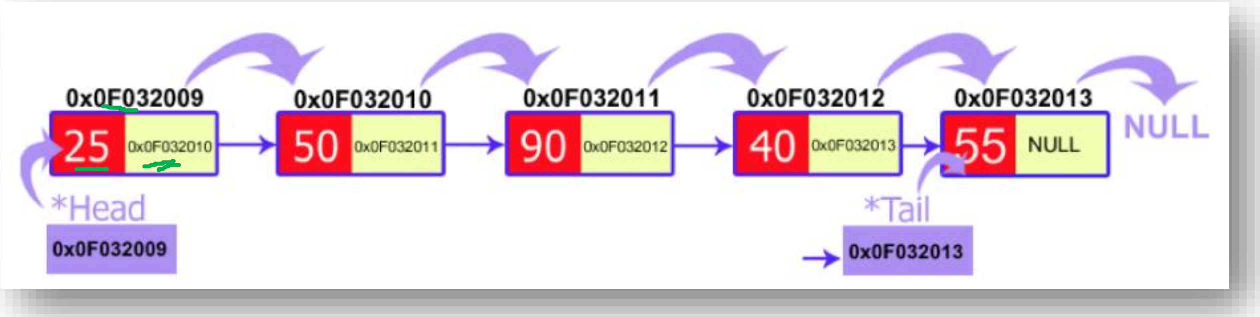
DATA STRUCTURE & PROGRAMMING II

Chapter 9- Linked List



Prepared by:

Dr. VALY Dona, Mr. NOU Sotheany, Mr. BOU Channa

Lecture overview

❑ Overall lectures

1. Introduction to algorithm
2. Basic data types and statements
3. Control structures and Loop
4. Array
5. Data structure
6. Sub-programs

7. Recursive
8. Pointers
9. *Linked Lists*
10. Stacks and Queues
11. Sorting algorithms
12. Trees

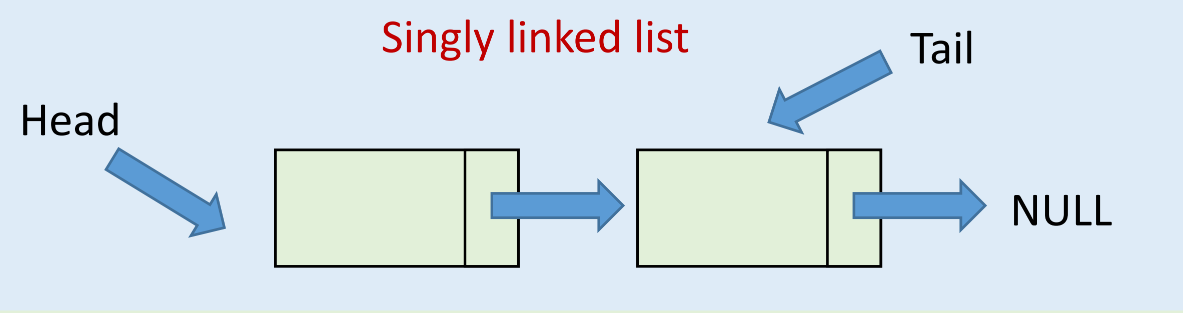
C++

Outline

❑ A Brief of Outline

- What is linked list?
  - Single linked list? Double linked list?
- What are the advantages of using linked list and array?
- Linked list implementation in C++
  - Examples

What is Linked list?



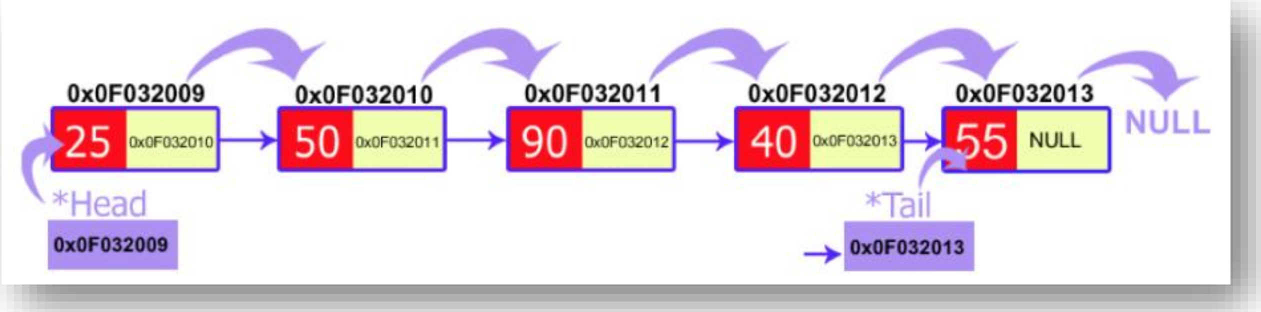
❑ Definition

- **A linked list** is a data structure that can store an indefinite amount of elements (dynamic size)
- In a linked list, each element is linked with each other. Elements in a linked list are accessed sequentially.

```
struct Element
    data: integer
    *next: Element
End struct
```

```
struct List
    n: integer
    *head: Element
    *tail: Element
End struct
```

- Each element contains
    - ✓ **Data**
    - ✓ **A link (pointer)**
        - ✓ to its next element (successor)
        - ✓ and/or to its previous element (predecessor)



- Element = called a ***node***
- In linked list, the first element is ***head*** and the last element is ***tail***

Array Vs. Linked List

❑ Pros and Con

Array

- Fixed size
- Once created, can’t add or reduce number of elements to be stored
- Can random access
- Faster access
  - Elements in contiguous memory locations

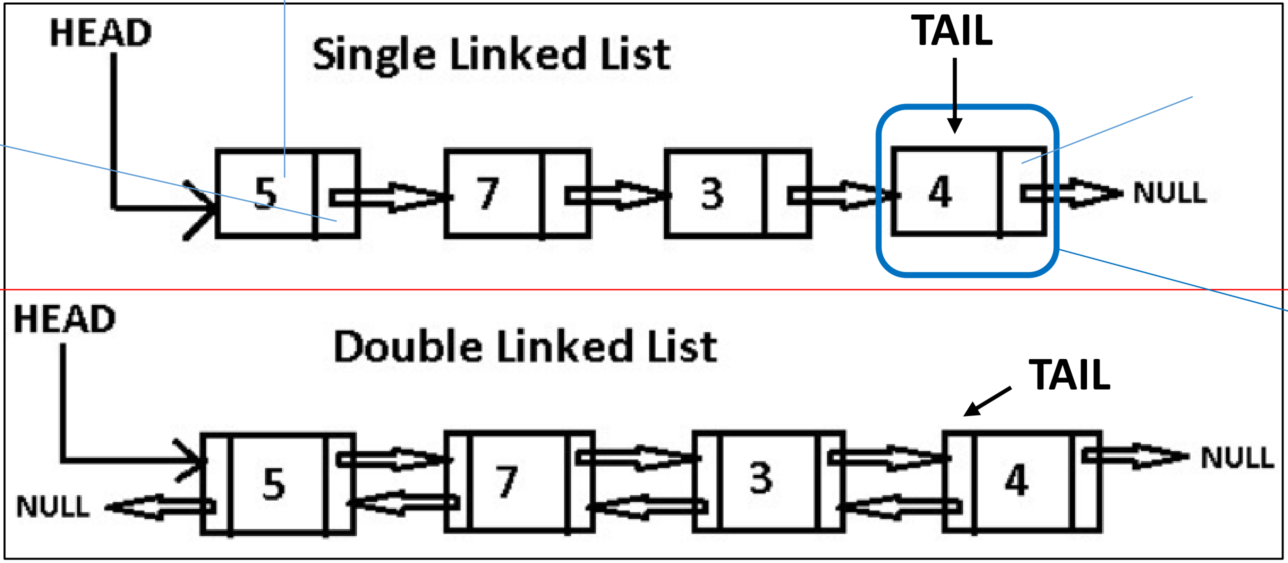
Linked List

- Dynamically shrink and grow
- Dynamic memory management
- No random access is allowed
- Slower access
  - Elements not in contiguous memory locations

What is Linked list?

❑ Type of Linked List

- There are two types of linked lists:
  - A single linked list is a linked list that has **a link to either its successor or predecessor**.
  - A double linked list is a linked list that has **both links** to successor and predecessor.



Data

A pointer points to the next element (successor)

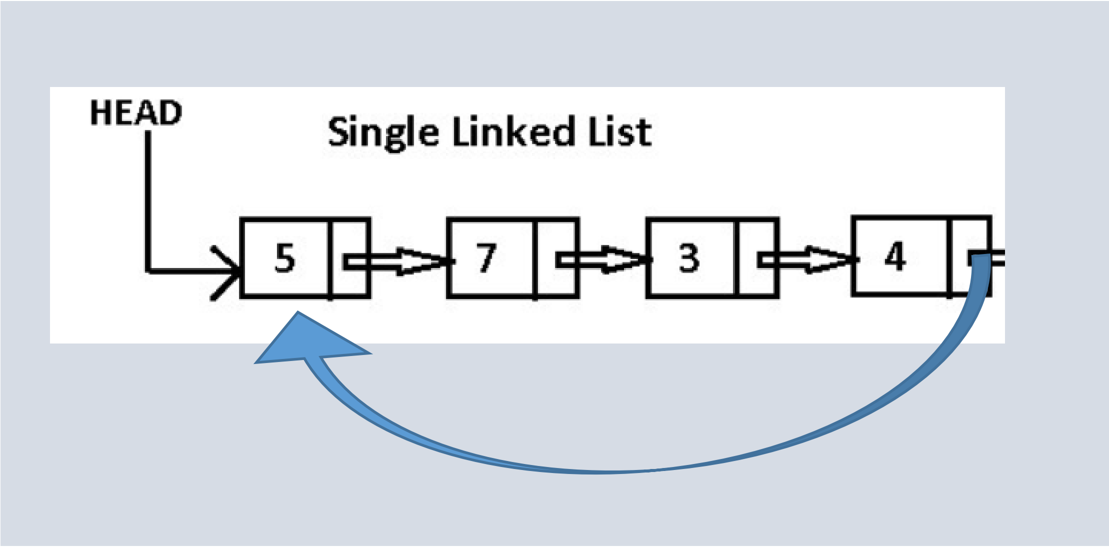
myList1

myList2

Last node of the list points to NULL

Remark

- A single or double linked list can be called **a circular linked list** when the last element (tail) points to the first element (head).



Circular linked list

List Operations

❑ Operations with a list

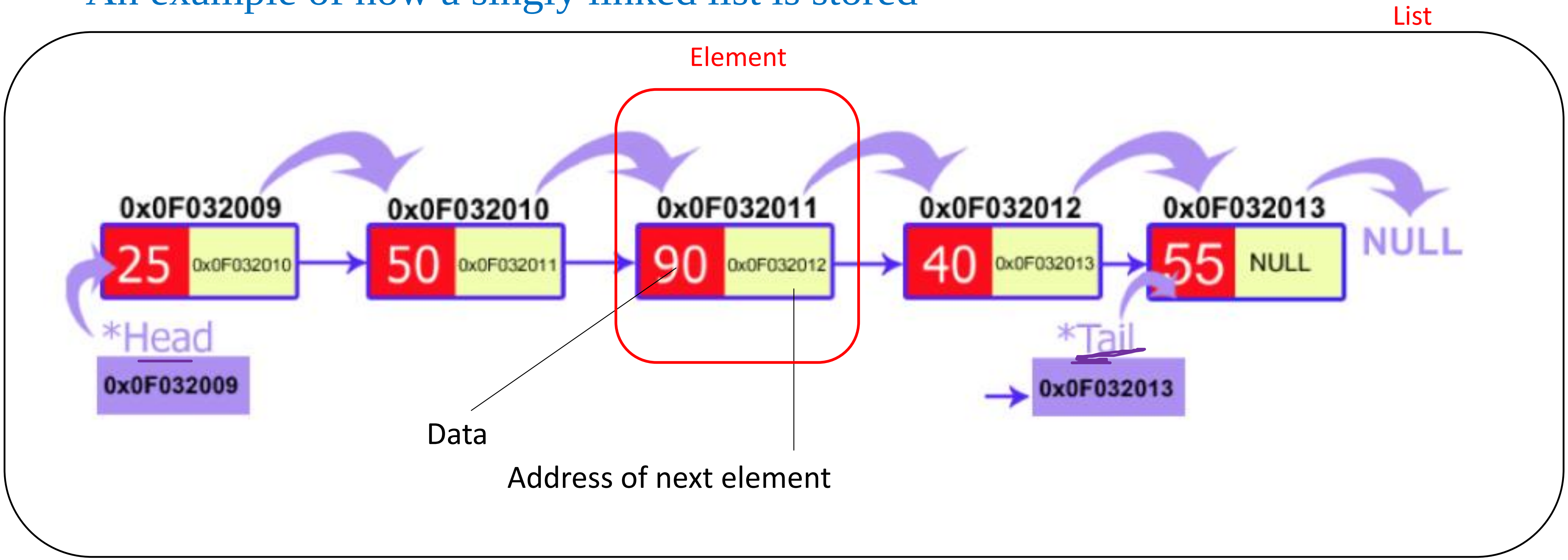
- ✓Creating a list
- ✓Insert a new element to a list
  - ✓ Insert to beginning, end, at a position
- ✓Delete an element from a list
  - ✓ Delete to beginning, end, at a position
- ✓Search an element in a list
- ✓Update an element in a list
- ✓Display data in list
- ✓Reverse a list
- ✓Combine two lists
- ✓… etc.

