ECE220

Lecture x0012 - 10/28

Linked lists - stacks & queues

Announcements

Announcements

- **Exam on 10/30:**

Announcements

- **Exam on 10/30:**
  - Study material has been posted

Announcements

- **Exam on 10/30:**
  - Study material has been posted
  - HKN Material is available on their website.

Announcements

- **Exam on 10/30:**
  - Study material has been posted
    - HKN Material is available on their website.
  - Lectures 7 through 15 inclusive

Announcements

- **Exam on 10/30:**
  - Study material has been posted
    - HKN Material is available on their website.
- Lectures 7 through 15 inclusive
- No lecture on Thursday.

Announcements

- **Exam on 10/30:**
    - Study material has been posted
        - HKN Material is available on their website.
- Lectures 7 through 15 inclusive
- No lecture on Thursday.
- **Rooms are both in CIF and ECEB!**

Recap

Recap

- Last week - Tuesday

Recap

- Last week - Tuesday
    - Dynamic memory allocation:
      `malloc, calloc, realloc, free`

Recap

- Last week - Tuesday
    - Dynamic memory allocation: `malloc, calloc, realloc, free`
    - Two dimensional arrays

Recap

- Last week - Tuesday
    - Dynamic memory allocation: `malloc, calloc, realloc, free`
    - Two dimensional arrays
    - Reading/writing `structs` to files

Recap

- Last week - Tuesday
    - Dynamic memory allocation: `malloc, calloc, realloc, free`
    - Two dimensional arrays
    - Reading/writing `structs` to files
    - Examples

Recap

- Last week - Tuesday
  - Dynamic memory allocation: `malloc, calloc, realloc, free`
  - Two dimensional arrays
  - Reading/writing `structs` to files
  - Examples
- Last week - Thursday

Recap

- Last week - Tuesday
  - Dynamic memory allocation: `malloc, calloc, realloc, free`
  - Two dimensional arrays
  - Reading/writing `structs` to files
  - Examples
- Last week - Thursday
  - Linked lists

Recap

- Last week - Tuesday
  - Dynamic memory allocation: `malloc, calloc, realloc, free`
  - Two dimensional arrays
  - Reading/writing `structs` to files
  - Examples
- Last week - Thursday
  - Linked lists
  - Traversal

Recap

- Last week - Tuesday
  - Dynamic memory allocation: `malloc, calloc, realloc, free`
  - Two dimensional arrays
  - Reading/writing `structs` to files
  - Examples
- Last week - Thursday
  - Linked lists
  - Traversal
  - Insertion - head, tail, sorted

Recap

- Last week - Tuesday
  - Dynamic memory allocation: `malloc, calloc, realloc, free`
  - Two dimensional arrays
  - Reading/writing `structs` to files
  - Examples
- Last week - Thursday
  - Linked lists
  - Traversal
  - Insertion - head, tail, sorted
  - Deletion - head, tail, middle

Review - singly linked lists (plain)

Review - singly linked lists (plain)

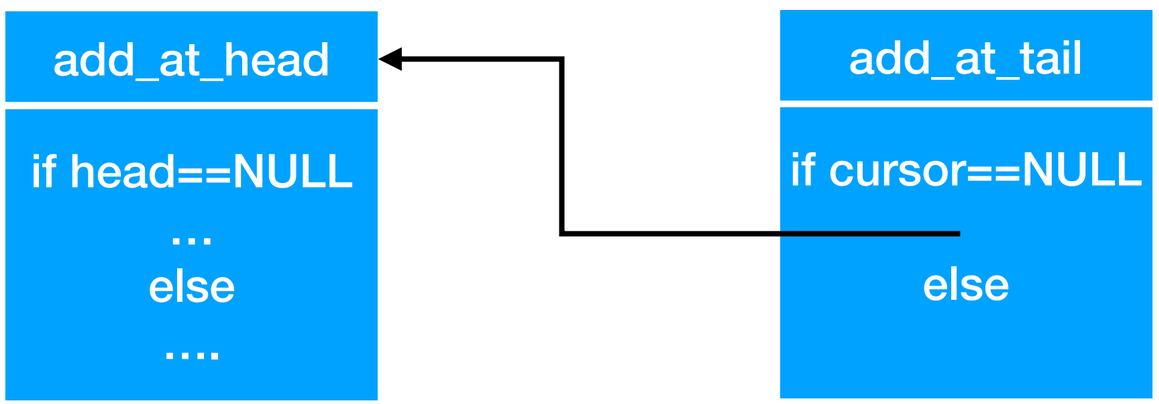
```
add_at_head
if head==NULL
...
else
....
```

