LINKED LIST

PROGRAMMING TECHNIQUES

ADVISOR: Trương Toàn Thịnh

CONTENTS

- Introduction
- Linked list & array of data
- Single linked list
- Some operations
- Stack & queue

INTRODUCTION

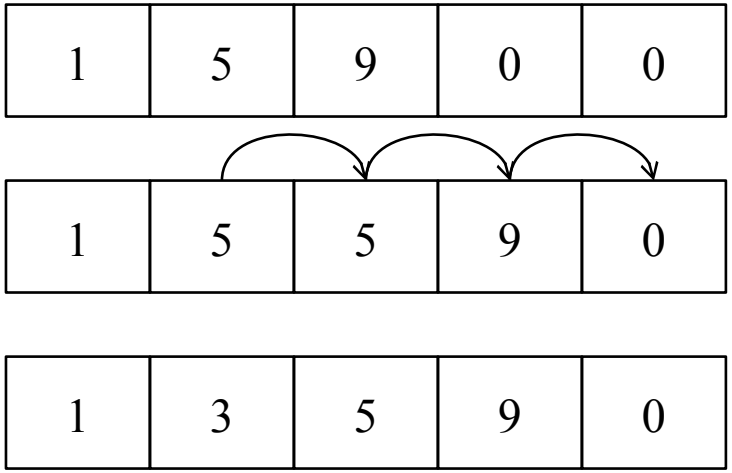
- A structure containing data member and an address of the next node
- A node of a linked list includes 2 components:
  - Data: contains information needed to store
  - Connection: an address of the next node

LINKED LIST & ARRAY OF DATA

- Example of 1D static array

```
int a[5] = {1, 5, 9}
```

- Drawbacks:
  - Need a prior size
  - Editing the array is complex



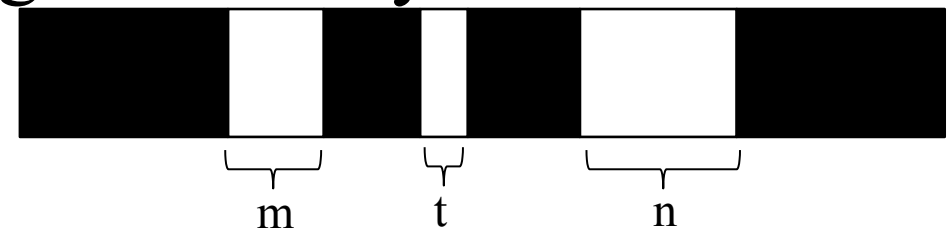
- Advantage: quickly random access

LINKED LIST & ARRAY OF DATA

- Example of 1D dynamic array

```
char* a = (char*)malloc(sizeof(char) * n)
char* a = new char[n];
```

- Advantage: efficient memory
- Drawback: The fragmentation causes fake lacking of memory

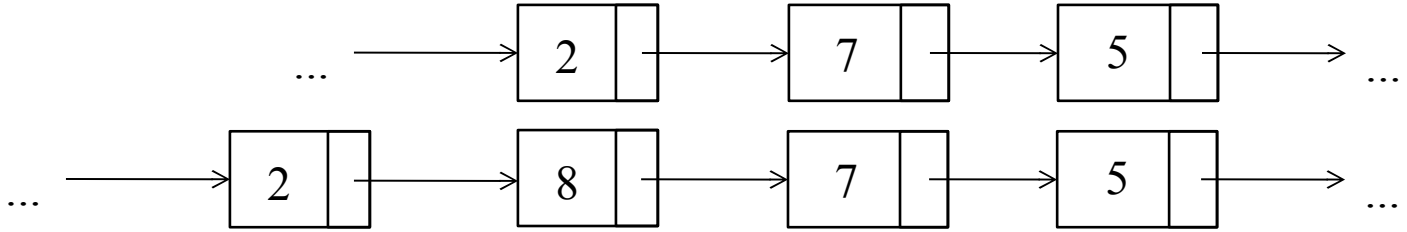



```
char* a = (char*)malloc(sizeof(char)*(m+n+t))
```

LINKED LIST & ARRAY OF DATA

- Example of single linked list

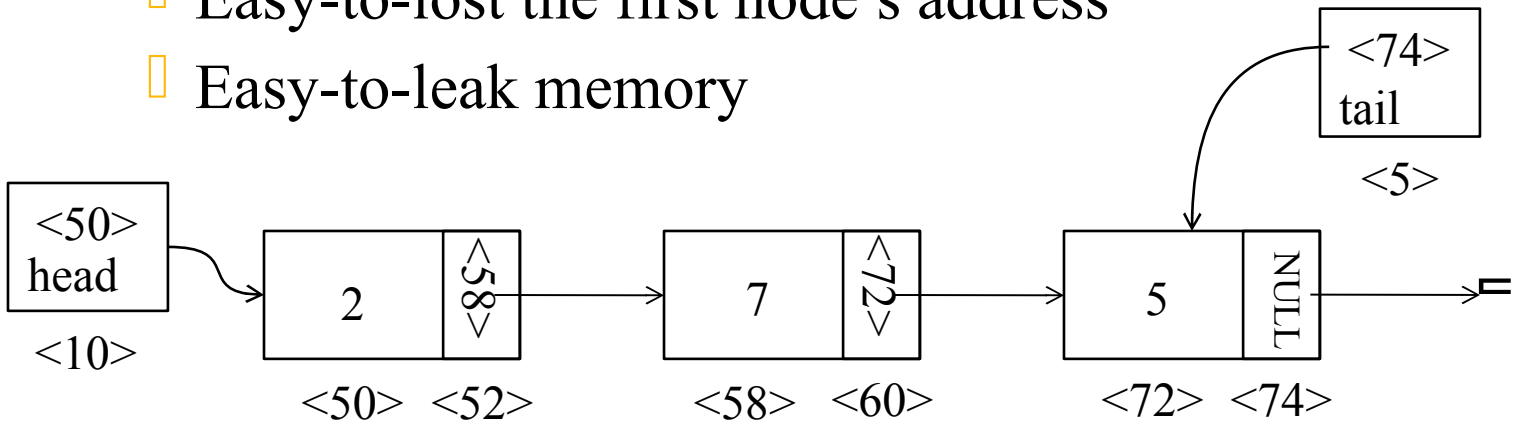
  - Contain an address similar to dynamic array ⊏ efficient memory
  - Easily editing this list
  - Not affected by fragmentation due to the size of each node is small enough



LINKED LIST & ARRAY OF DATA

- Example of single linked list

  - Drawbacks
    - Orderly access
    - Easy-to-lost the first node's address
    - Easy-to-leak memory

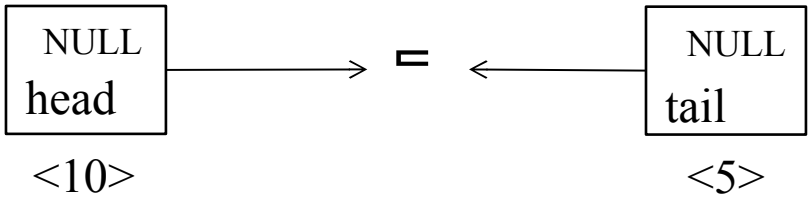


SINGLE LINKED LIST

- Definition

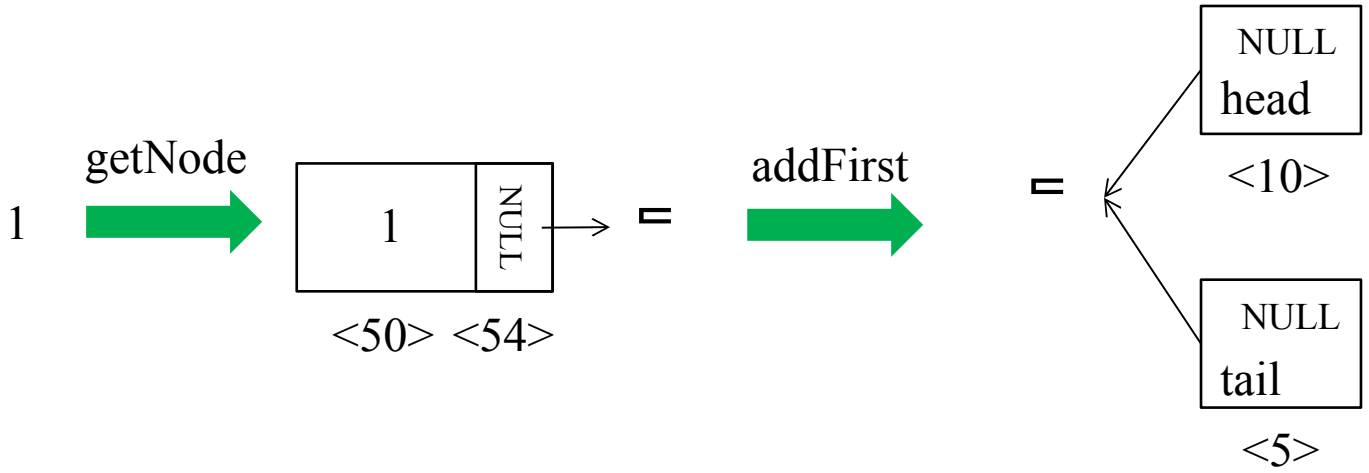
```
typedef struct node* ref;
struct node{
  int key;
  ref next;
};
```