Singly Linked List (SLL)

Singly linked list

❑ Overview

- An example of how a singly linked list is stored



List operation

❑ Operation with a list

- All elements of a linked list can be accessed by
  - First setup a pointer pointing to the first element (node) of the list
  - Loop to traverse the list until NULL

- One of the disadvantage of the single linked list is
  - Given a pointer A to a node, we can not reach any of the nodes that precede the node (previous element) to which A is pointing

Operation on linked list

❑ Operations

- Important operation
  - Create a list
  - Insert element to the list
    - At the beginning
    - At the end
    - At the specific position
  - Delete the element
    - At the beginning
    - At the end
    - At the specific position
  - Destroy a list

```
Struct Element
            data: data_type
            *next: Element
End struct
```

```
Struct List
            *head: Element
            *tail: Element
            n: Integer
End struct
```

- **n** store number of elements in list.
- **n** is zero when list is first created. Then n is incremented by 1 when there is an element added to list.

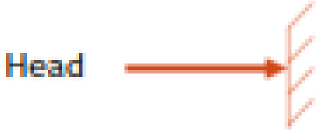
Examples

❑ Create an element

```
Var *head, *tmp : Element
```

- Create an empty list

```
head ← null
```

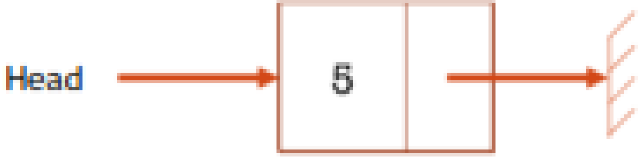


address

- Add an element of the list with value 5

```
tmp ← new(size(Element))
tmp→ data ← 5
tmp→ next ← null
head ← tmp
```

Reserve/allocate memory for this element

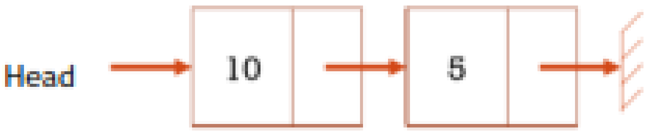


Examples

❑ Add and remove element

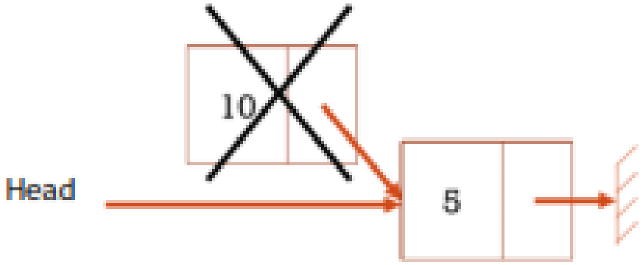
- Add a new element containing value 10 to the beginning of the list

```
tmp ← new(size(Element))
tmp→ data ← 10
tmp→ next ← head
head ← tmp
```



- Delete the first element from the list

```
tmp  ← head
head ← head → next
free(tmp)
```



Create a list

❑ A function to create an empty list

```
Function create_list( ) : Pointer of List
        var  *ls :  List

        ls ← new(size(List))
        ls→n  ←  0
        ls→head ← null
        ls→tail ← null

        return ls

End function
```

Steps to create an empty list:

1. Create a list variable
2. Allocate memory
3. Set 0 to n since we are creating an empty list
4. Head points to **null**
5. Tail points to **null**

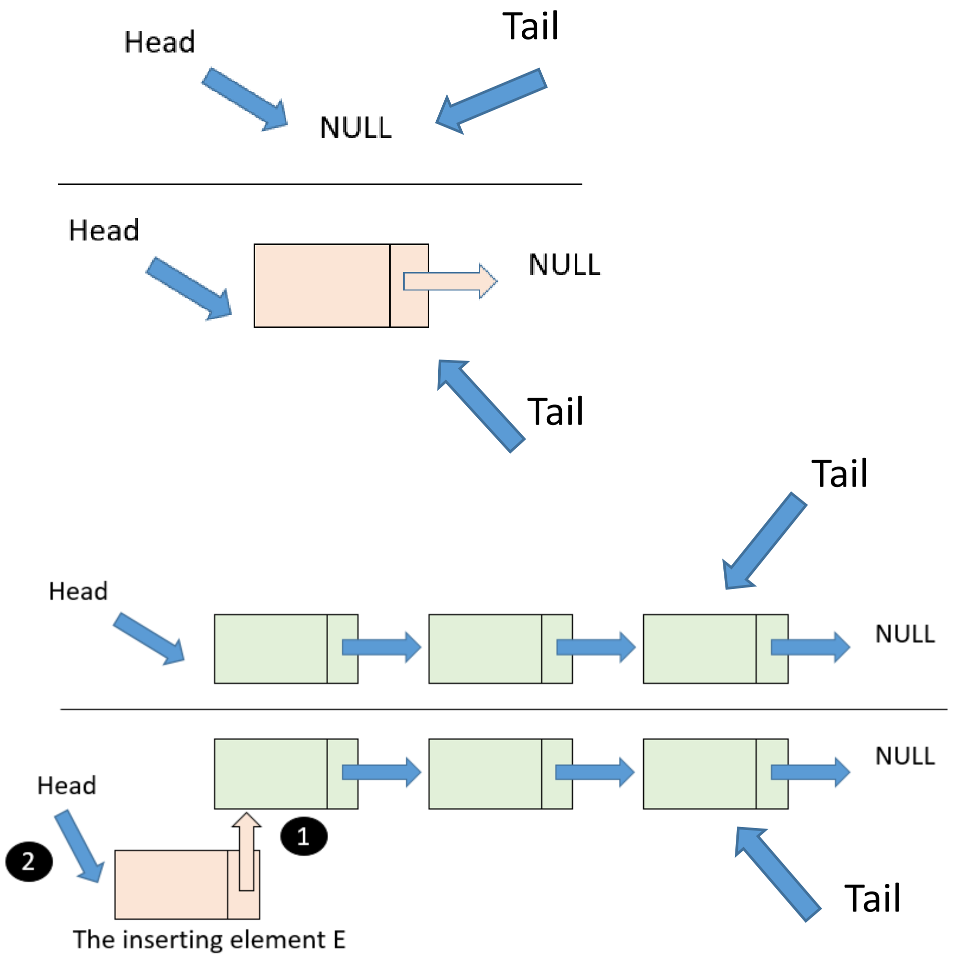
Insertion

❑ Insert an element to the beginning of the list

```
Procedure insert_be(*ls: List, d: data_type)
        var  *E: Element
   1    E ← new(size(Element))
        E→data ← d

   2    E→next ← ls→head
   3    ls→head ← E
        if(ls→n ==0) then
   4        ls→tail ← E
        end if
   5    ls→n ← ls→n + 1
End procedure
```

Steps to add element to beginning of list

1. Create a new element E
2. Make next pointer of E points to head of list
3. Update E to be head of list
4. Update tail if needed
5. Increase n by 1 (n is number of elements in list)



Display elements in list

```
Procedure void(*ls: List)
    var *tmp: Element
    tmp ← ls→head

    while(tmp!=NULL) do
        write(tmp→data)
        tmp ← tmp→next
    end while
End procedure
```