Review - singly linked lists (plain)

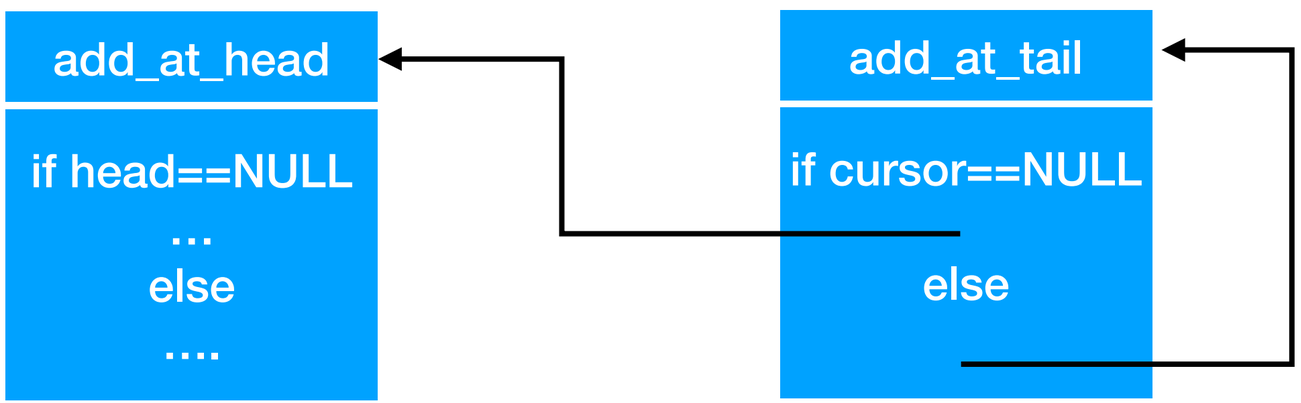
add_at_head
if head==NULL … else ….

add_at_tail
if cursor==NULL else

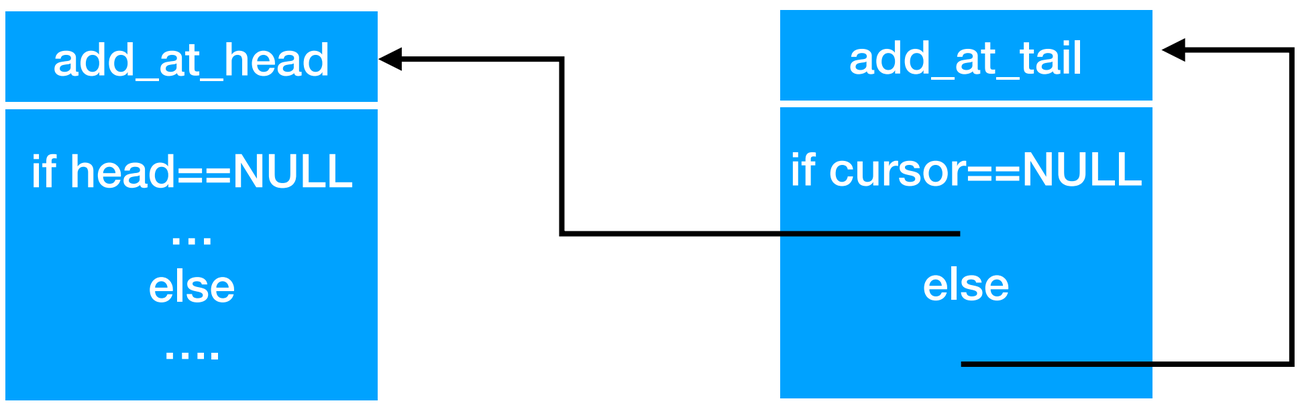
Review - singly linked lists (plain)



Review - singly linked lists (plain)