- Initialization
  - ref head = NULL, tail = NULL;



SINGLE LINKED LIST

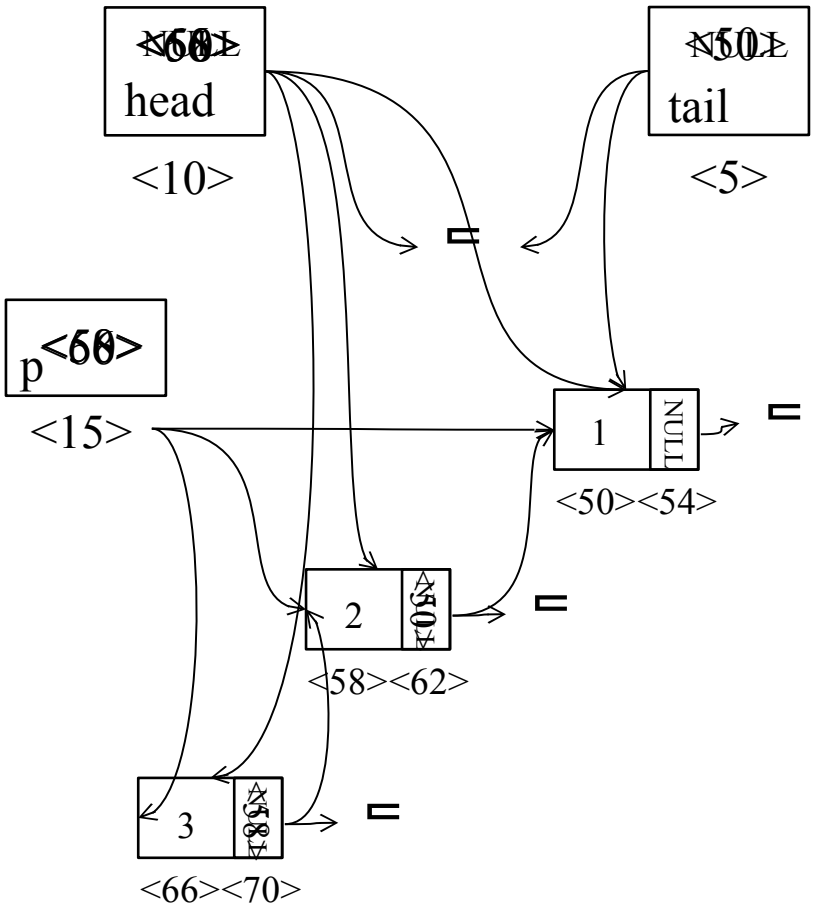
- Building single linked list
  - Step 1: user inputs data needed to create node
  - Step 2: function getNode() creates node
  - Step 3: function addFirst() add node at the start of the list
  - Step 4: go to step 1 or stop



SINGLE LINKED LIST

- Building

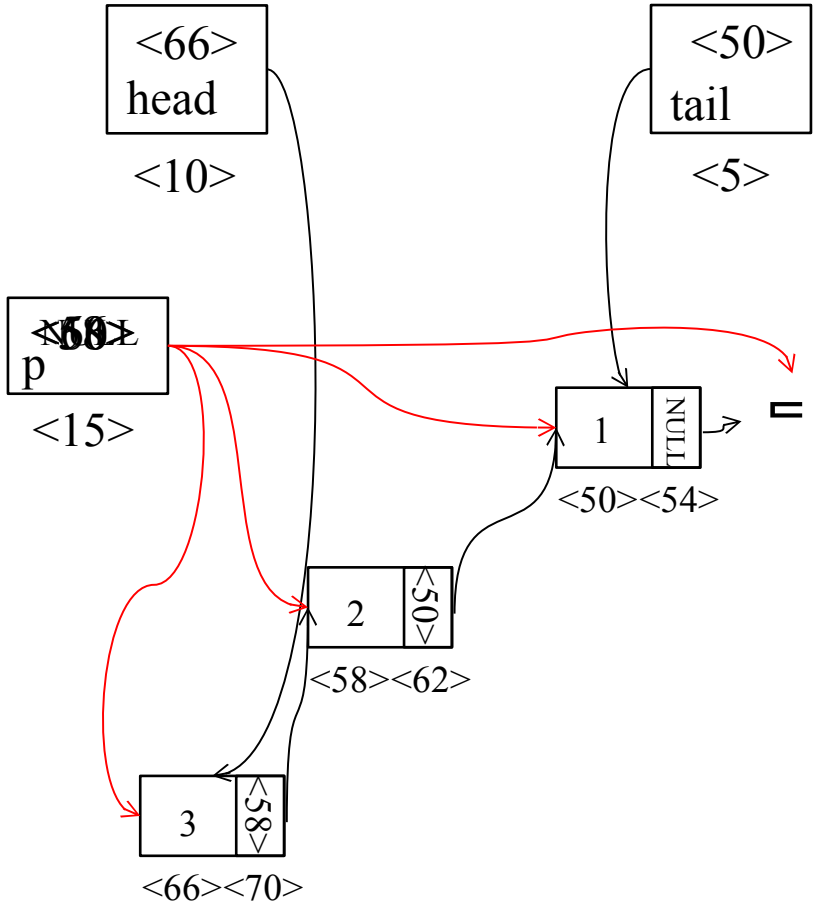
Lines	Description
1	ref getNode(int k){
2	ref p = (ref)malloc(sizeof(struct node));
3	if(p == NULL) return NULL;
4	p->key = k;
5	p->next = NULL;
6	return p;
7	}
8	void addFirst(ref& head, ref& tail, int k){
9	ref p = getNode(k);
10	if(head == NULL) head = tail = p;
11	else{p->next = head; head = p;}
12	}



SINGLE LINKED LIST

- Scanning

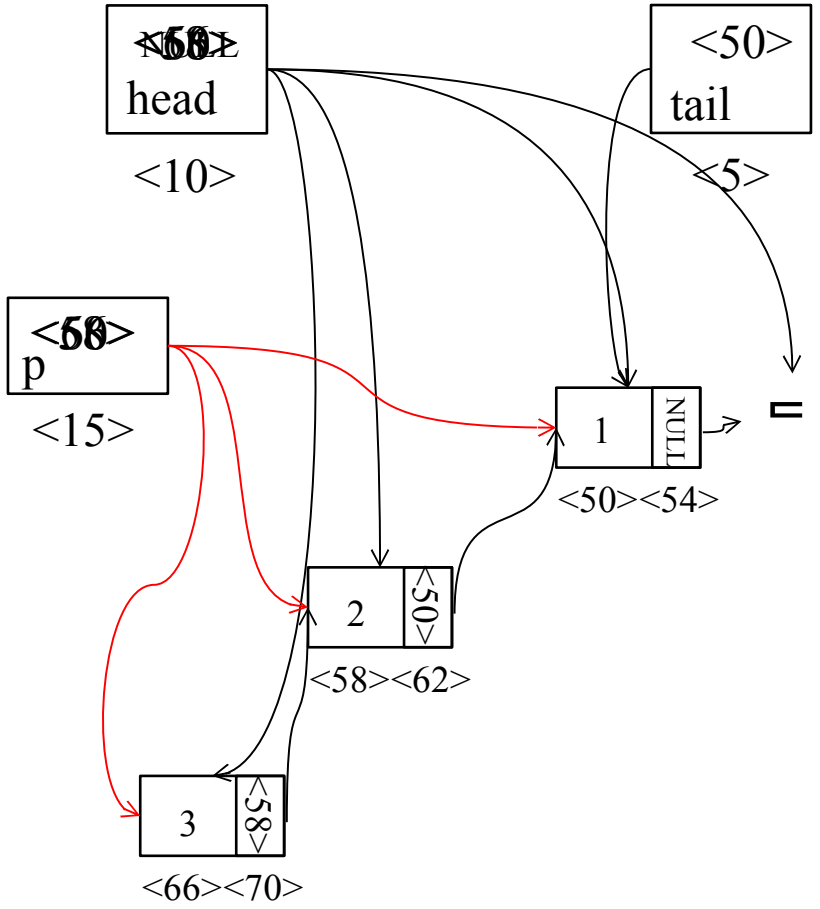
Lines	Description
1	void printList(ref head){
2	ref p;
3	if(head == NULL) return;
4	else{
5	for(p = head; p != NULL; p = p->next)
6	printf("%d\n", p->key);
7	}
8	}