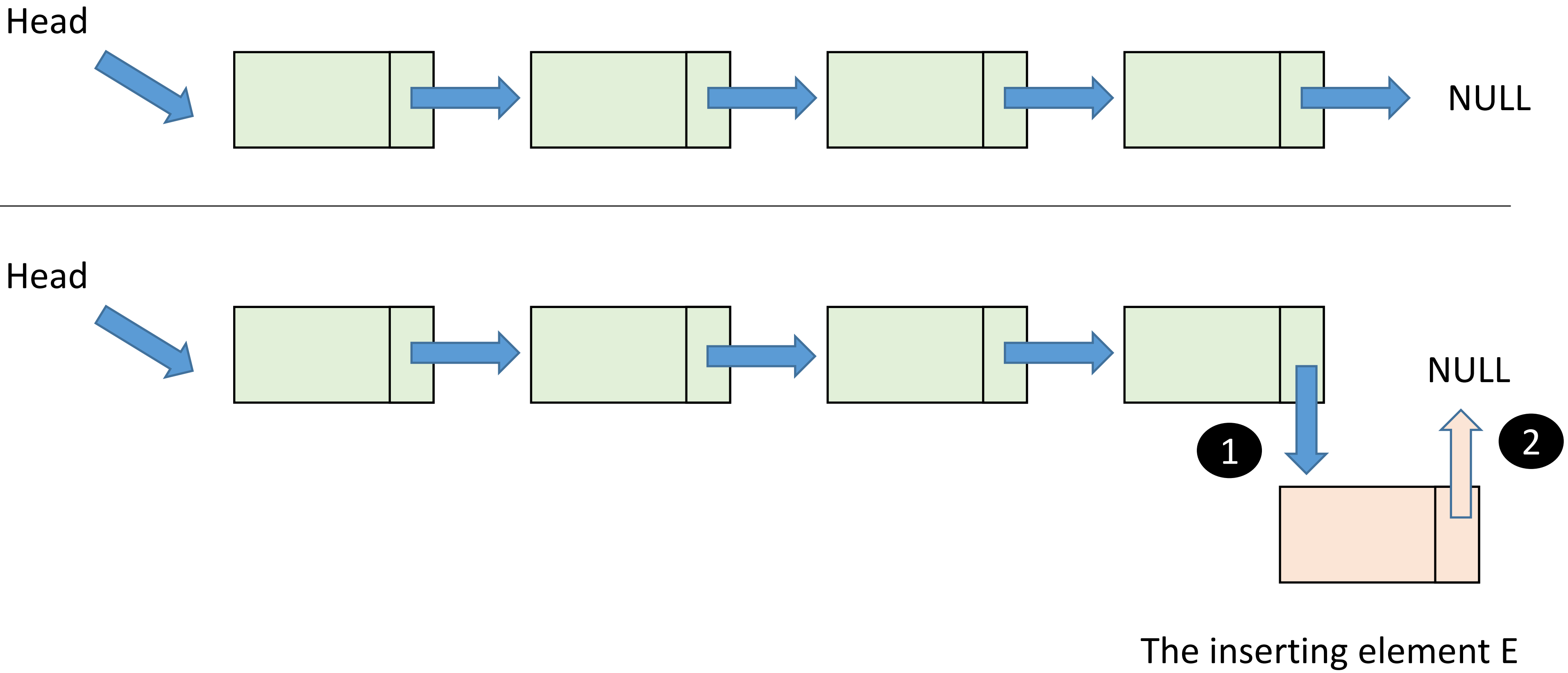
Steps to display element in list

1. Start from head
2. Move to each element each time
3. …
4. …

Insertion

❑ Insert an element to the end of the list

- Create an element E
- Simply make the last element (tail) points to E



Insert an element to the list

❑ Insert an element to end of the list

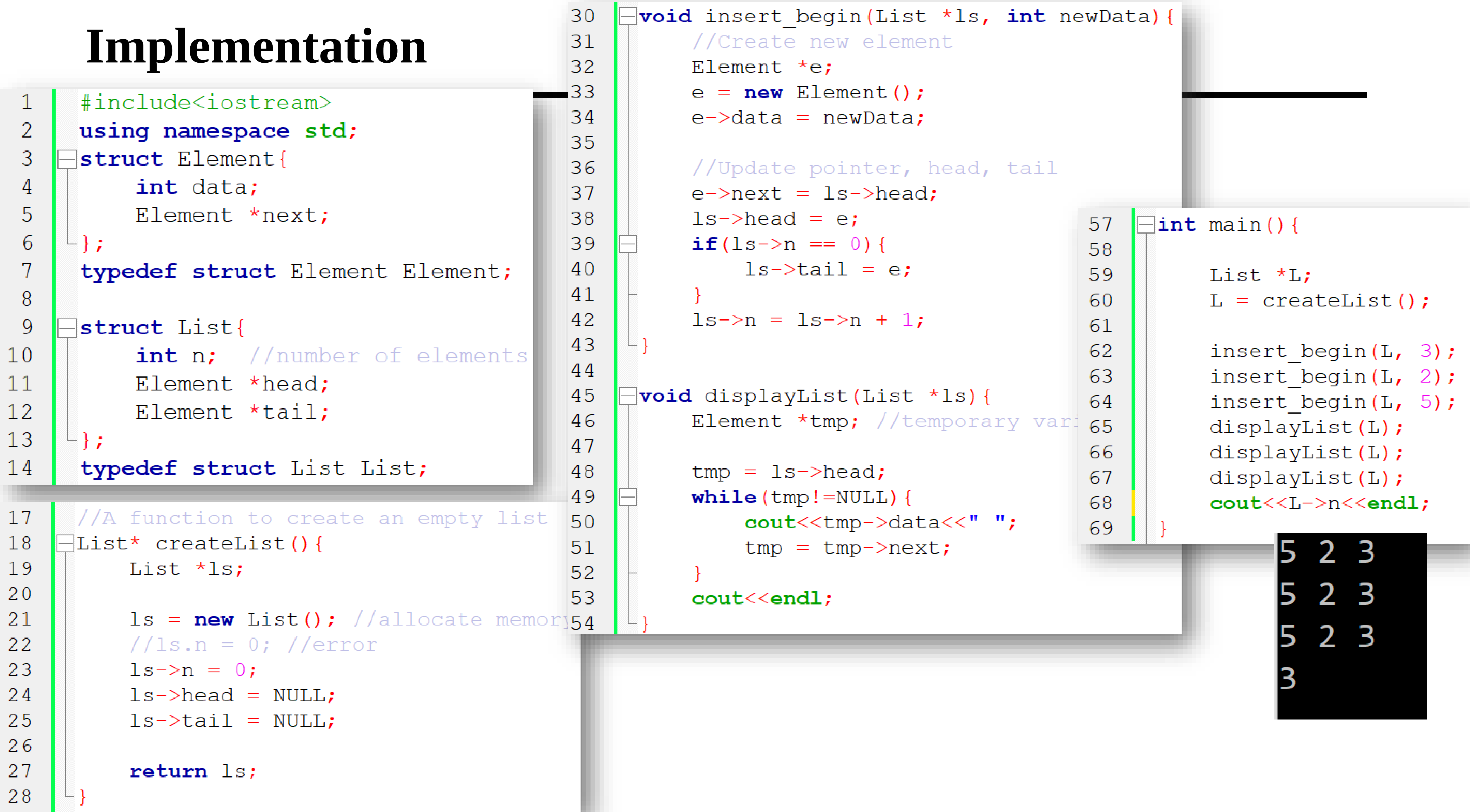
```
Procedure insert_end(*ls: List, d: data_type)
        var *E: Element
        if (ls→n == 0) then
                insert_be(ls, d)
        else
                E ← new(size(Element))
                E→data ← d
                E→next ← null

                ls→tail→next ← E
                ls→tail  ← E
                ls→n ← ls→n + 1
        end if
End procedure
```

Steps to add element to end of list

1. …
2. …
3. …
4. …

Q&A

Implementation



```cpp
#include<iostream>
using namespace std;
struct Element{
    int data;
    Element *next;
};
typedef struct Element Element;

struct List{
    int n;  //number of elements
    Element *head;
    Element *tail;
};
typedef struct List List;
```

```cpp
//A function to create an empty list
List* createList(){
    List *ls;

    ls = new List(); //allocate memory
    //ls.n = 0; //error
    ls->n = 0;
    ls->head = NULL;
    ls->tail = NULL;

    return ls;
}
```

```cpp
void insert_begin(List *ls, int newData){
    //Create new element
    Element *e;
    e = new Element();
    e->data = newData;

    //Update pointer, head, tail
    e->next = ls->head;
    ls->head = e;
    if(ls->n == 0){
        ls->tail = e;
    }
    ls->n = ls->n + 1;
}

void displayList(List *ls){
    Element *tmp; //temporary vari

    tmp = ls->head;
    while(tmp!=NULL){
        cout<<tmp->data<<" ";
        tmp = tmp->next;
    }
    cout<<endl;
}
```

```cpp
int main(){

    List *L;
    L = createList();

    insert_begin(L, 3);
    insert_begin(L, 2);
    insert_begin(L, 5);
    displayList(L);
    displayList(L);
    displayList(L);
    cout<<L->n<<endl;
}
```

```
5 2 3
5 2 3
5 2 3
3
```

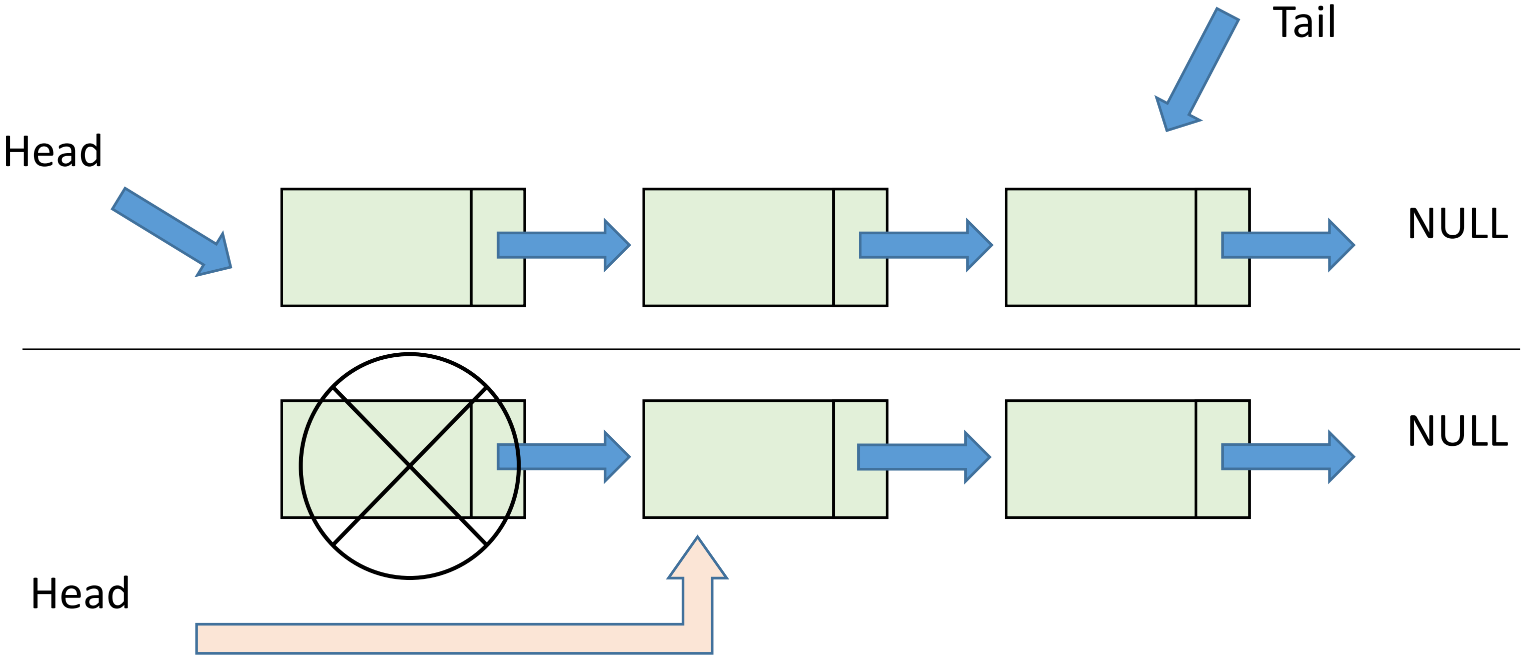
How to delete data from linked list

❖ Delete first element (delete from beginning)

Deletion

❑ Delete the first element

Before

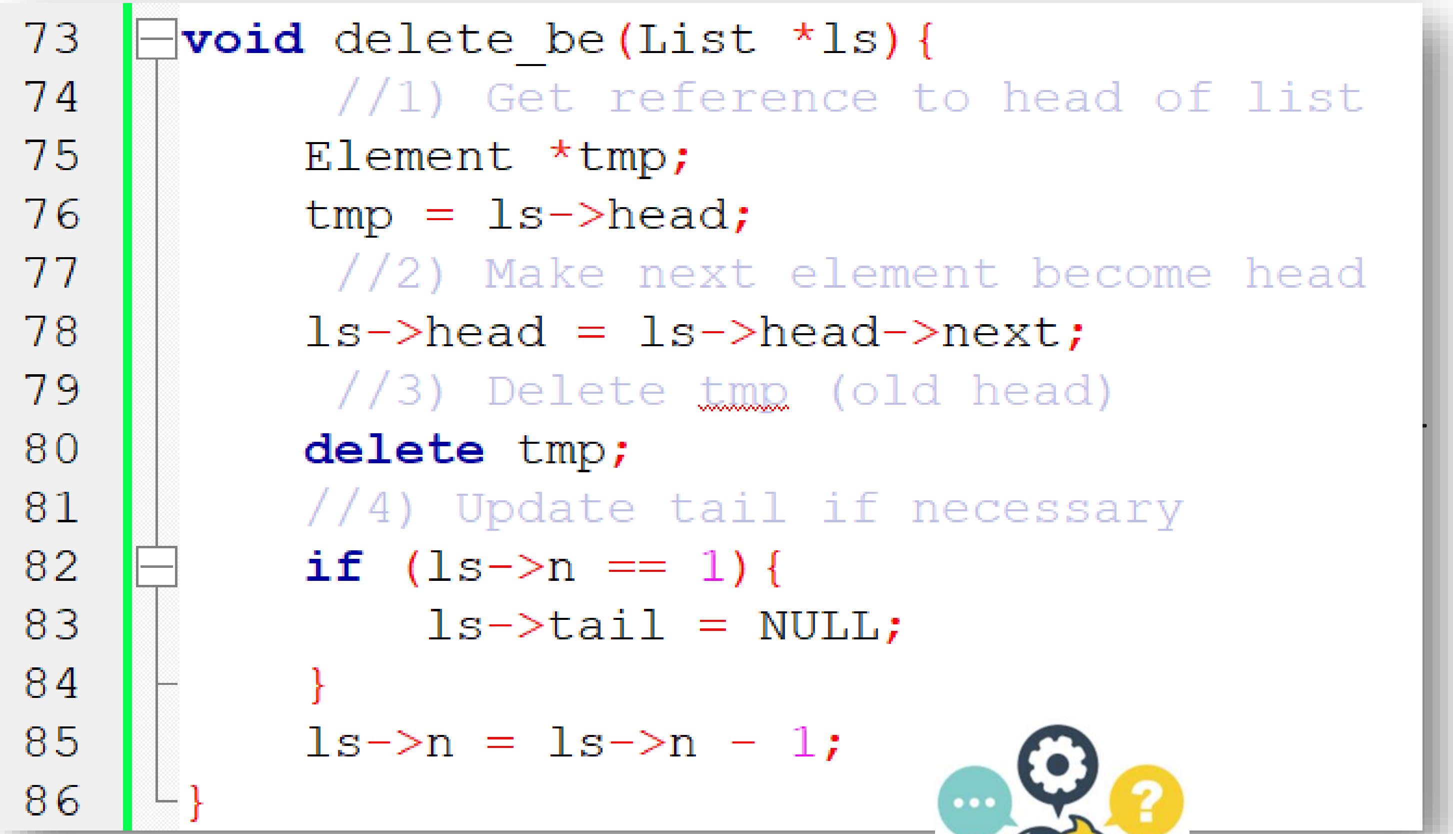


After

Delete the first element (delete beginning)

```
Procedure delete_be(*ls: List)
    //1) Get reference to head of list
    var *tmp: Element
    tmp ← ls→head
    //2) Make next element become head
    ls→head ← ls→head→next
    //3) Delete tmp (old head)
    free(tmp)
    //4) Update tail if necessary
    if (ls→n == 1) then
        ls→tail ← NULL
    end if
    ls→n ← ls→n - 1
End procedure
```