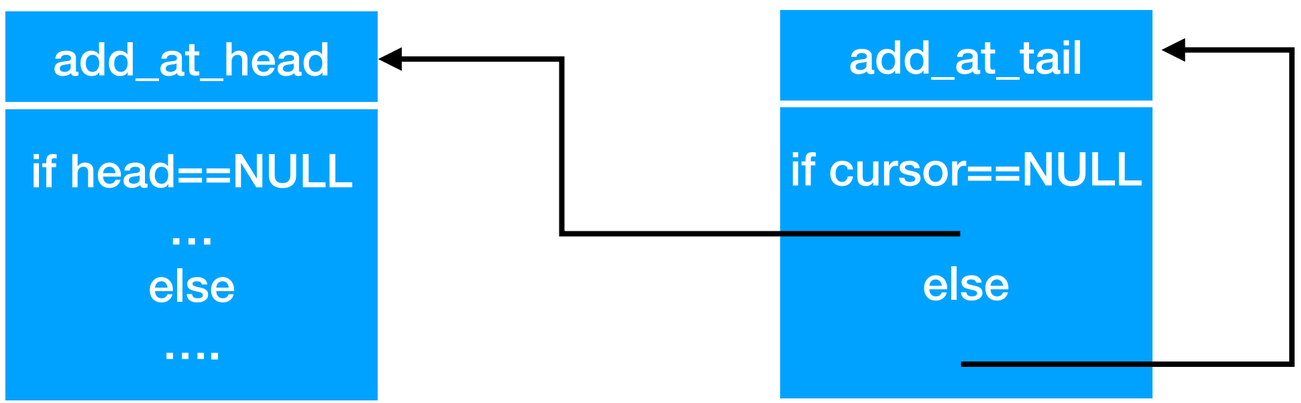
Review - singly linked lists (plain)

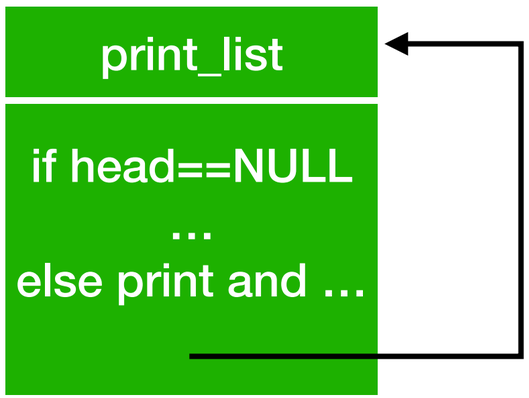


print_list

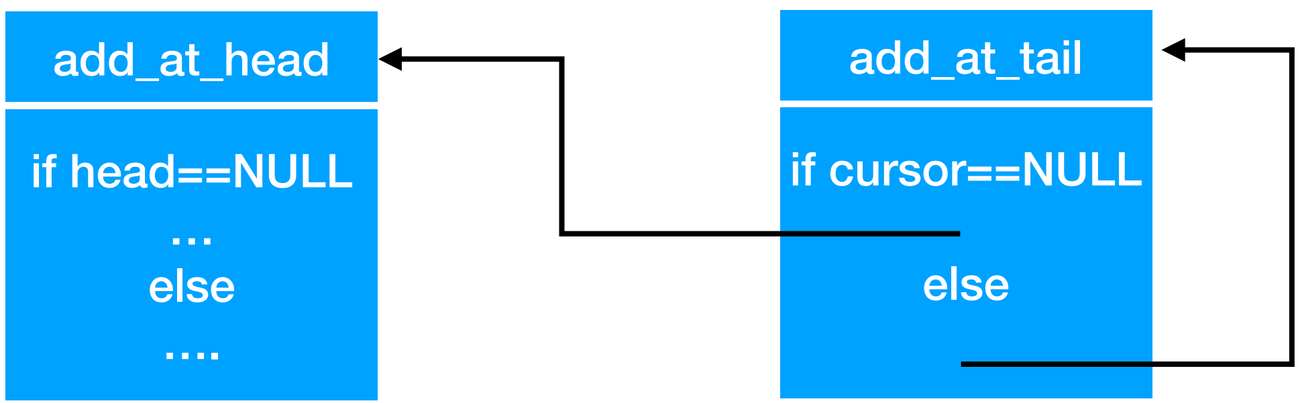
if head==NULL
...
else print and …

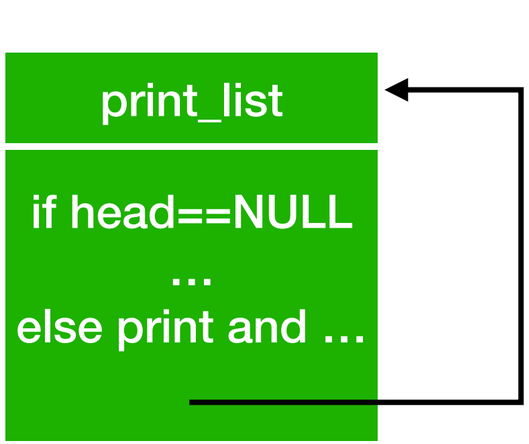
Review - singly linked lists (plain)