SINGLE LINKED LIST

- Destroying

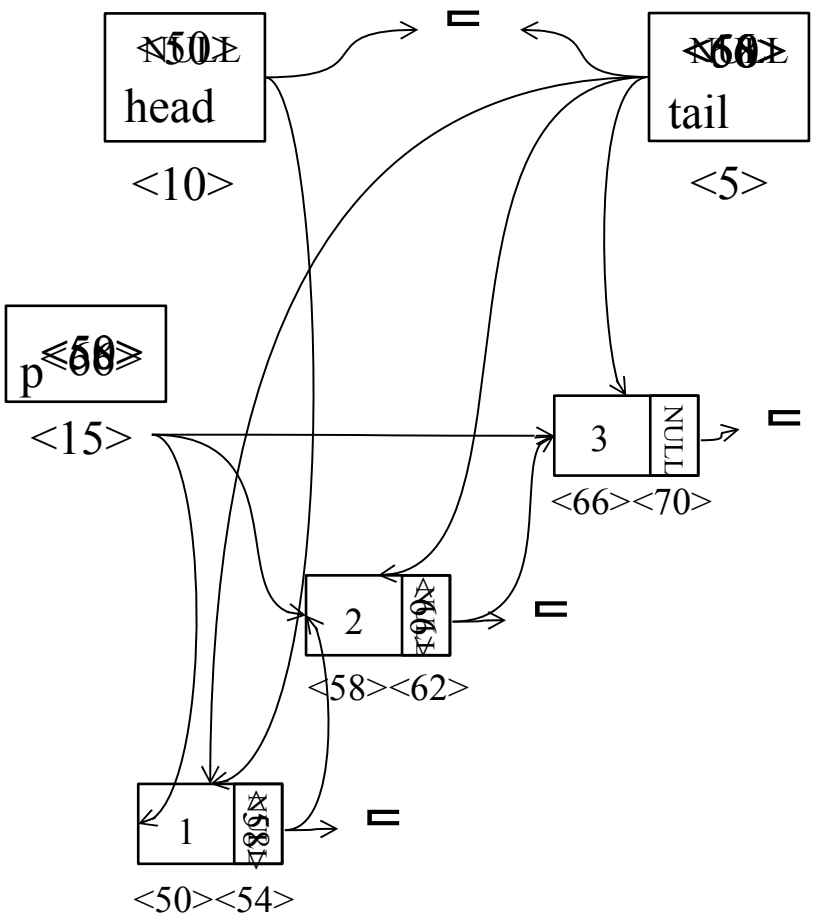
Lines	Description
1	void destroyList(ref& head){
2	ref p;
3	if(head == NULL) return;
4	while(head)
5	p = head;
6	head = head->next;
7	free(p);
8	}
9	}



SINGLE LINKED LIST

- Adding item at the end of the list

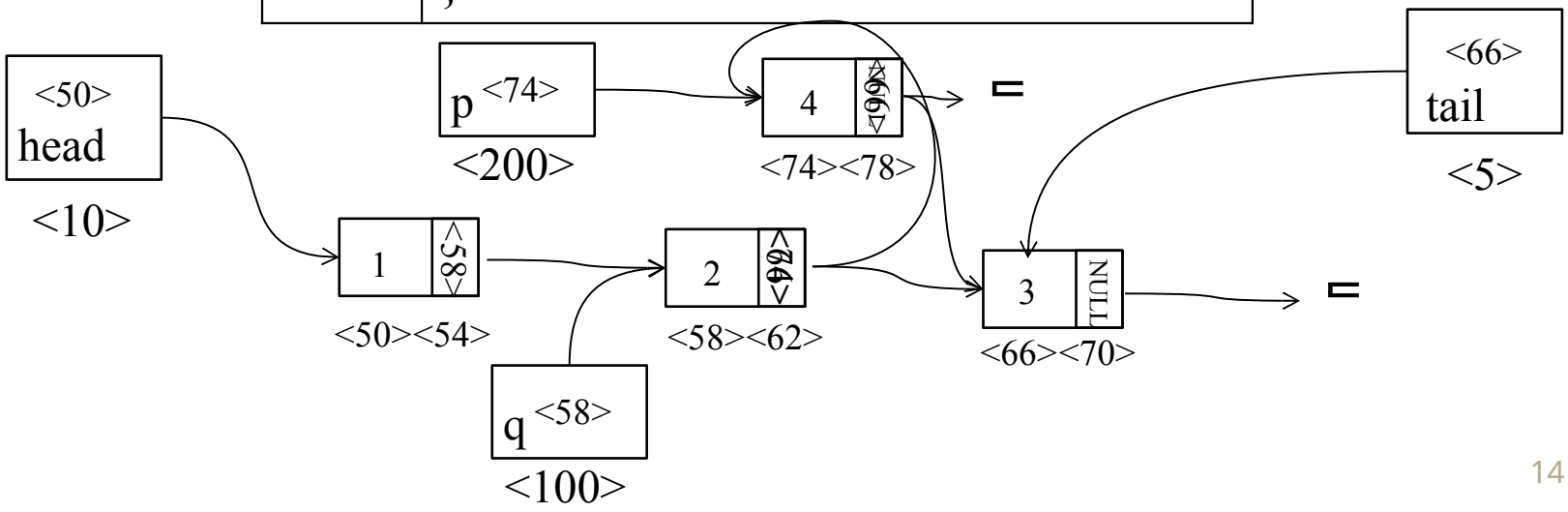
Lines	Description
1	ref getNode(int k){
2	ref p = (ref)malloc(sizeof(struct node));
3	if(p == NULL) return NULL;
4	p->key = k;
5	p->next = NULL;
6	return p;
7	}
8	void addLast(ref& head, ref& tail, int k){
9	ref p = getNode(k);
10	if(head == NULL) head = tail = p;
11	else{tail->next = p; tail = p;}
12	}



SINGLE LINKED LIST

- Insert an item after another item

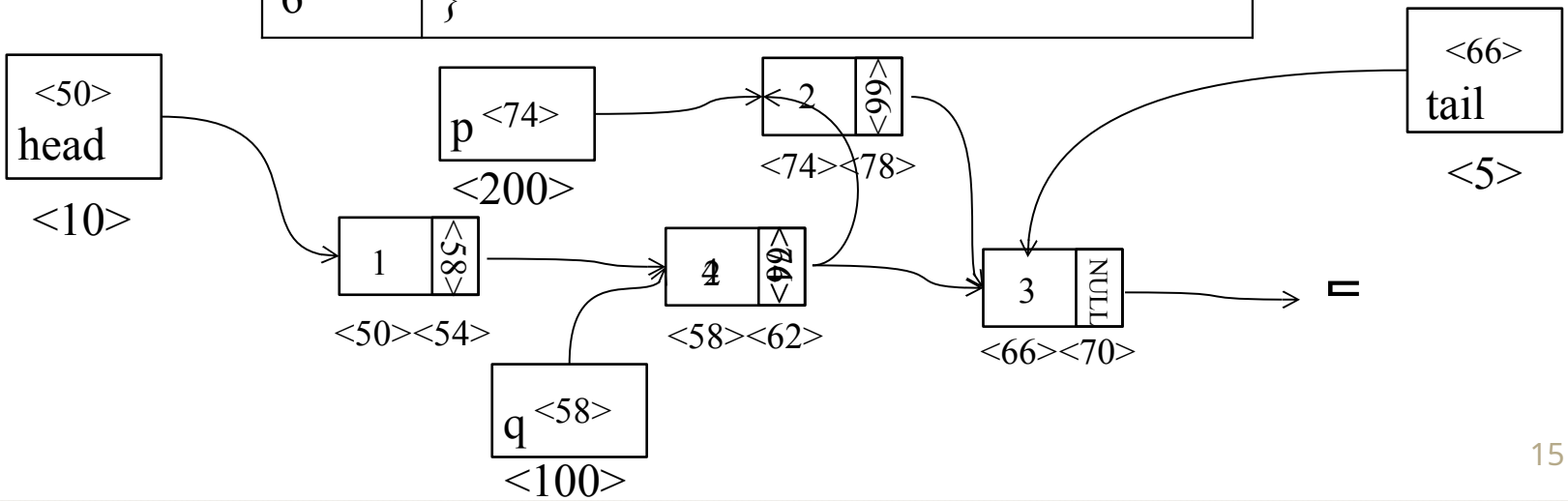
Lines	Description
1	void insertAfter (ref q, int k){
2	ref p = getNode(k);
3	p->next = q->next;
4	q->next = p;
5	}