Delete first element
(delete beginning)



```
void delete_be(List *ls){
     //1) Get reference to head of list
    Element *tmp;
    tmp = ls->head;
     //2) Make next element become head
    ls->head = ls->head->next;
     //3) Delete tmp (old head)
    delete tmp;
    //4) Update tail if necessary
    if (ls->n == 1){
        ls->tail = NULL;
    }
    ls->n = ls->n - 1;
}
```

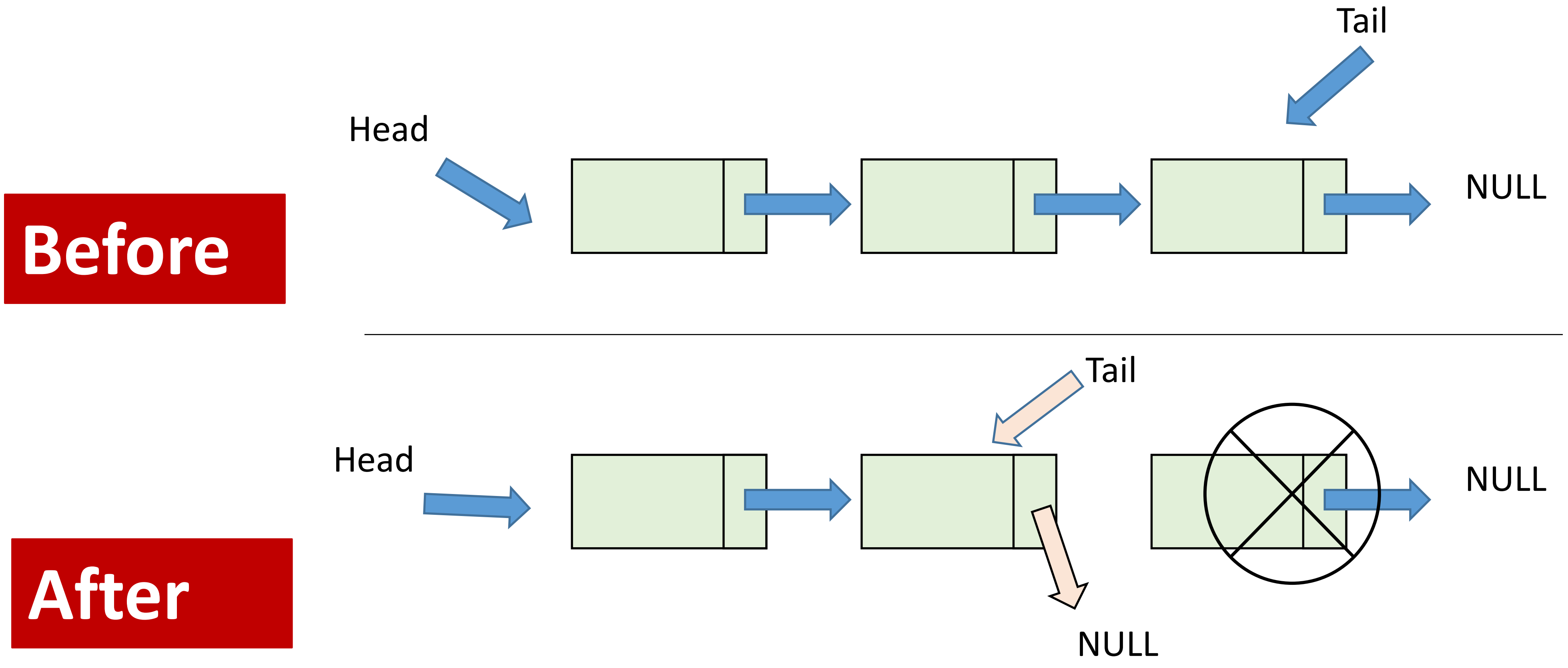
How it works … ?

How to delete data from linked list

- Delete last element (delete from end)

Deletion

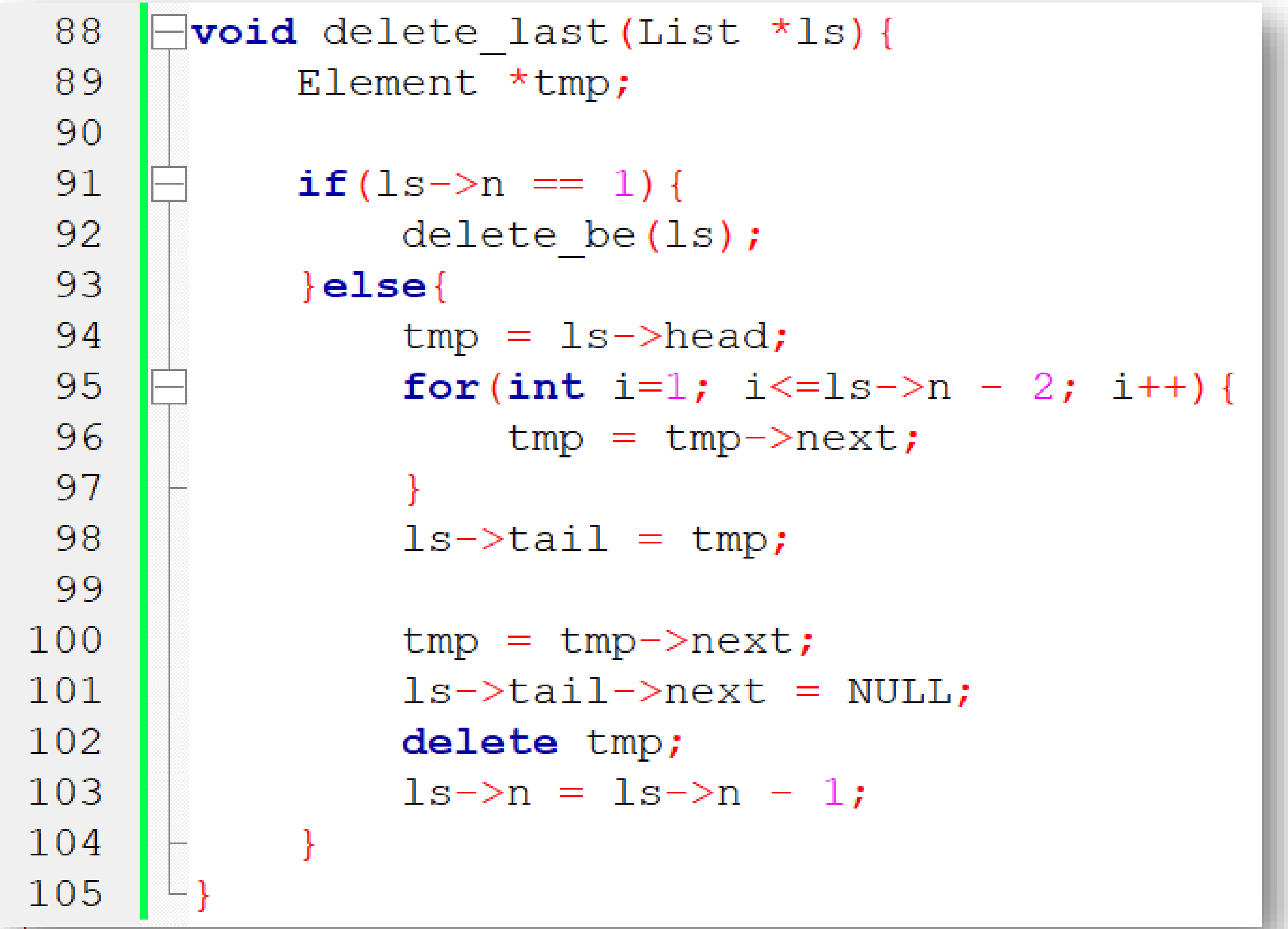
- ❑ Delete the last element from single linked list



Delete the last element

```
Procedure delete_last(*ls: List)
        var *tmp: Element
        var i: integer
        if(ls→n==1) then
                delete_be(li)
        else                                            1
                //Go to the 2nd last element
                tmp ← ls→head
                for(i←1;  <= ls→n  - 2; i++) do
                        tmp ← tmp→next
                end for
        2       //update tail and delete last old element
                ls→tail ← tmp
                tmp ← tmp→next          3
                ls→tail→next ← NULL     4
        5       free(tmp)
                ls→n ← ls→n - 1
        end
End procedures
```

Delete first element
(delete beginning)



```
void delete_last(List *ls){
    Element *tmp;

    if(ls->n == 1){
        delete_be(ls);
    }else{
        tmp = ls->head;
        for(int i=1; i<=ls->n - 2; i++){
            tmp = tmp->next;
        }
        ls->tail = tmp;

        tmp = tmp->next;
        ls->tail->next = NULL;
        delete tmp;
        ls->n = ls->n - 1;
    }
}
```

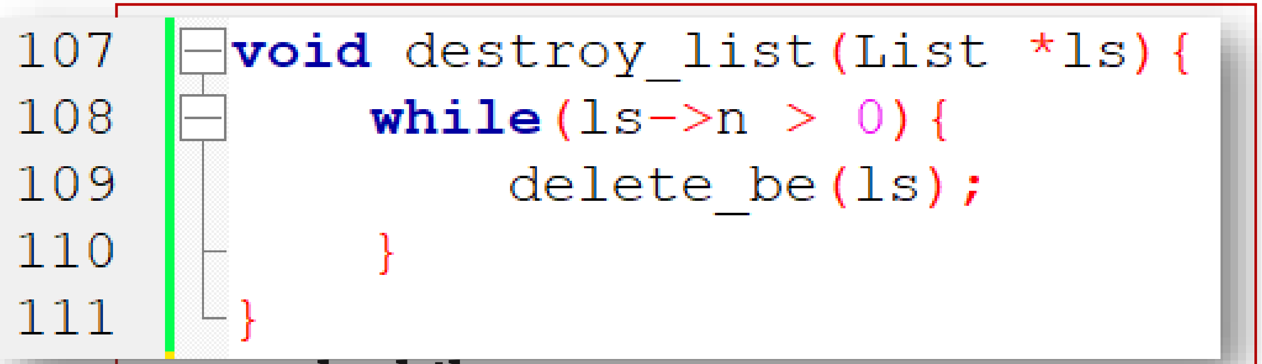
How it works ... ?