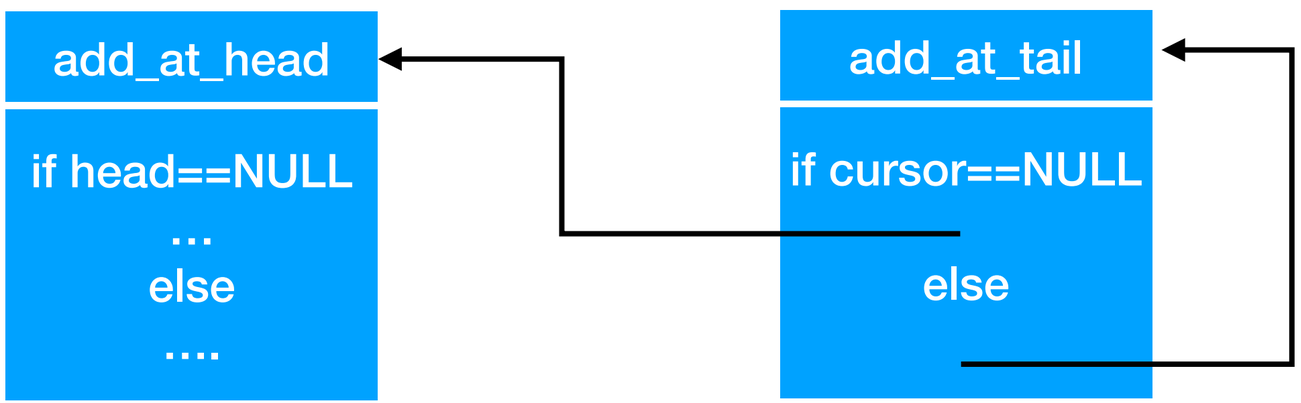


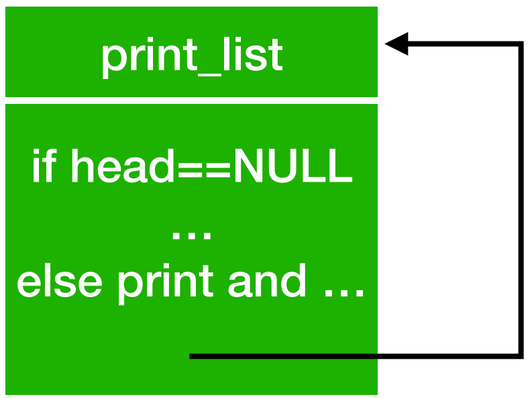
Review - singly linked lists (plain)

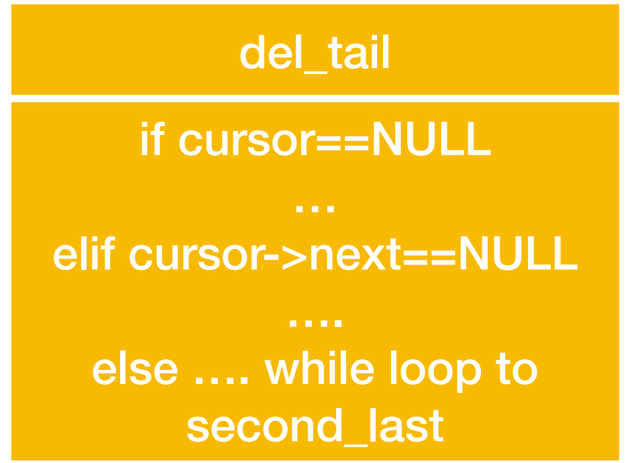




Review - singly linked lists (plain)







Today - Lesson Objectives

- Understand how recursion works in the case of linked lists.
- Understand stacks and queues as special cases of generic linked lists.
- Be able to implement stacks and queues using linked lists.
- Be able to implement binary search on a sorted linked list.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
  temp.data = nums[i];
  temp.next = NULL;
  add_at_tail(&headptr, &temp);
}
```

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
}
```