SINGLE LINKED LIST

- Adding an item before another item

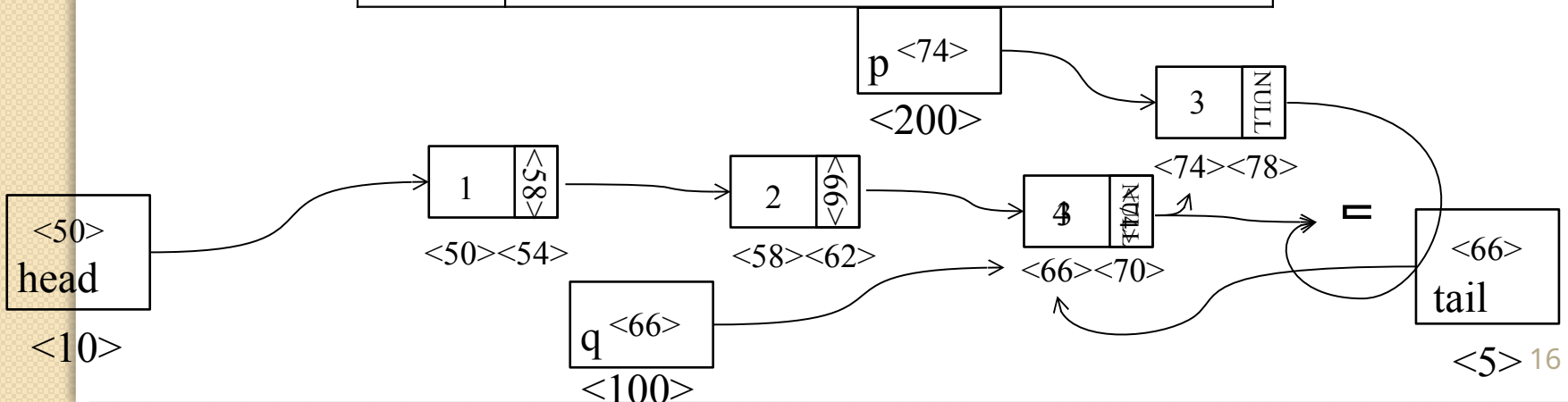
Lines	Description
1	void insertBefore (ref q, int k){// k = 4
2	ref p = (ref)malloc(sizeof(struct node));
3	*p = *q;
4	q->next = p;
5	q->key = k;
6	}



SINGLE LINKED LIST

- Adding an item before another item (error when q points to the last item)

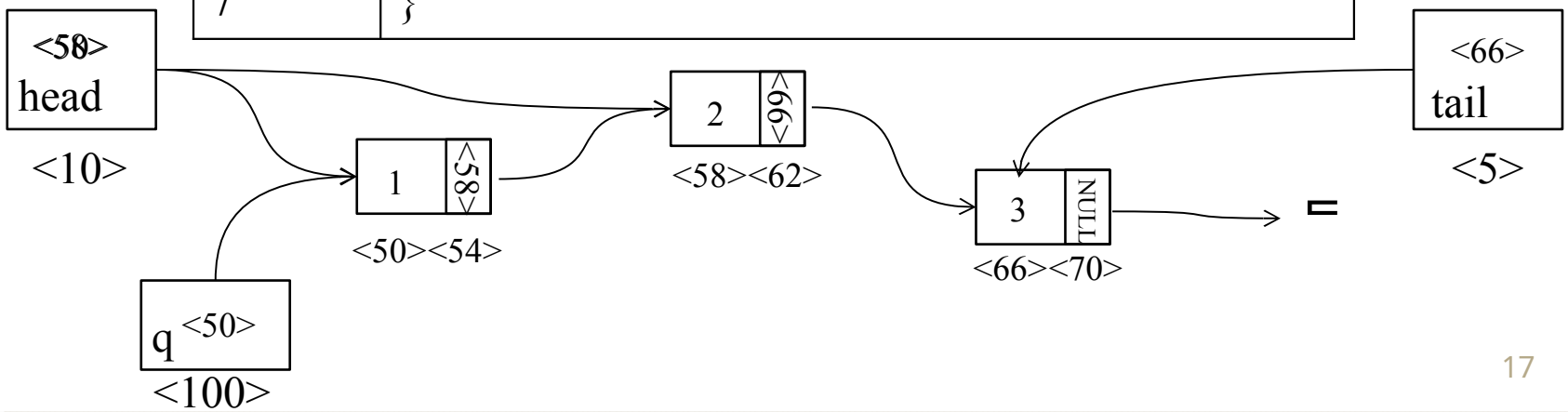
Lines	Description
1	void insertBefore (ref q, int k){ // k = 4
2	ref p = (ref)malloc(sizeof(struct node));
3	*p = *q;
4	q->next = p;
5	q->key = k;
6	}



SINGLE LINKED LIST

- Deleting the first item of the list

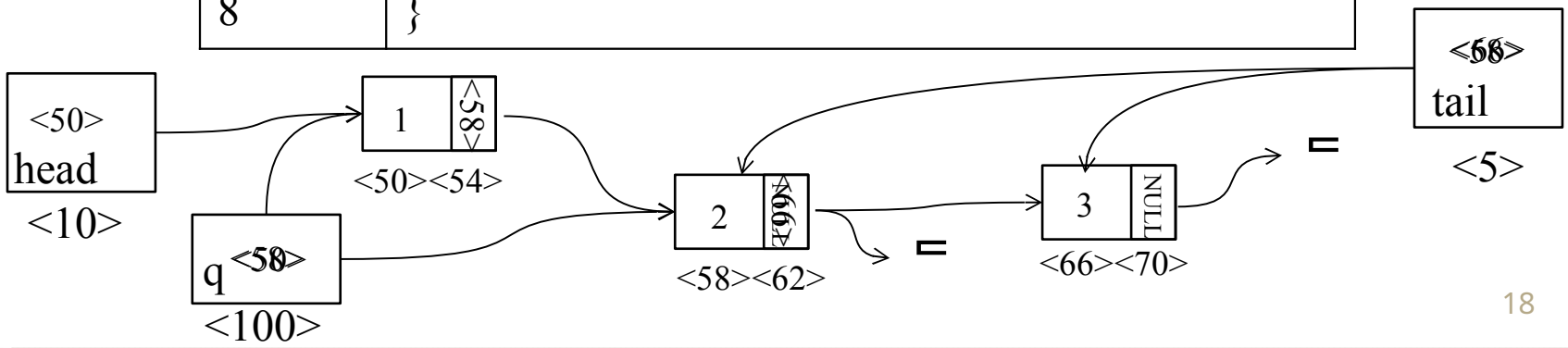
Lines	Desciption
1	void deleteBegin(ref& head, ref& tail){
2	if(head == tail) { free(head); head = tail = NULL}
3	else{
4	ref q = head;
5	head = head->next;
6	free(q);
7	}



SINGLE LINKED LIST

- Deleting the last item of the list

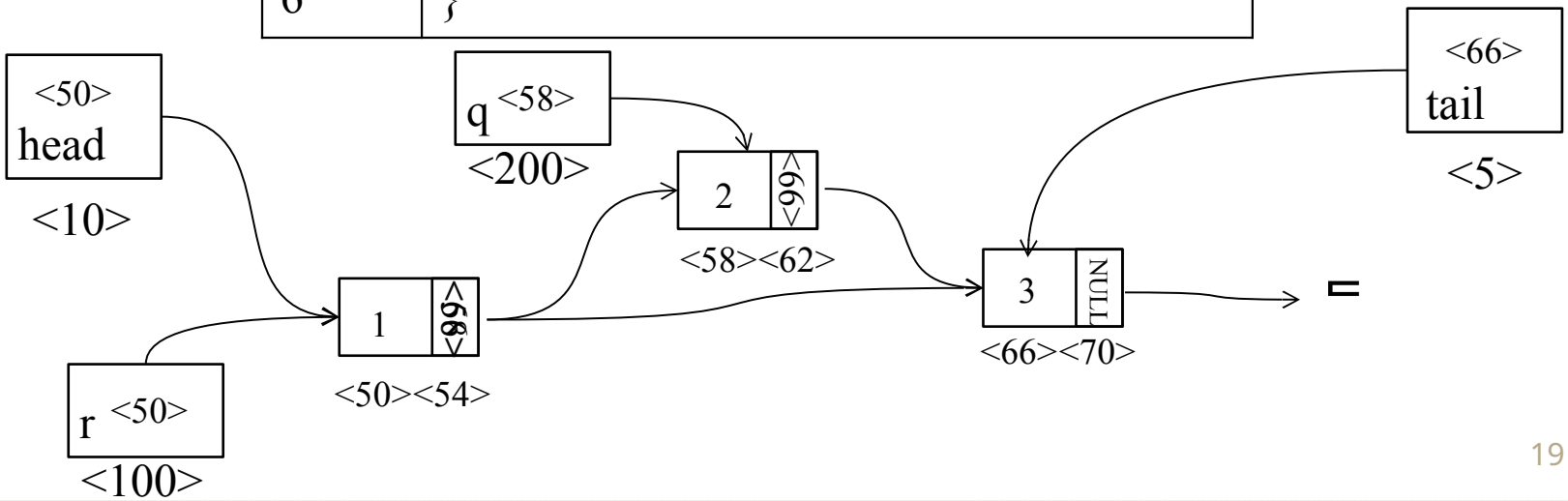
Lines	Description
1	void deleteEnd(ref& head, ref& tail){
2	if(head == tail) { free(head); head = tail = NULL}
3	else{
4	for(ref q = head; q->next!=tail; q=q->next);
5	free(tail);
6	tail = q;
7	q->next = NULL;
8	}



