 really is pointing to the last node in the list, with temp2 pointing to the penultimate node:



Now we have reached step 8. The next thing to do is to delete the node pointed to by temp1



and set the next pointer of what temp2 indicates to NULL:



The code to delete last node.

```
void delete_end_node()
{ node *temp1, *temp2;
if (start_ptr == NULL) cout << "The list is empty!" << endl;
Else
{ temp1 = start_ptr;
  if (temp1->nxt == NULL)
  { delete temp1;
    start_ptr = NULL;
  } Else
  { while (temp1->nxt != NULL)
    { temp2 = temp1;
      temp1 = temp1->nxt;
    }
    delete temp1;
    temp2->nxt = NULL;
  }
}
```

Navigating through the list

☰ Move on:

```
    if (current->nxt == NULL)
cout << "You are at the end of the list." << endl; else current =
    current->nxt;
```

Move Back:

```
if (current == start_ptr)
cout << "You are at the start of the list" << endl; else
{ node *previous; // Declare the pointer
  previous = start_ptr;
  while (previous->nxt != current)
  { previous = previous->nxt;
  }
  current = previous;
}
```

Deleting from the center of the list

```
≡ First navigate to the list (use above code)
if (current->nxt == NULL)
cout << "There is no node after current" << endl; else
{ node *temp;
temp = current->nxt;
current->nxt = temp->nxt; // Could be NULL delete temp; }
```

Here is the code to *add* a node after the current one

```
if (current->nxt == NULL)
add_node_at_end();
else {
node *temp;
new temp;
get_details(temp);
// Make the new node point to the same thing as the
    current node
temp->nxt = current->nxt;
// Make the current node point to the new link in the chain
current->nxt = temp; }
```

Add the node with (Samrawit,
32,1.64) next to Current – Quizz 5%



Start_ptr

current