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
}
```

	x3019		temp.data
	x3020		temp.next
nums	5	2	1

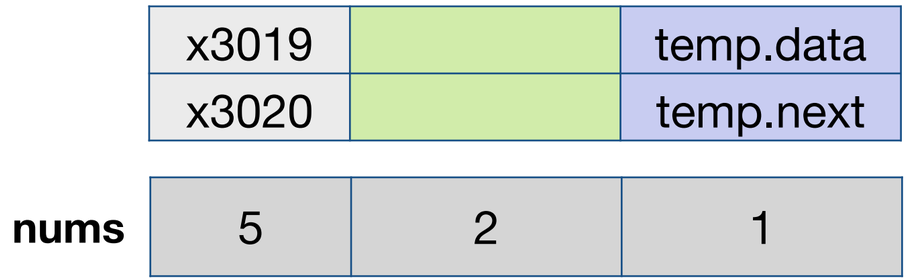
Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
}
```



Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
}
```

	x3000	NULL	headptr
	x3019		temp.data
	x3020		temp.next
nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
  }
```

x3000	NULL	headptr

x3019	5	temp.data
x3020	NULL	temp.next

nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
  }
```

	x3000	NULL	headptr
	x3019	5	temp.data
	x3020	NULL	temp.next
nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

	x3000	NULL	headptr
	x3019	5	temp.data
	x3020	NULL	temp.next
nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

	x3000	NULL	headptr
	x3019	5	temp.data
	x3020	NULL	temp.next
nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

	x3000	NULL	headptr
	x3019	5	temp.data
	x3020	NULL	temp.next
nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

add_at_tail

x1233		cursor
x1234		new

x3000	NULL	headptr
x3019	5	temp.data
x3020	NULL	temp.next

nums

5	2	1

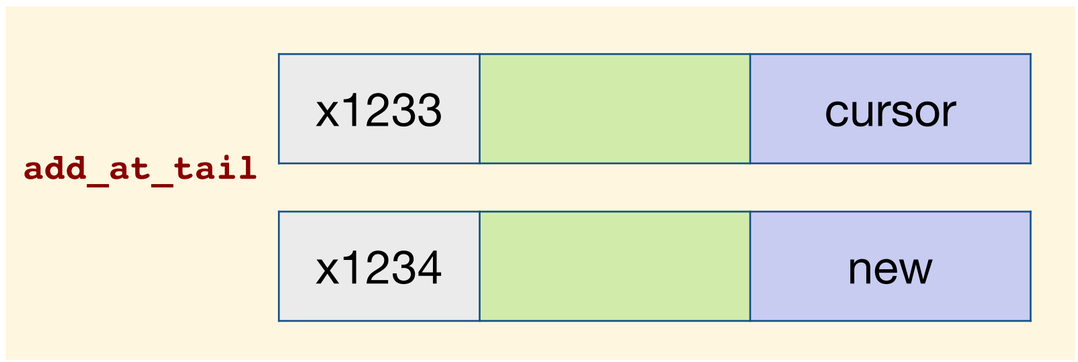
Memory addresses are for illustration — it may not match LC3 CC or RTS.

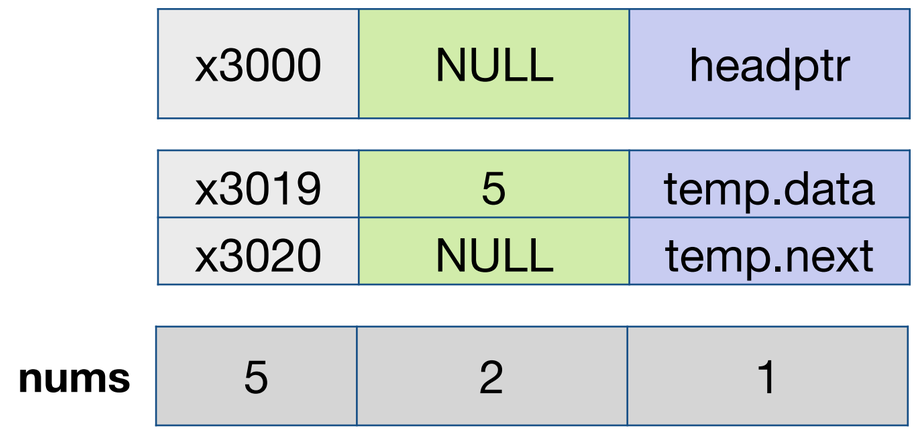
```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&headptr, &temp);
```





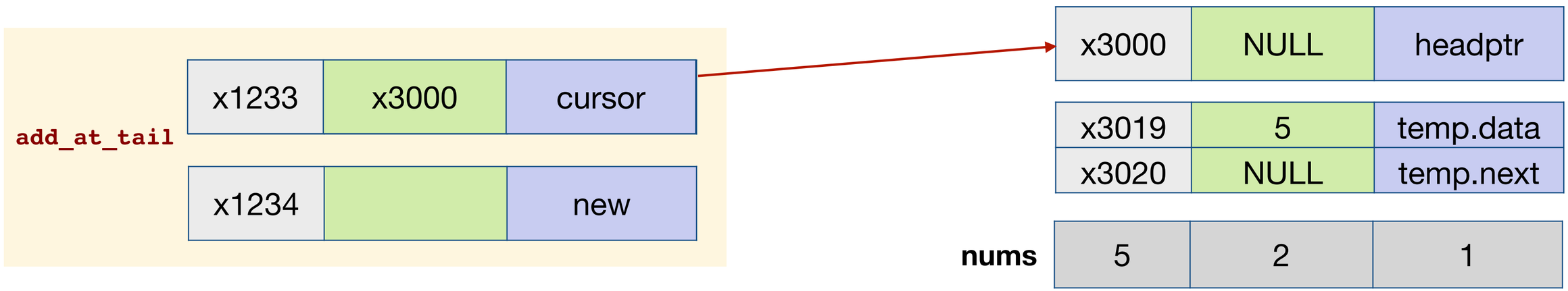
Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&headptr, &temp);
```