SINGLE LINKED LIST

- Deleting another inner item of the list

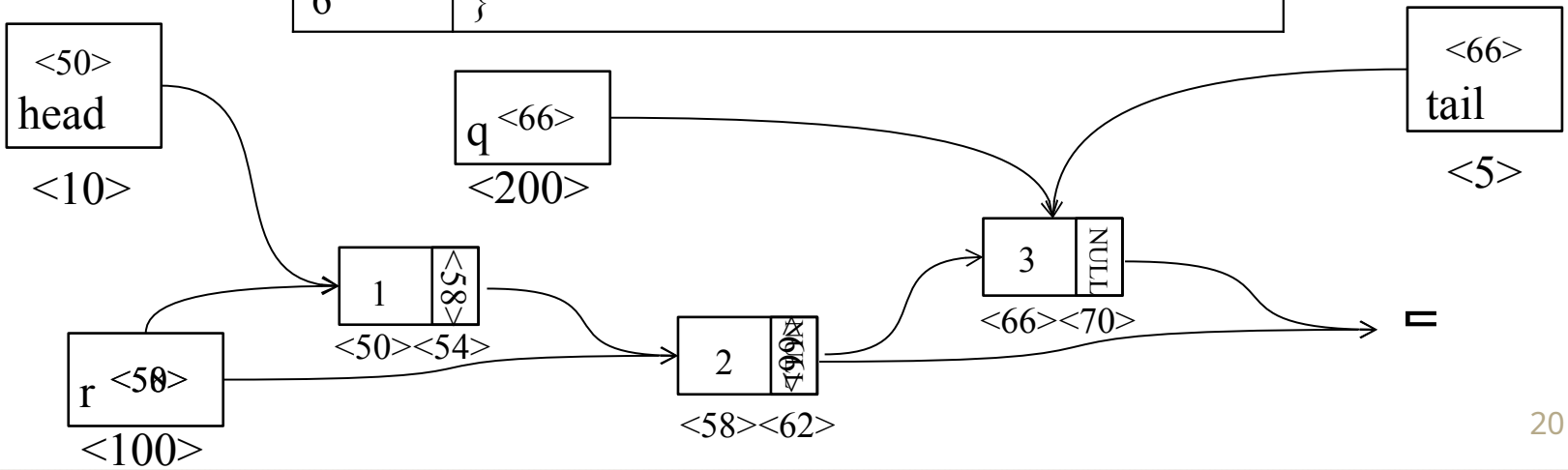
Lines	Description
1	void deleteMid(ref& head, ref q){
2	ref r;
3	for(r = head; r->next != q; r = r->next);
4	r->next = q->next;
5	free(q);
6	}



SINGLE LINKED LIST

- Deleting another inner item of the list (error when q points to the last item)

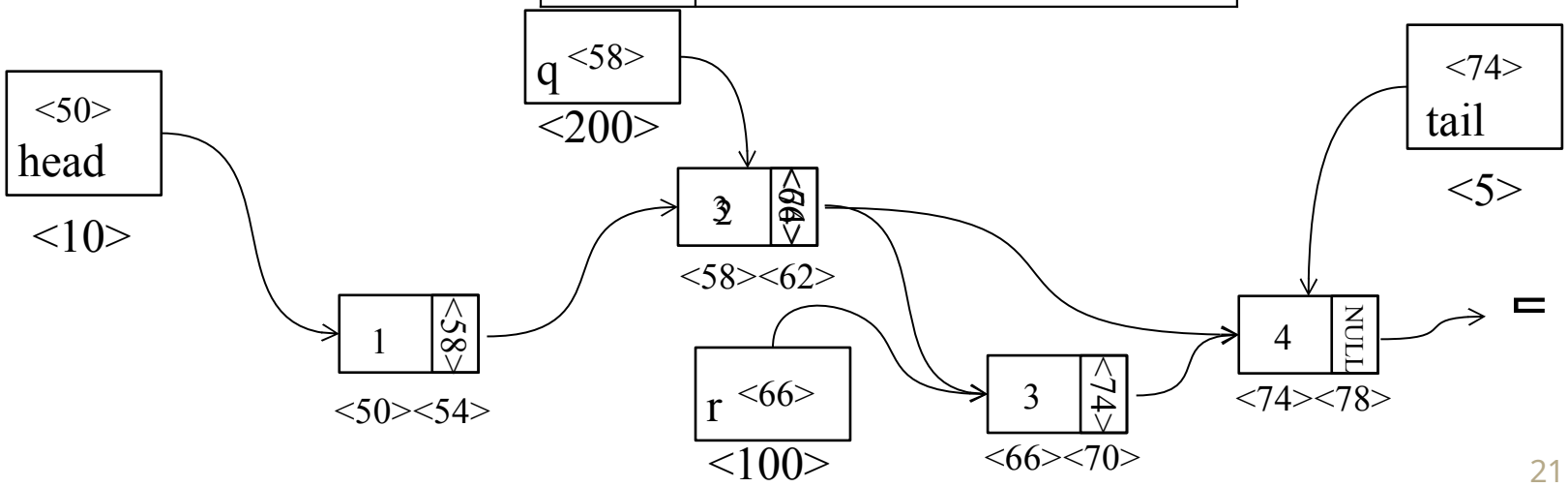
Lines	Description
1	void deleteMid(ref& head, ref q){
2	ref r;
3	for(r = head; r->next != q; r = r->next);
4	r->next = q->next;
5	free(q);
6	}



SINGLE LINKED LIST

- Deleting another inner item of the list (improved)

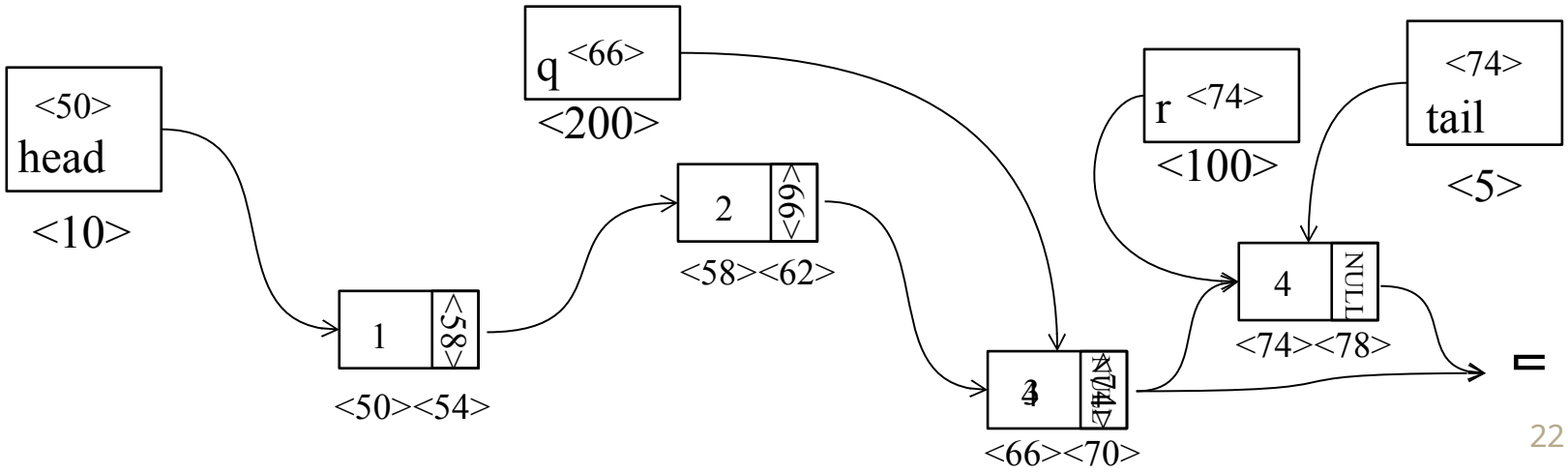
Lines	Description
1	void delMid(ref q){
2	ref r = q->next;
3	*q = *r;
4	free(r);
5	}