How to delete data from linked list

❖ Delete all data (destroy list)

Destroy a list

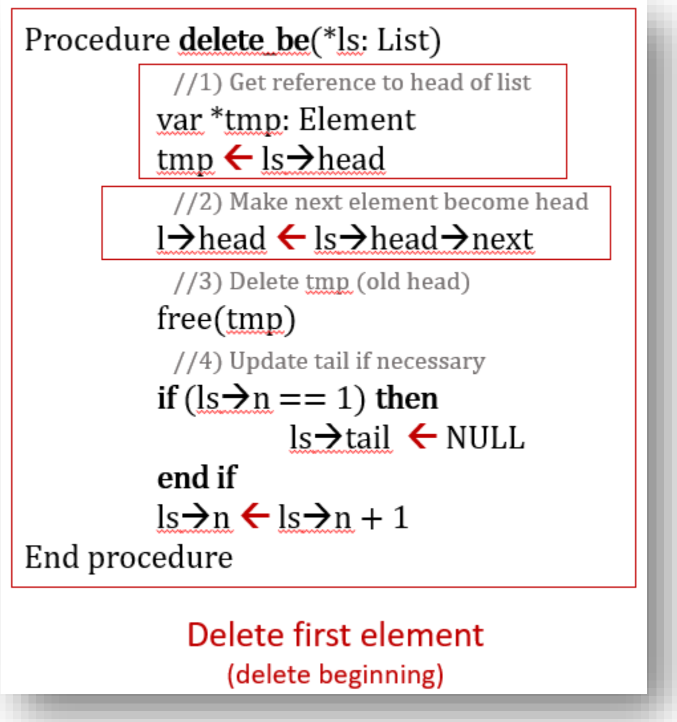
❑ Delete all data in list



```
void destroy_list(List *ls){
    while(ls->n > 0){
        delete_be(ls);
    }
}
```

end while

End procedure



```
Procedure delete_be(*ls: List)
        //1) Get reference to head of list
        var *tmp: Element
        tmp ← ls→head
        //2) Make next element become head
        l→head ← ls→head→next
        //3) Delete tmp (old head)
        free(tmp)
        //4) Update tail if necessary
        if (ls→n == 1) then
                ls→tail ← NULL
        end if
        ls→n ← ls→n + 1
End procedure
```

Delete first element
(delete beginning)

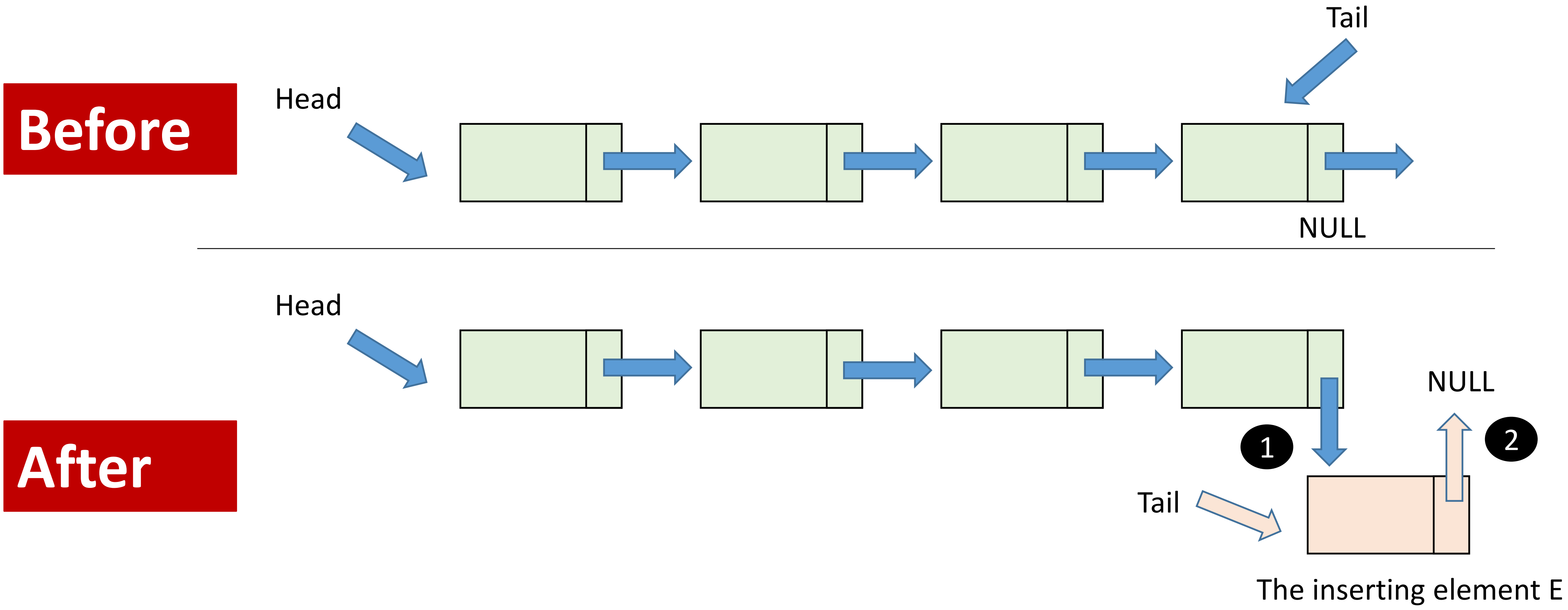
How it works ... ?

How to add data to linked list

- ❖ Add to end of list

Insertion

❑ Insert an element to the end of the list



Before

Head

Tail

NULL

After

Head

NULL

1

2

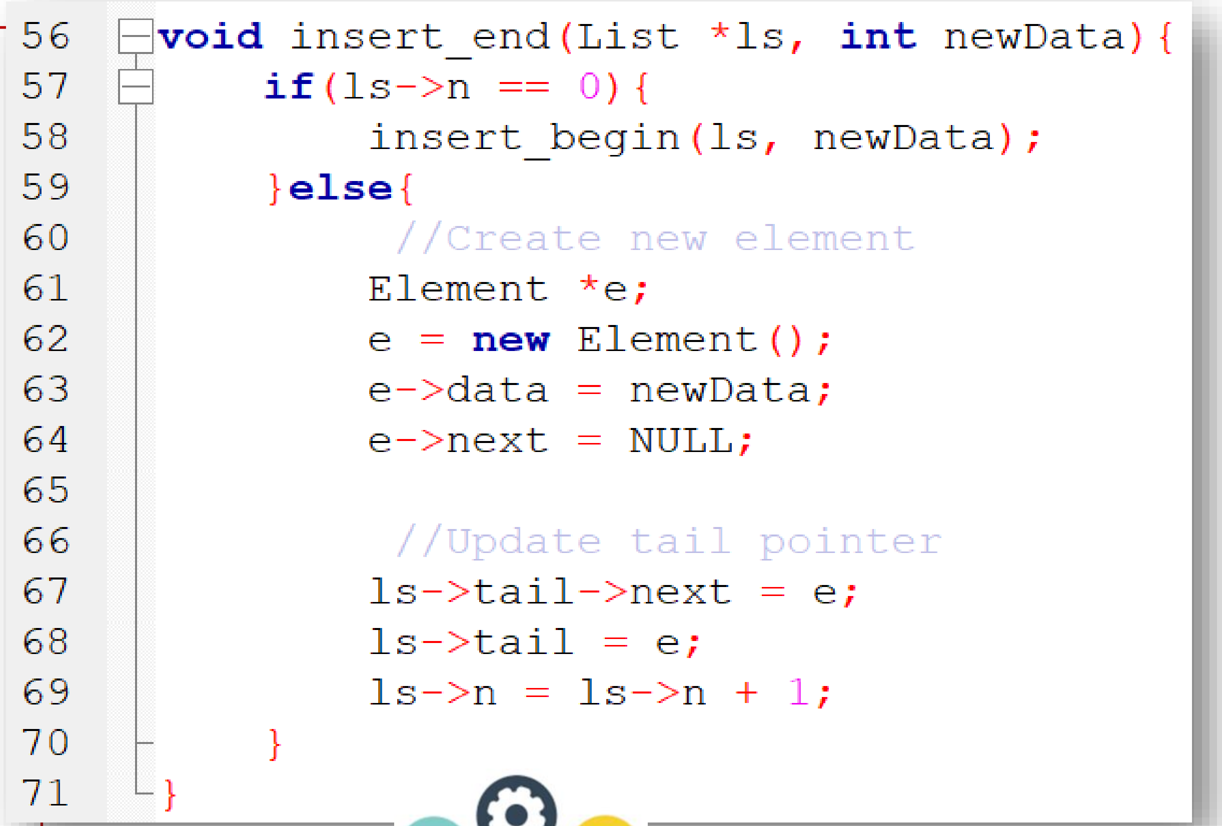
Tail

The inserting element E

Insert an element to the list

❑ Insert an element to end of the list

```
Procedure insert_end(*ls: List, d: data_type)
        var *E: Element
        if (ls→n == 0) then
                insert_be(ls, d)
        else
                E ← new(size(Element))
                E→data ← d
                E→next ← NULL

                ls→tail→next ← E
                ls→tail  ← E
                ls→n ← ls→n + 1
        end if
End procedure
```



```
void insert_end(List *ls, int newData){
    if(ls->n == 0){
        insert_begin(ls, newData);
    }else{
         //Create new element
        Element *e;
        e = new Element();
        e->data = newData;
        e->next = NULL;

         //Update tail pointer
        ls->tail->next = e;
        ls->tail = e;
        ls->n = ls->n + 1;
    }
}
```

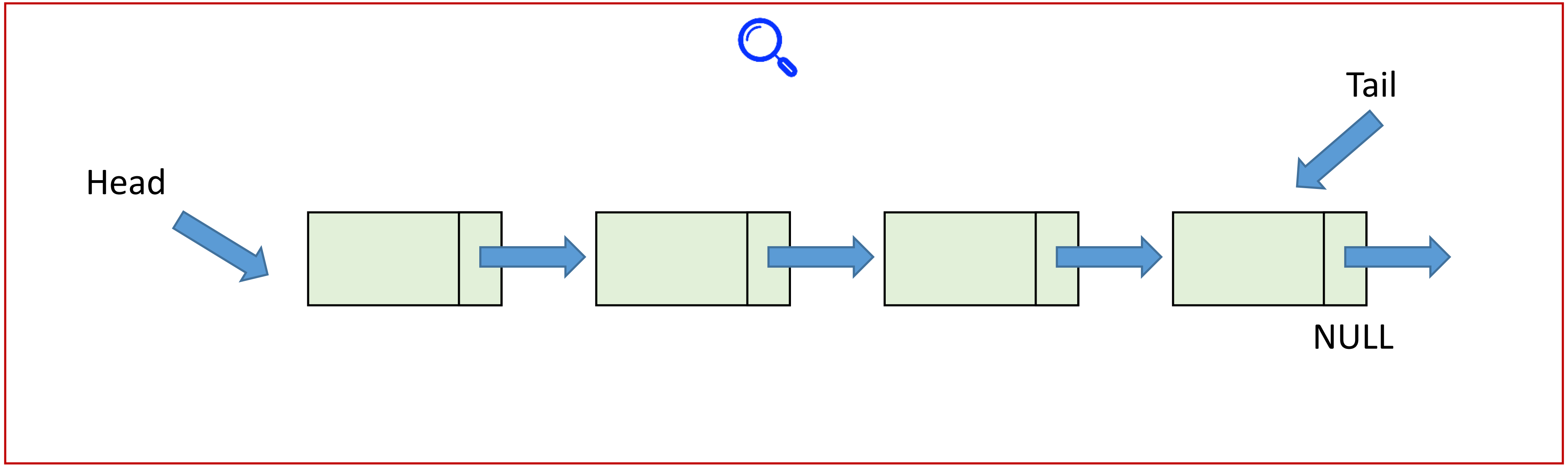
How it works ... ?