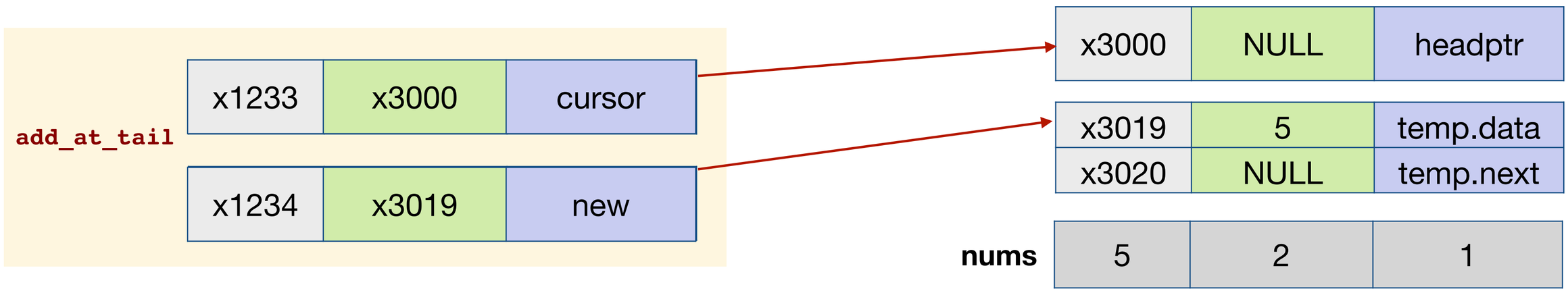
Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&headptr, &temp);
```



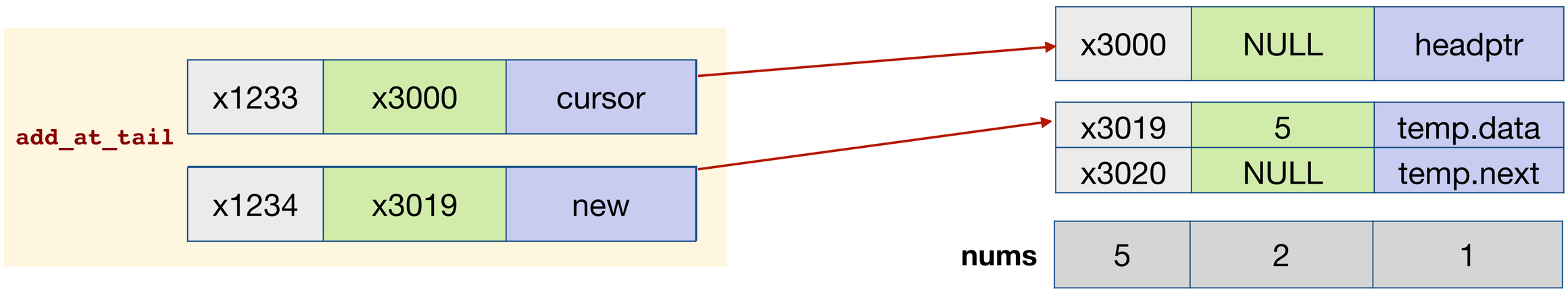
Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&headptr, &temp);
```