SINGLE LINKED LIST

- Deleting another inner item of the list (error when q points to the penultimate item)

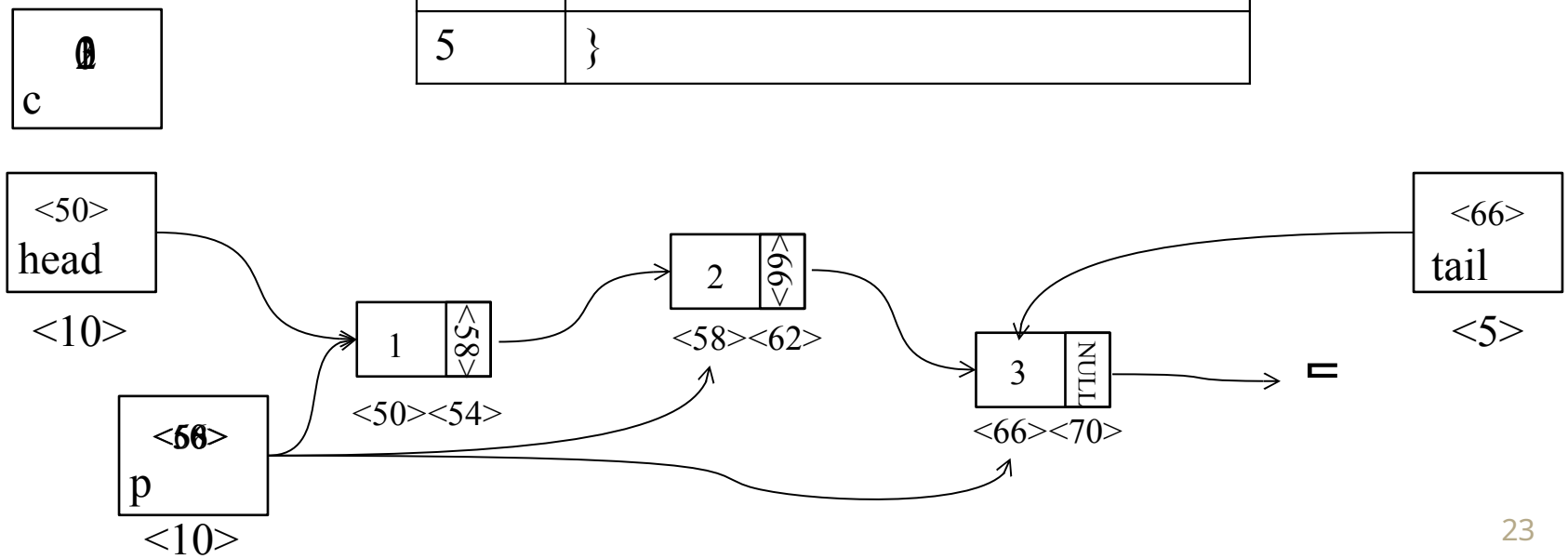
Lines	Description
1	void delMid(ref q){
2	ref r = q->next;
3	*q = *r;
4	free(r);
5	}



SOME OPERATIONS

- Count a number of nodes of linked list

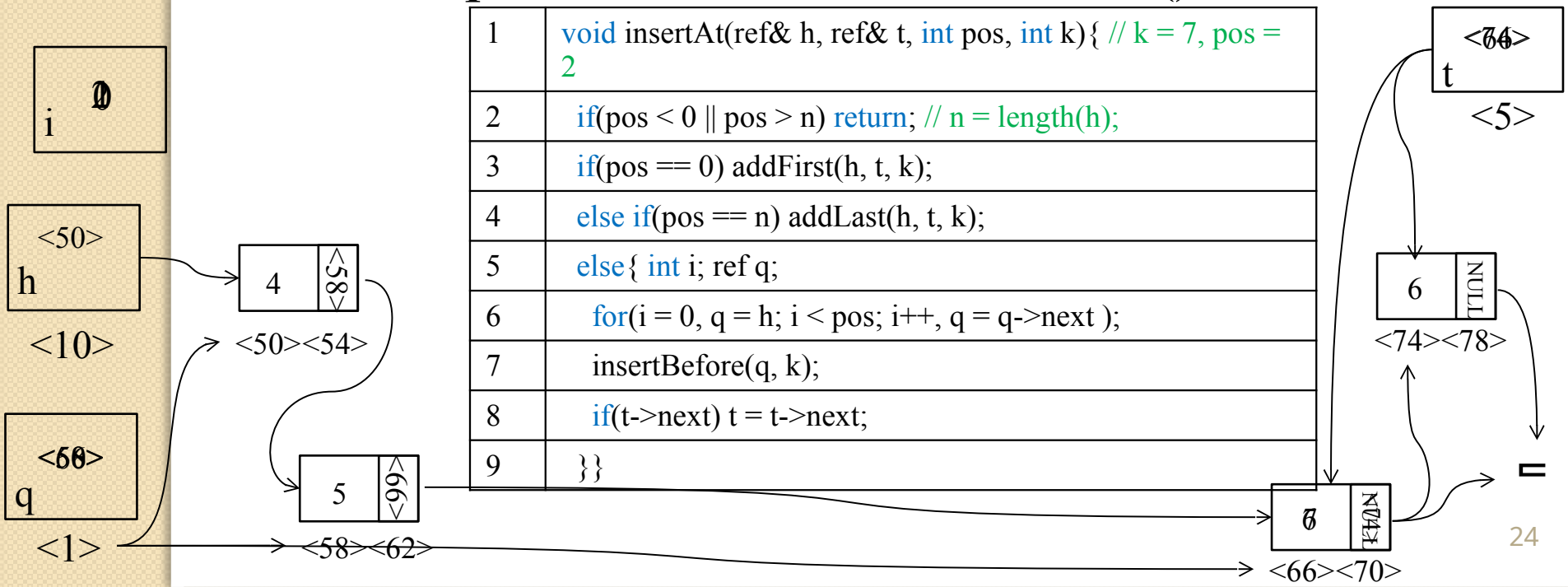
Lines	Description
1	int length(ref head){
2	ref p; int c = 0;
3	for(p = head; p; p = p->next) c++;
4	return c;
5	}



SOME OPERATIONS

- Insert node at the position (named pos)
  - pos = 0 => call addFirst()
  - pos = n => call addLast(), with n = length()
  - 0 < pos < n => call insertBefore()

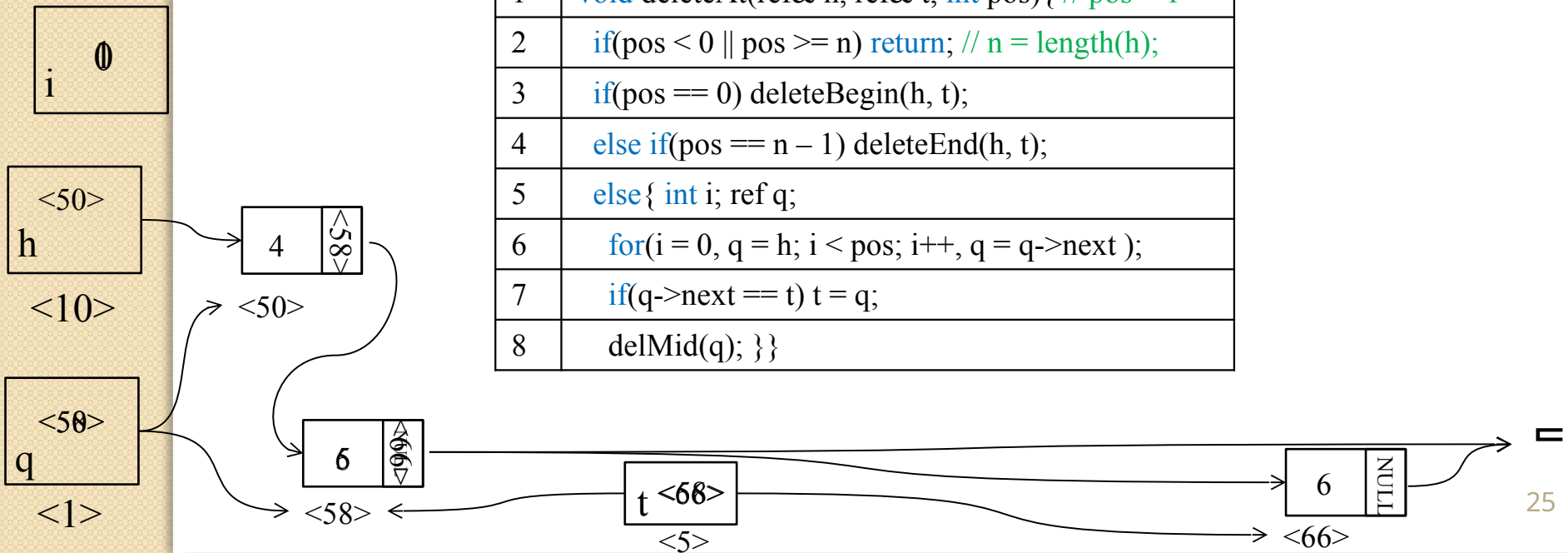
```
1  void insertAt(ref& h, ref& t, int pos, int k){ // k = 7, pos = 2
2    if(pos < 0 || pos > n) return; // n = length(h);
3    if(pos == 0) addFirst(h, t, k);
4    else if(pos == n) addLast(h, t, k);
5    else{ int i; ref q;
6      for(i = 0, q = h; i < pos; i++, q = q->next );
7      insertBefore(q, k);
8      if(t->next) t = t->next;
9    }}
```