How to search data in linked list

- Searching for data

Search

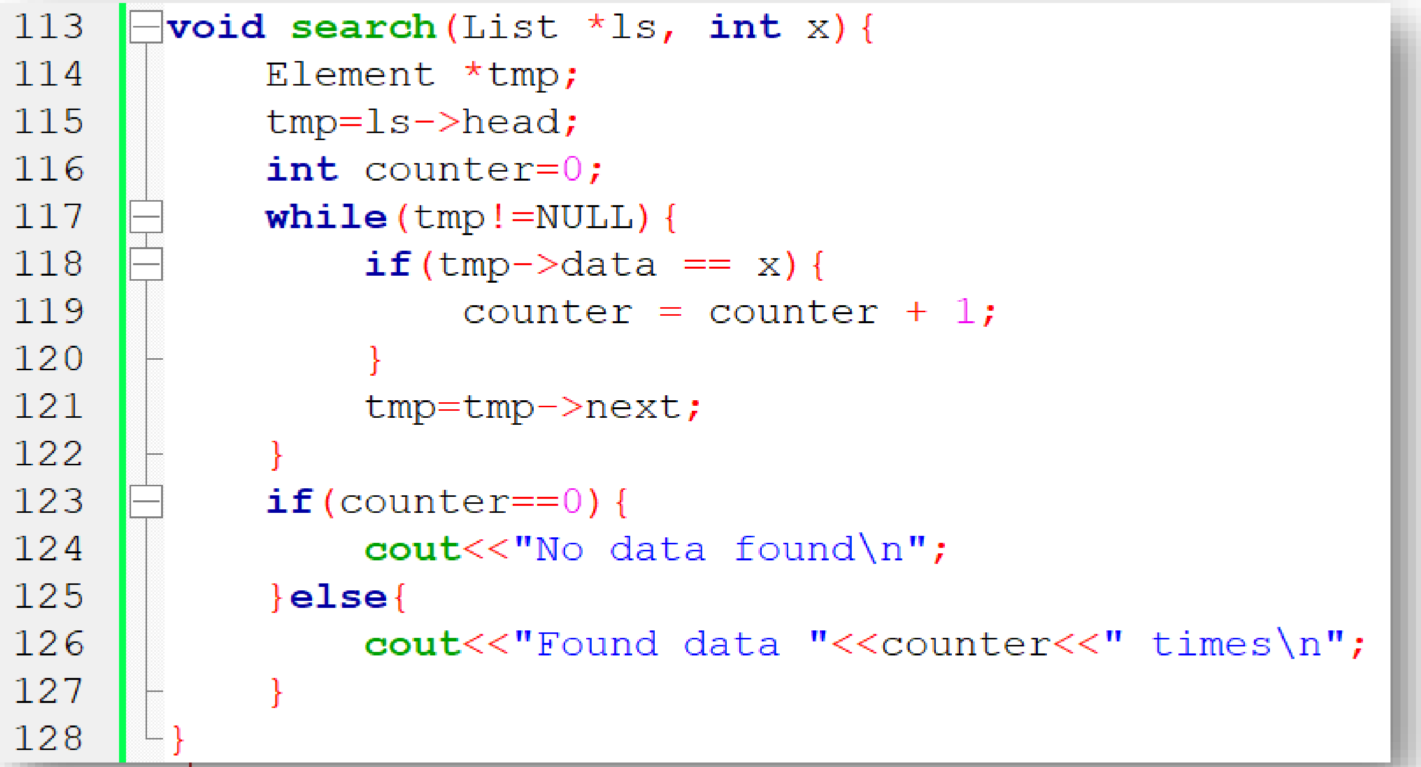
- ❑ Search for data in list



Search

❑ Search for data in list

```
Procedure search(*ls: List, d: data_type)
    var *tmp: Element
    var counter: integer
    counter ← 0

    tmp ← ls→head
    while(tmp != NULL) do
        if(tmp→data == d) then
            counter ← counter + 1
        end if
        tmp ← tmp→next
    end while

    if(counter==0) do
        write(“Not found”)
    else
        write(“Found ”, counter, “ times”)
    end if
End procedure
```



```
void search(List *ls, int x){
    Element *tmp;
    tmp=ls->head;
    int counter=0;
    while(tmp!=NULL){
        if(tmp->data == x){
            counter = counter + 1;
        }
        tmp=tmp->next;
    }
    if(counter==0){
        cout<<"No data found\n";
    }else{
        cout<<"Found data "<<counter<<" times\n";
    }
}
```

How it works … ?

Q&A

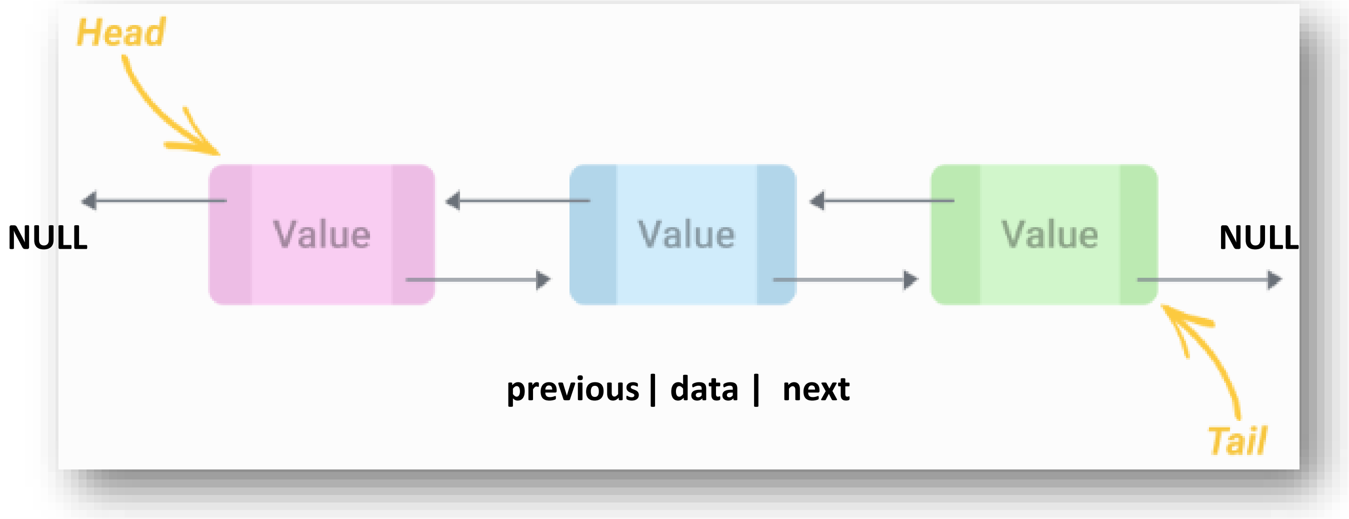
Search

❑ Search for some data

- Each element is traversed till the data in the element matched with the required value

```
void search(int x){
    Element *tmp;
    tmp=head;
    while(tmp!=NULL){
        if(tmp->data == x){
            cout<<“Found: ”<<tmp->data;
            break;
        }
        tmp=tmp->next;
    }
}
```

Double Linked List (DLL)



Double linked list

```
struct Element{
    int data;
    Element *next;
    Element *previous;
};
```

```
struct List{
    Element *head;
    Element *tail;
    int n;
};
```

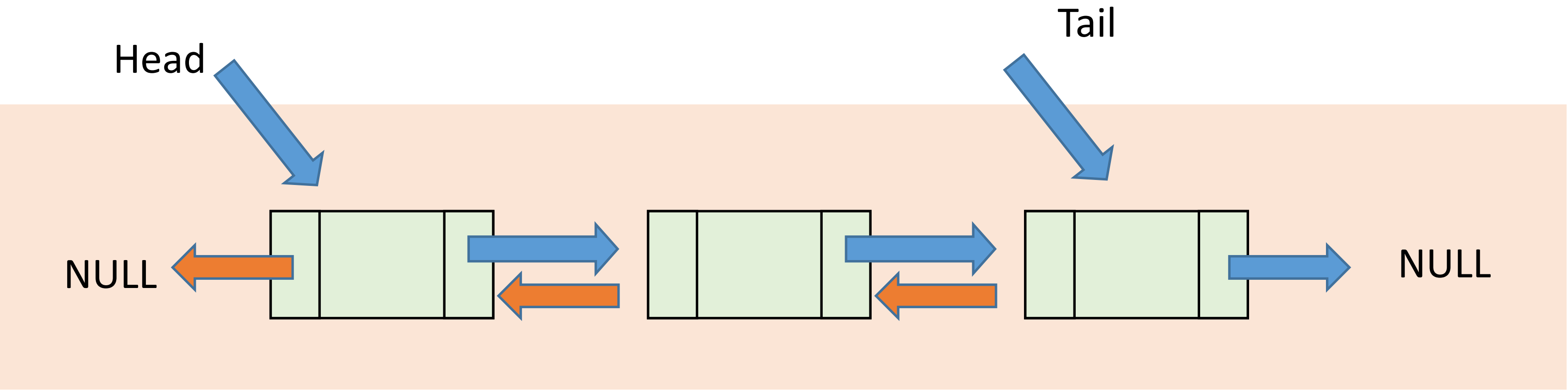
- Each element contains
  - ✓ Data
  - ✓ A link to its next element (successor)
  - ✓ A link to its previous element (predecessor)
    - They are created as a pointer

Element

Var data: integer
Var *next: Element
Var *previous: Element

List

Var *head: Element
Var *tail: Element
Var n: integer

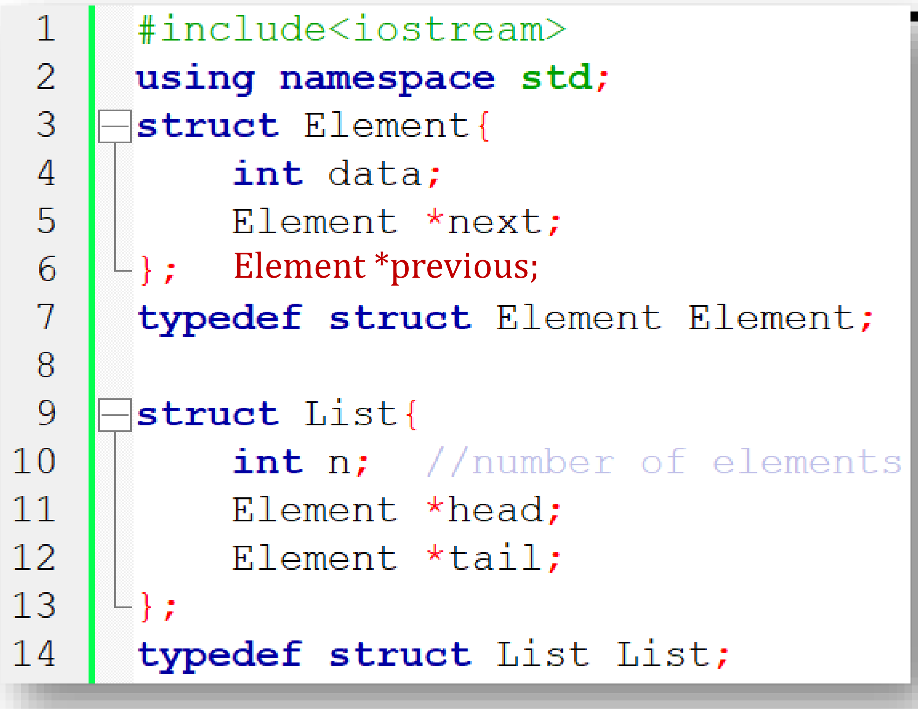


Implementation:

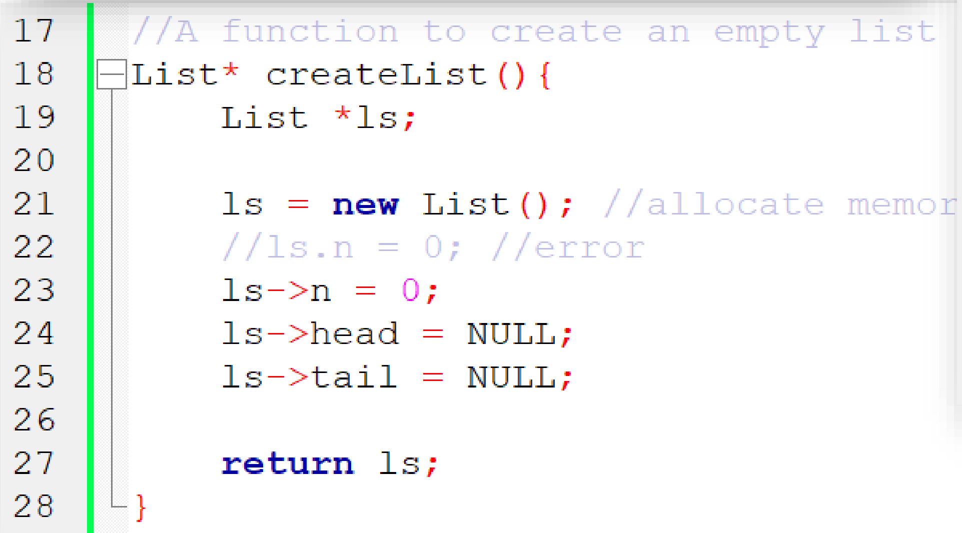
Double Linked List (DLL)

C++

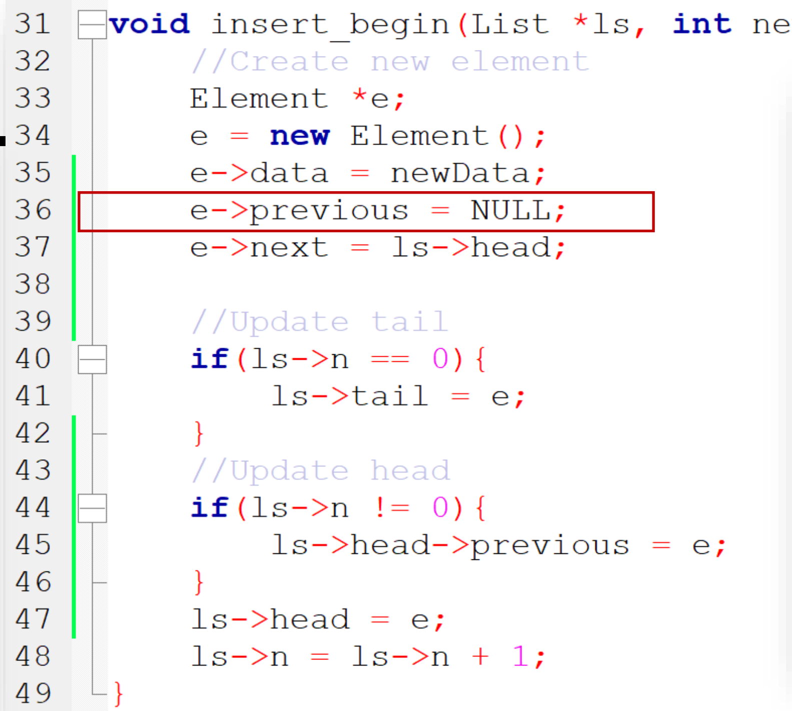
DLL



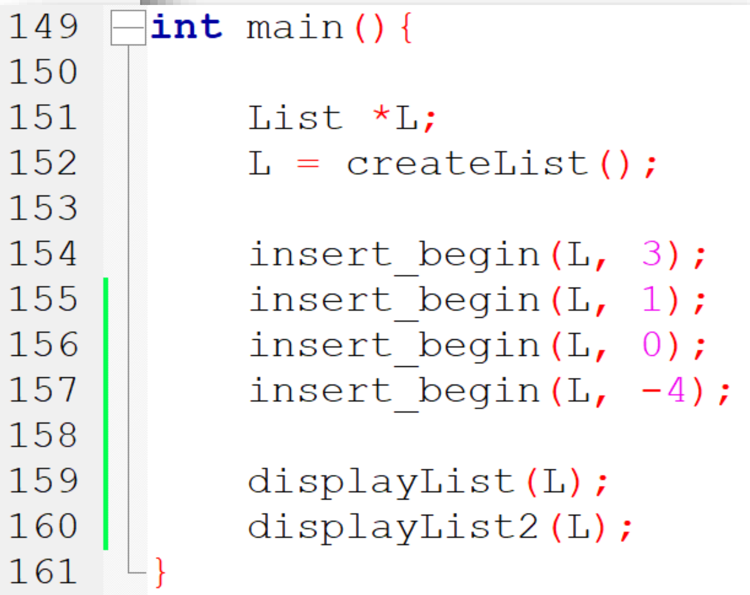
```cpp
#include<iostream>
using namespace std;
struct Element{
    int data;
    Element *next;
};  Element *previous;
typedef struct Element Element;

struct List{
    int n;  //number of elements
    Element *head;
    Element *tail;
};
typedef struct List List;
```



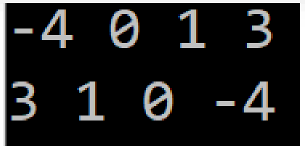
```cpp
//A function to create an empty list
List* createList(){
    List *ls;

    ls = new List(); //allocate memor
    //ls.n = 0; //error
    ls->n = 0;
    ls->head = NULL;
    ls->tail = NULL;

    return ls;
}
```



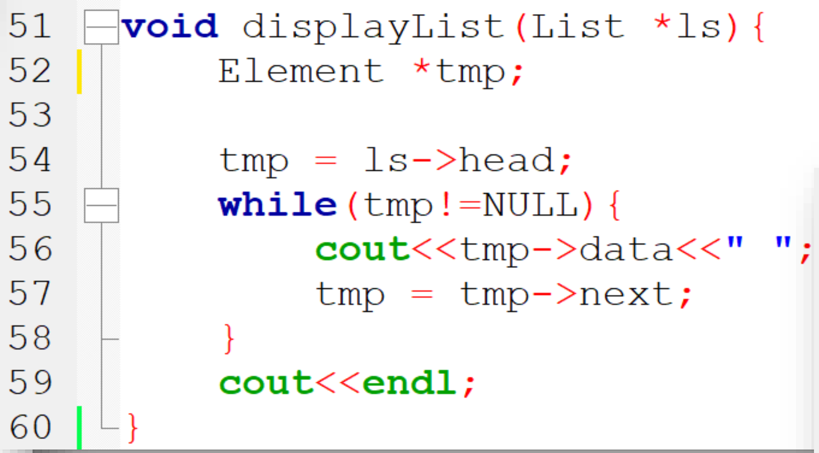
```cpp
void insert_begin(List *ls, int newData){
    //Create new element
    Element *e;
    e = new Element();
    e->data = newData;
    e->previous = NULL;
    e->next = ls->head;

    //Update tail
    if(ls->n == 0){
        ls->tail = e;
    }
    //Update head
    if(ls->n != 0){
        ls->head->previous = e;
    }
    ls->head = e;
    ls->n = ls->n + 1;
}
```



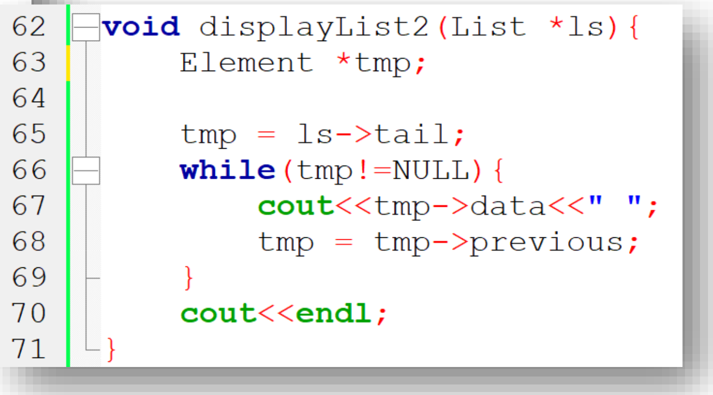
```cpp
int main(){

    List *L;
    L = createList();

    insert_begin(L, 3);
    insert_begin(L, 1);
    insert_begin(L, 0);
    insert_begin(L, -4);

    displayList(L);
    displayList2(L);
}
```



```
-4 0 1 3
3 1 0 -4
```



```cpp
void displayList(List *ls){
    Element *tmp;

    tmp = ls->head;
    while(tmp!=NULL){
        cout<<tmp->data<<" ";
        tmp = tmp->next;
    }
    cout<<endl;
}
```



```cpp
void displayList2(List *ls){
    Element *tmp;

    tmp = ls->tail;
    while(tmp!=NULL){
        cout<<tmp->data<<" ";
        tmp = tmp->previous;
    }
    cout<<endl;
}
```

Q and A

Doubly Linked List

❑ Creating a list

```
#include<iostream>
using namespace std;
```

1

```
struct Element{
    int data;
    Element *next;
    Element *previous;
};
```

2

```
struct List{
    Element  *head;
    Element *tail;
    int n;
};
```

1. Create data structure, 2. Create a list
3. Initialize an empty list

```
int main(){
```

3

```
    List *l;
    l = new List;
    l->n = 0;
    Element *e1,*e2,*e3;
```

4

```
    e1=new Element;
    e1->data = 1;
    e1->next = NULL;
    e1->previous = NULL;
    l->head = e1;
    l->tail = e1;
    l->n = l->n + 1;
```

5

```
    e2=new Element;
    e2->data = 2;
    e2->next = NULL;
    l->tail->next = e2;
    e2->previous = l->tail;
    l->tail = e2;
    l->n = l->n + 1;
```

4., 5., and 6. Add elements e1, e2, e3 to the end of the list

6

```
    e3=new Element;
    e3->data = 3;
    e3->next = NULL;
    l->tail->next = e3;
    e3->previous = l->tail;
    l->tail = e3;
    l->n = l->n + 1;
```

7

```
    Element *t;
    t = l->head;
    while(t != NULL){
        cout<<t->data<<" ";
        t = t->next;
    }
```

8

```
    Element *t2;
    t2 = l->tail;
    while(t2 != NULL){
        cout<<t2->data<<" ";
        t2 = t2->previous;
    }
}
```

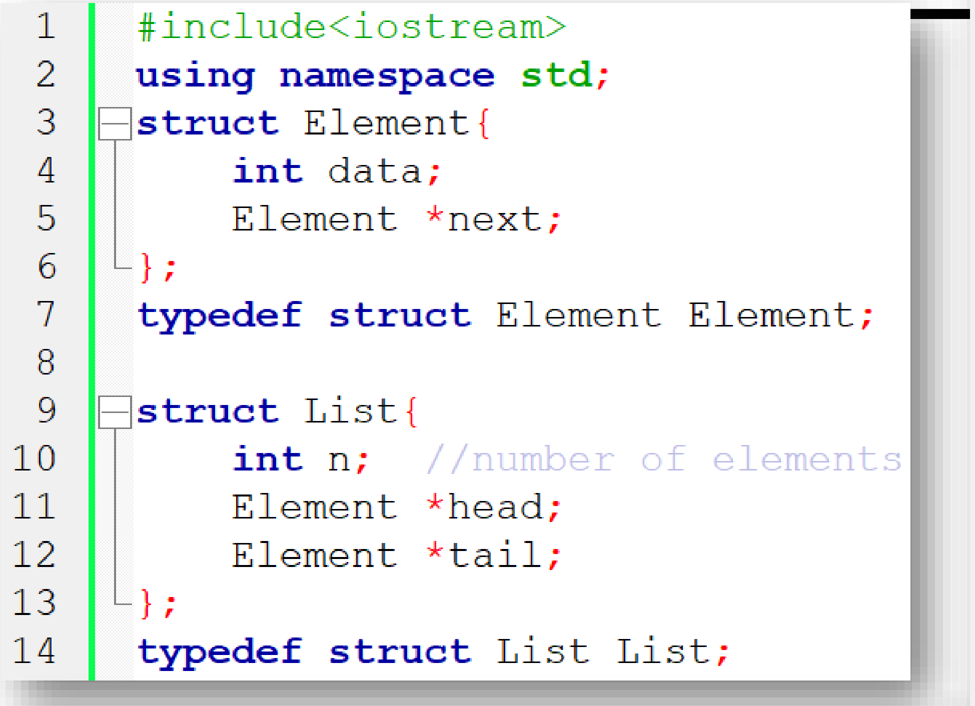
7. Display data in list from head (forward)
8. Display data in list from tail (backward)