SOME OPERATIONS

- Delete node at the position (named pos)
  - pos = 0 => call deleteBegin()
  - pos = n – 1 => call deleteEnd()
  - 0 < pos < n – 1 => call delMid(ref)

```
1  void deleteAt(ref& h, ref& t, int pos){ // pos = 1
2    if(pos < 0 || pos >= n) return; // n = length(h);
3    if(pos == 0) deleteBegin(h, t);
4    else if(pos == n – 1) deleteEnd(h, t);
5    else{ int i; ref q;
6      for(i = 0, q = h; i < pos; i++, q = q->next );
7      if(q->next == t) t = q;
8      delMid(q); }}
```

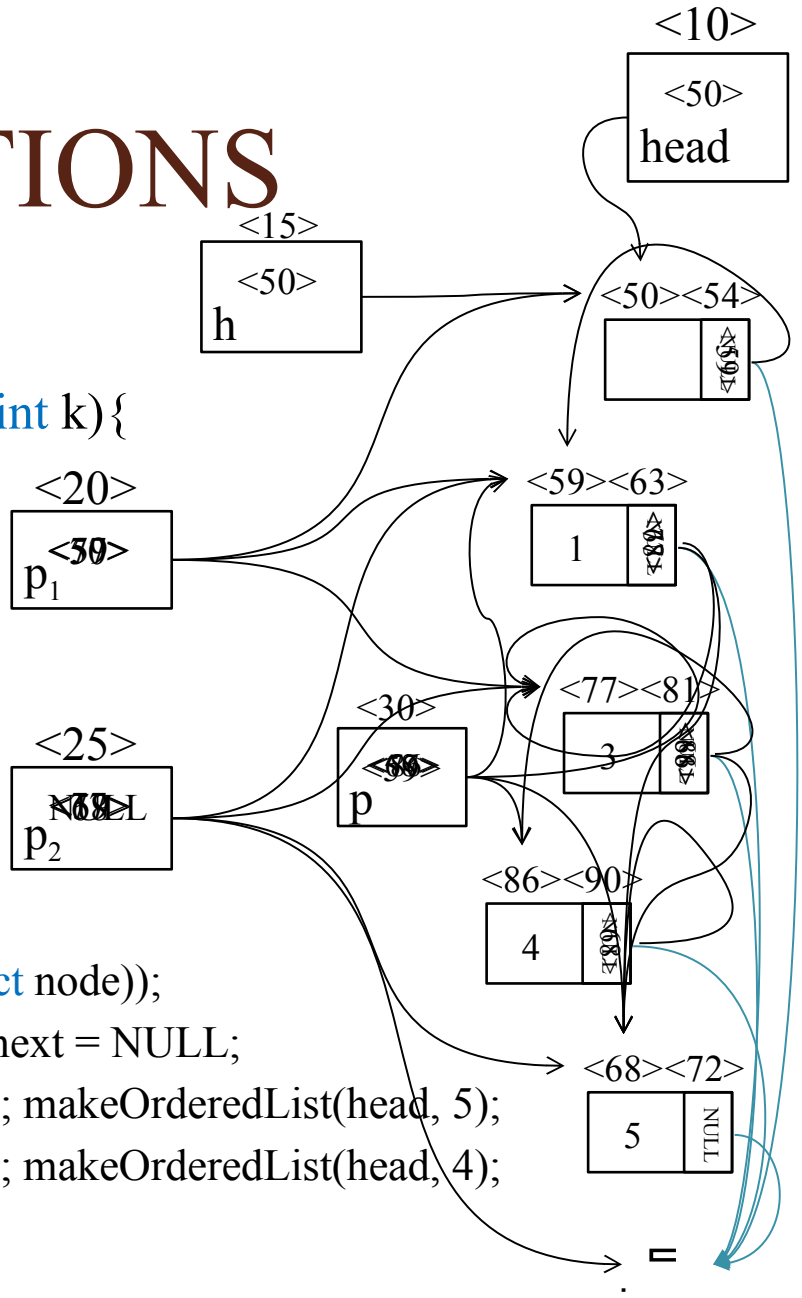


SOME OPERATIONS

- Building linked list with increasing order
  - Using the method of dummy head
  - Maintain 2 pointers p_1 & p_2 pointing to 2 adjacent nodes of the list
  - At first, if the list is empty, there is no exist p_2 => need a fake node head (dummy head)
  - New node k^{th} always is at between p_1 & p_2
- Demonstration with integers

SOME OPERATIONS

- Example
  - void **makeOrderedList**(ref h, int k){
    - ref p1 = h, p2 = p1->next;
    - while(p2 && p2->key < k){
      - p1 = p2; p2 = p1->next;
    - }
    - ref p = getNode(k);
    - p1->next = p; p->next = p2;
  - }
  - void main(){
    - ref head = (ref)malloc(sizeof(struct node));
    - if(head != NULL) head->next = NULL;
    - makeOrderedList(head, 1); makeOrderedList(head, 5);
    - makeOrderedList(head, 3); makeOrderedList(head, 4);
    - printList(head->next);
    - destroyList(head);
  - }

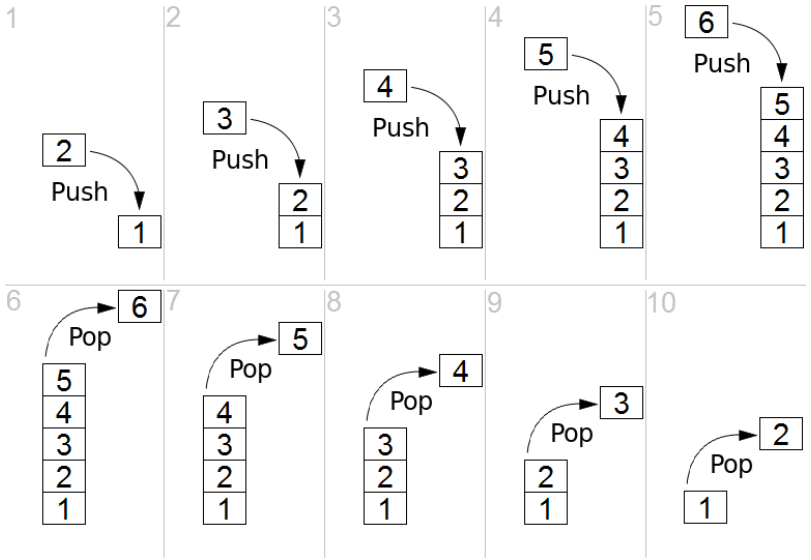


STACK & QUEUE

- Stack and Queue are the collection of the items
- Collection often has 3 basic operations
  - Add an item
  - Delete an item
  - Check if the collection is empty or not
- Stack has 3 operations: push, pop, and isEmpty
- Queue has 3 operations: enQueue, deQueue, and isEmpty