Practice on Implementation of Linked List using C++

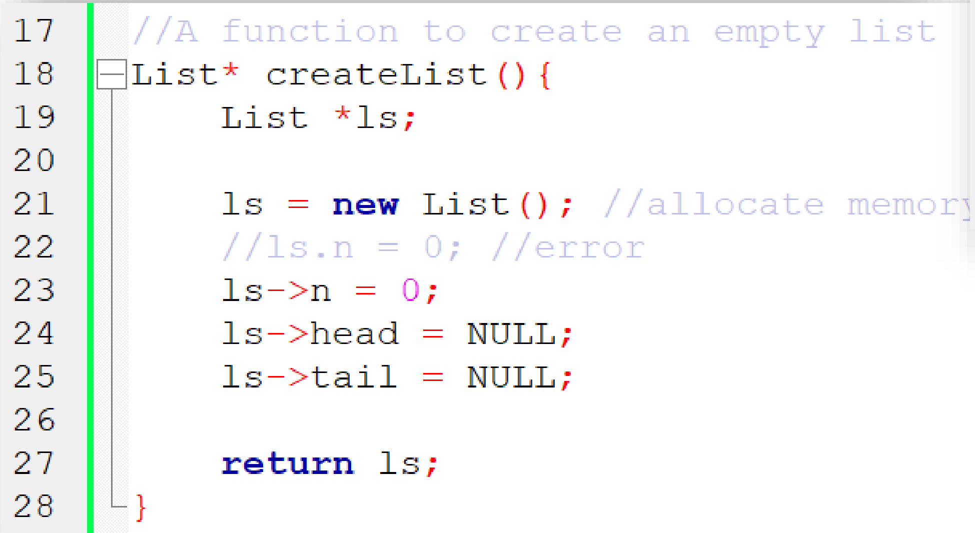
❑ Exercises

1. Create a singly linked list that can store integer numbers. Then perform these operations below
   a. Create two data structure (Element and List) then create an empty singly linked list
   b. Add the number 7 to the end of the list
   c. Add the number 4 to the end of the list
   d. Add 1 to the beginning of the list
   e. Remove the first element of the list
   f. Add 0 the beginning of the list
   g. Display all numbers in the list
2. Create a doubly linked list that stores the English alphabet (A-Z). Then display the list.

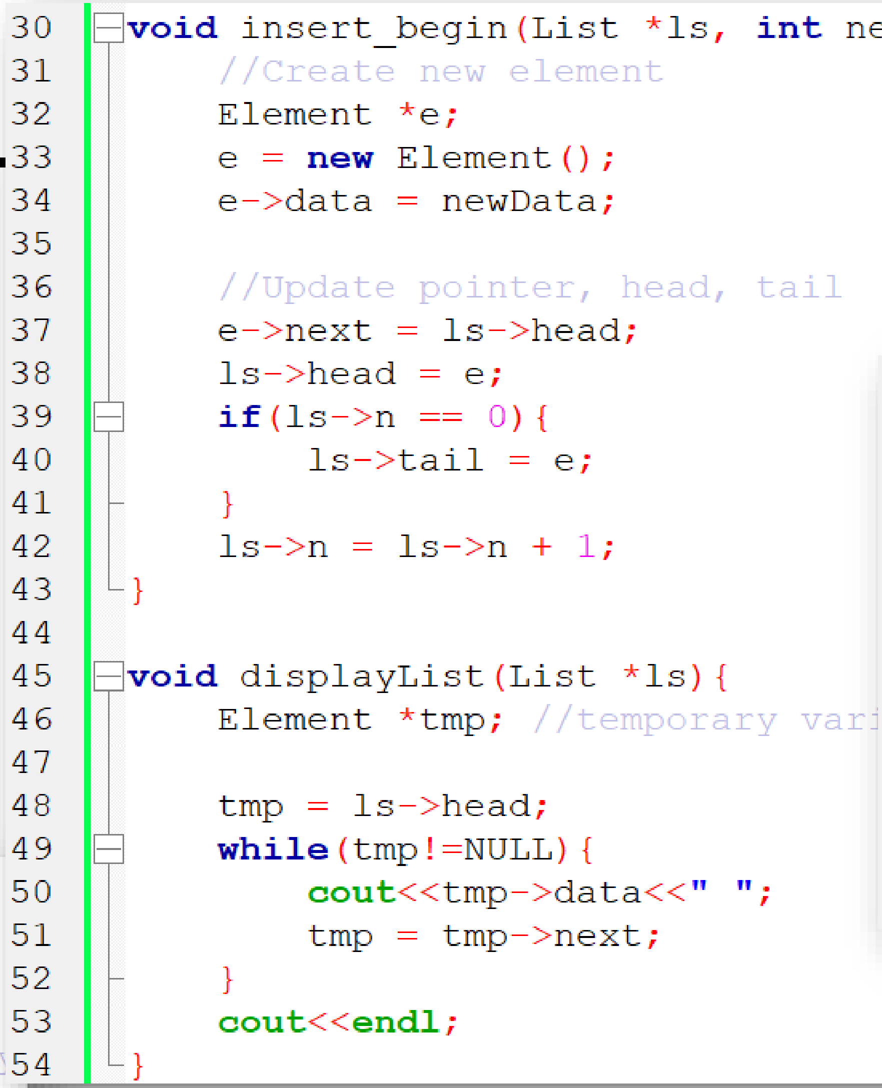
SLL recall



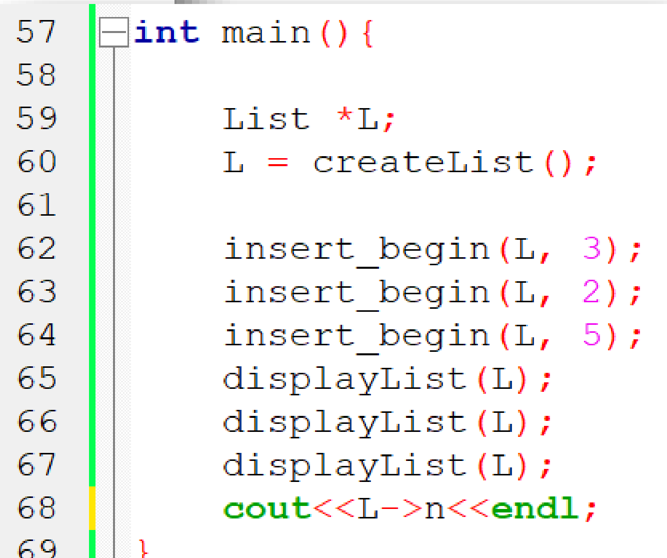
```cpp
#include<iostream>
using namespace std;
struct Element{
    int data;
    Element *next;
};
typedef struct Element Element;

struct List{
    int n;  //number of elements
    Element *head;
    Element *tail;
};
typedef struct List List;
```



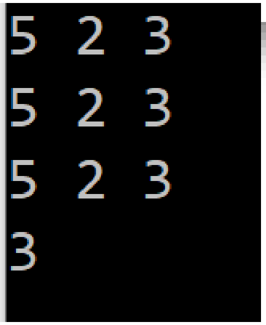
```cpp
//A function to create an empty list
List* createList(){
    List *ls;

    ls = new List(); //allocate memory
    //ls.n = 0; //error
    ls->n = 0;
    ls->head = NULL;
    ls->tail = NULL;

    return ls;
}
```



```cpp
void insert_begin(List *ls, int newData){
    //Create new element
    Element *e;
    e = new Element();
    e->data = newData;

    //Update pointer, head, tail
    e->next = ls->head;
    ls->head = e;
    if(ls->n == 0){
        ls->tail = e;
    }
    ls->n = ls->n + 1;
}

void displayList(List *ls){
    Element *tmp; //temporary vari

    tmp = ls->head;
    while(tmp!=NULL){
        cout<<tmp->data<<" ";
        tmp = tmp->next;
    }
    cout<<endl;
}
```



```cpp
int main(){

    List *L;
    L = createList();

    insert_begin(L, 3);
    insert_begin(L, 2);
    insert_begin(L, 5);
    displayList(L);
    displayList(L);
    displayList(L);
    cout<<L->n<<endl;
}
```



```
5 2 3
5 2 3
5 2 3
3
```

Q and A

Practice on Implementation of Linked List using C++

❑ Write a C++ program to

1. Create a singly linked list that can store integer numbers. Then
   a. Add the number 7 to the end of the list
   b. Add the number 4 to the end of the list
   c. Add the number 4 to the end of the list
   d. Display the list
   e. Delete the first element
   f. Display the list
2. Create a singly linked list that can store names of students. Then
   a. Add a student named “Jack” to the beginning of the list
   b. Add a student named “Rose” to the beginning of the list
   c. Add a student named “Sok” to end of the list
   d. Display all students’ names in the list
   e. Delete the last element
   f. Display all students’ names in the list
3. Create a singly linked list that stores the English alphabet (A-Z). Then display the list.

Practice on Implementation of Linked List using C++

❑ Write a C++ program to

4. Create a singly linked list that can store integer numbers. Then initialize the list with these values [1, 2, 3, 5, 0, -1]. Write a program to

    a. Create a function to add a number entered by a user to the end of the list

    b. Create a function to add a number entered by a user to the beginning of the list

    c. Create a function to delete the number at the beginning of the list

    d. Create a function to delete the number at the beginning of the list

    e. Create a function to check whether the list contains a number. The function return *true* when the list contains that number. Return *false* otherwise.

    Do ex1 again using above functions.

Practice on Implementation of Linked List using C++

❑ Write a C++ program to

5. Get a positive integer n input by a user. Then read n numbers from the input and build a singly linked list such that the first input number is the 1st element of the list, the second input number is the 2nd element of the list, so on and so forth.

6. Create a singly linked list for storing information of students. Each student has id, name, sex and average score. Then write a program to

   a. Create a function to ask for information (id, name, sex, average score) of a student then add her/him to the end of the list. The program does not add the student to the list when the student with the same id is already exist in the list.

   b. Create a function to display information of all students in the list

Assignment

-Teamwork

-How to score: individual and team score

-Substitute midterm exam

-Assigned date: 10th May 2019

-Deadline: 24th May 2019 (2 weeks)

-Present your work: 29th May 2019

Practice on Implementation of Linked List using C++

❑ Write a C++ program to

1. Create a singly linked list that can store integer numbers. Then
   a. Add the number 7 to the end of the list
   b. Add the number 4 to the end of the list
   c. Add the number 4 to the end of the list
   d. Display the list
   e. Delete the first element
   f. Display the list
2. Create a singly linked list that can store names of students. Then
   a. Add a student named “Jack” to the beginning of the list
   b. Add a student named “Rose” to the beginning of the list
   c. Add a student named “Sok” to end of the list
   d. Display all students’ names in the list
   e. Delete the last element
   f. Display all students’ names in the list
3. Create a singly linked list that stores the English alphabet (A-Z). Then display the list.

Practice on Implementation of Linked List using C++

❑ Write a C++ program to

4. Create a singly linked list that can store integer numbers. Then initialize the list with these values [1, 2, 3, 5, 0, -1]. Write a program to

   a. Create a function to add a number entered by a user to the end of the list

   b. Create a function to add a number entered by a user to the beginning of the list

   c. Create a function to delete the number at the beginning of the list

   d. Create a function to delete the number at the beginning of the list

   e. Create a function to check whether the list contains a number. The function return *true* when the list contains that number. Return *false* otherwise.

   Do ex1 again using above functions.

Practice on Implementation of Linked List using C++

❑ Write a C++ program to

5. Get a positive integer n input by a user. Then read n numbers from the input and build a singly linked list such that the first input number is the 1st element of the list, the second input number is the 2nd element of the list, so on and so forth.

6. Create a singly linked list for storing information of students. Each student has id, name, sex and average score. Then write a program to

   a. Create a function to ask for information (id, name, sex, average score) of a student then add her/him to the end of the list. The program does not add the student to the list when the student with the same id is already exist in the list.

   b. Create a function to display information of all students in the list

Practice on Implementation of Linked List using C++

❑ Write a C++ program to

7. Do ex1-6 with doubly linked list instead.

Q and A