STACK

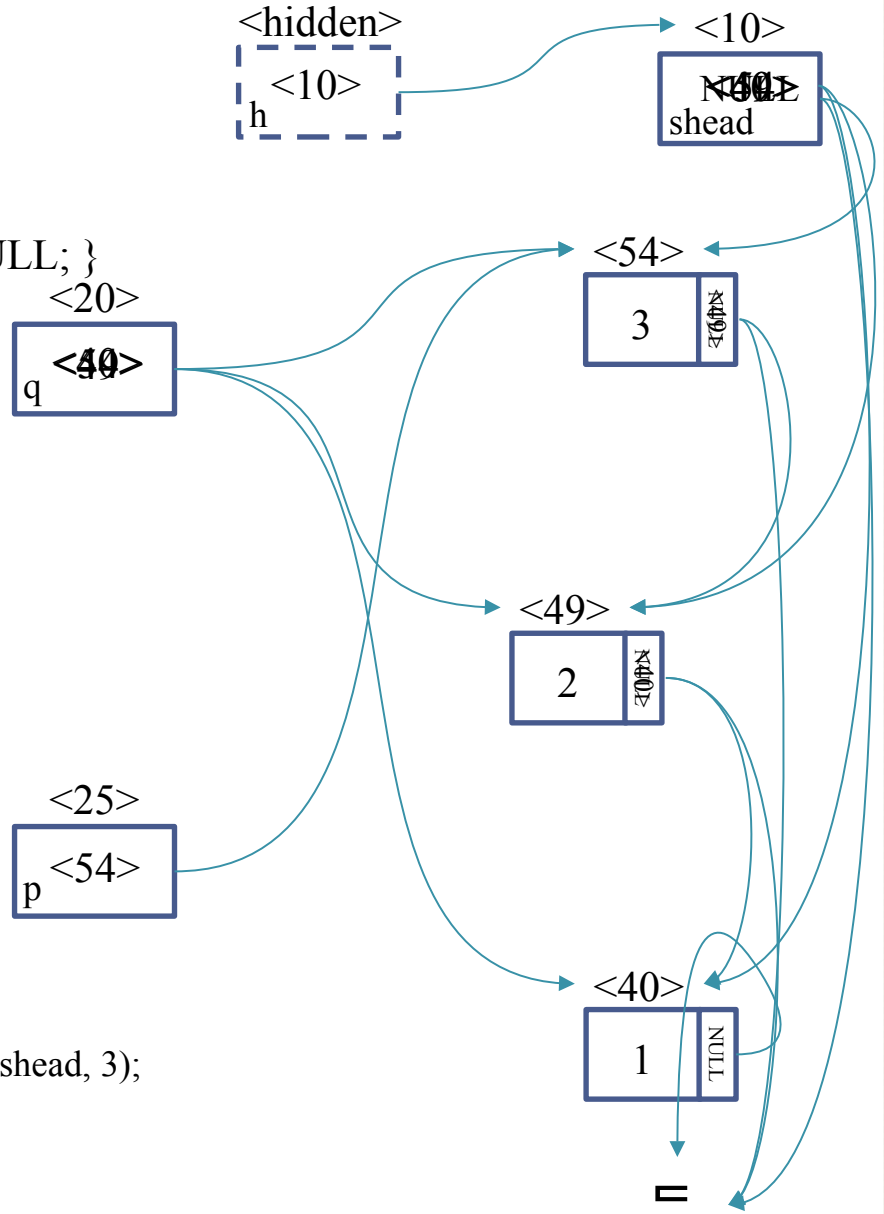
- Stack
  - Performed with last-in-first-out – LIFO
  - May use linked list or array to implement a stack



Source: *https://en.wikipedia.org/wiki/Stack_(abstract_data_type)*

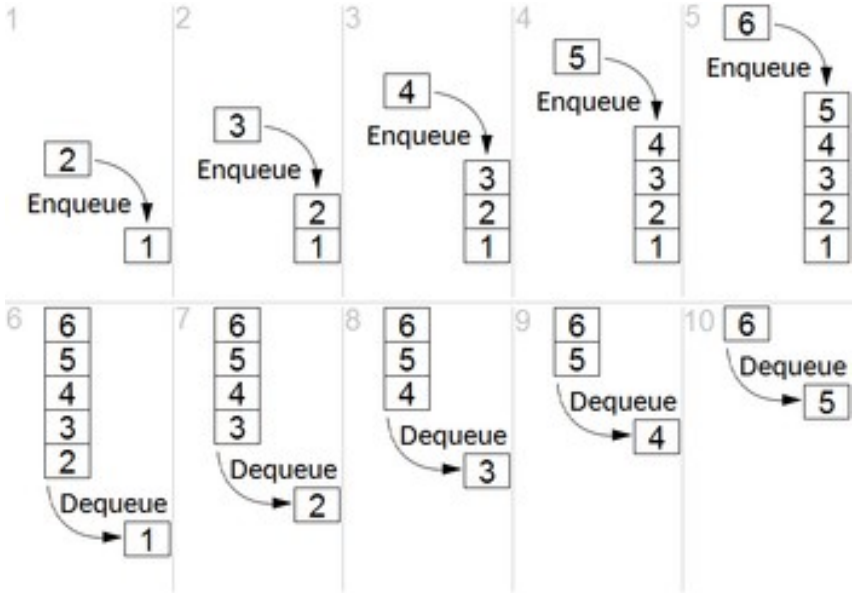
STACK

- Stack
  - int isEmpty(ref h){ return h == NULL; }
  - void push(ref& h, int k){
    - ref q = getNode(k);
    - q->next = h;
    - h = q;
  - }
  - ref pop(ref& h){
    - if(isEmpty(h)) return NULL;
    - ref q = h;
    - h = h->next;
    - q->next = NULL;
    - return q;
  - }
  - void main(){
    - ref shead = NULL;
    - push(shead, 1); push(shead, 2); push(shead, 3);
    - ref p = pop(shead);
    - free(p);
  - }



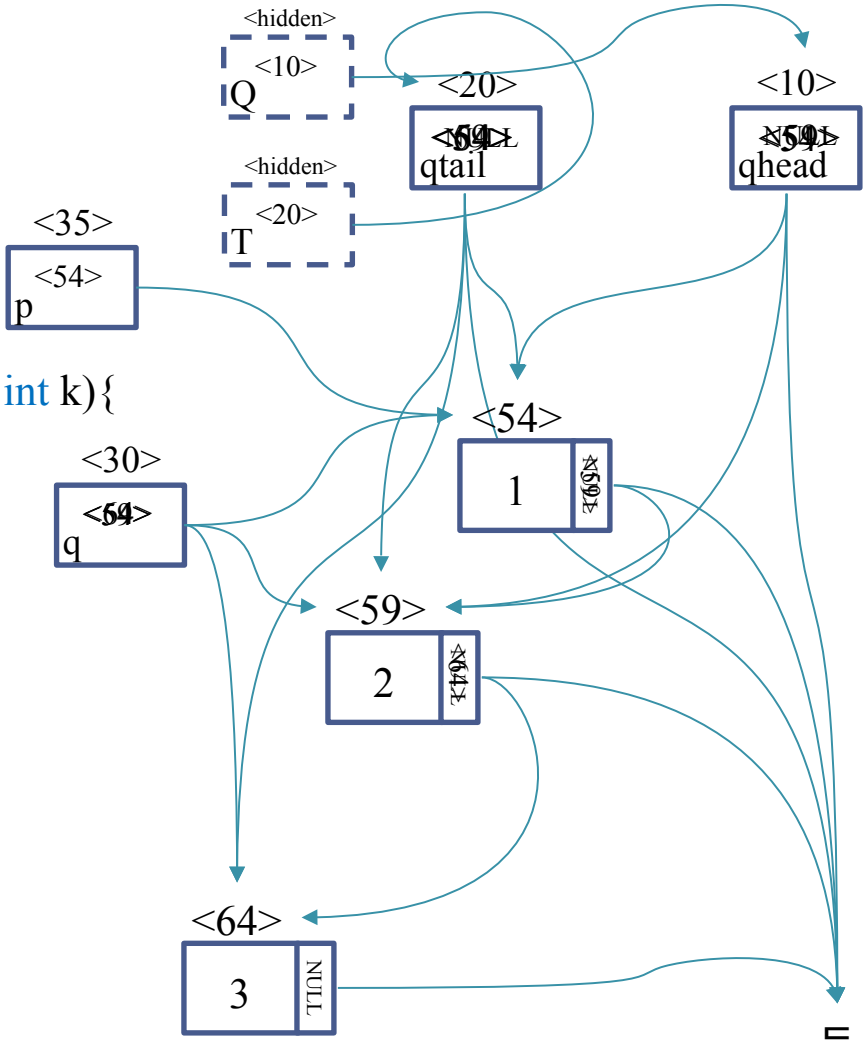
QUEUE

- Queue
  - Performed with first-in-first-out – FIFO
  - May use linked list or array to implement a queue



Source: *https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)*

QUEUE



- Queue
  - void enQueue(ref& Q, ref& T, int k){
    - ref q = getNode(k);
    - if(isEmpty(Q)) Q = T = q;
    - else T = T->next = q;
  - }
  - ref deQueue(ref& Q, ref& T){
    - if(isEmpty(Q)) return NULL;
    - ref q = Q;
    - if(Q == T) Q = T = NULL;
    - else Q = Q->next;
    - q->next = NULL;
    - return q;
  - }
  - void main(){
    - ref qhead = NULL, qtail = NULL;
    - enQueue(qhead, qtail, 1); enQueue(qhead, qtail, 2); enQueue(qhead, qtail, 3);
    - ref p = deQueue(qhead, qtail);
    - free(p);
  - }