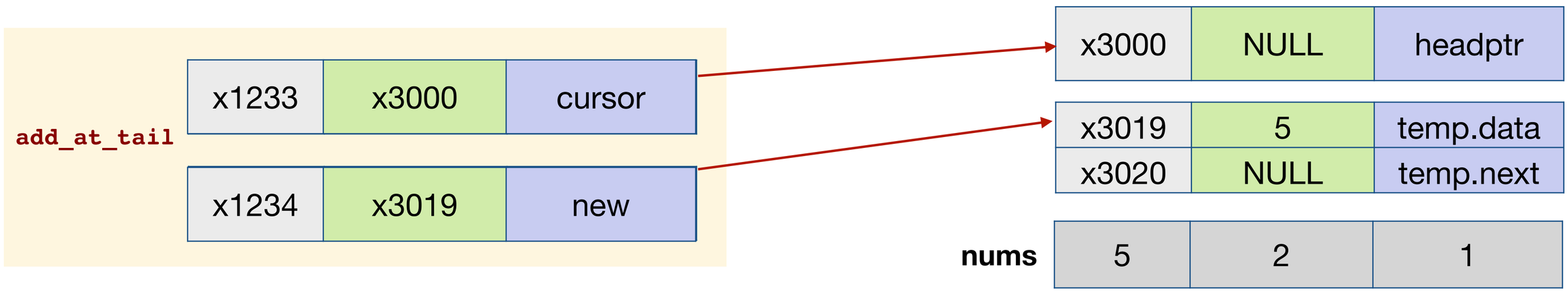
Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&headptr, &temp);
```



```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

add_at_tail

x1233	x3000	cursor

x1234	x3019	new

x3000	NULL	headptr

x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums

5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

x3000	NULL	headptr
x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums

5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

x3000	NULL	headptr
x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums

5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

add_at_head

x2019	x3000	cursor
x2020	x3019	new

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

`Function call : add_at_head(cursor, new);`

x3000	NULL	headptr

x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017		temp.data
x2018		temp.next

add_at_head

x2019	x3000	cursor
x2020	x3019	new

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	NULL	headptr

x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	temp.data
x2018	NULL	temp.next

add_at_head

x2019	x3000	cursor
x2020	x3019	new

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	NULL	headptr

x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	temp.data
x2018	NULL	temp.next

add_at_head

x2019	x3000	cursor
x2020	x3019	new

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

`Function call : add_at_head(cursor, new);`

x3000	NULL	headptr

x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	temp.data
x2018	NULL	temp.next

add_at_head

x2019	x3000	cursor
x2020	x3019	new

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	**x2017**	headptr

x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	temp.data
x2018	NULL	temp.next

add_at_head

x2019	x3000	cursor
x2020	x3019	new

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	**x2017**	headptr

x3019	5	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
  }
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

x3000	x2017	headptr

x3019	5	temp.data
x3020	NULL	temp.next

nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
}
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

x3000	x2017	headptr

x3019	5	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
  }
```

x3000	x2017	headptr

x3019	5	temp.data
x3020	NULL	temp.next

nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
  }
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
int main(void){
  int nums[] = {5, 2, 1};
  int total = 3;
  node temp;
  node *headptr = NULL;

  for (int i = 0; i < total; i++) {
    temp.data = nums[i];
    temp.next = NULL;
    add_at_tail(&headptr, &temp);
  }
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

add_at_tail

x1233		cursor
x1234		new

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums

5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
Function call : add_at_tail(&headptr, &temp);
```

add_at_tail

x1233		cursor
x1234		new

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums

5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

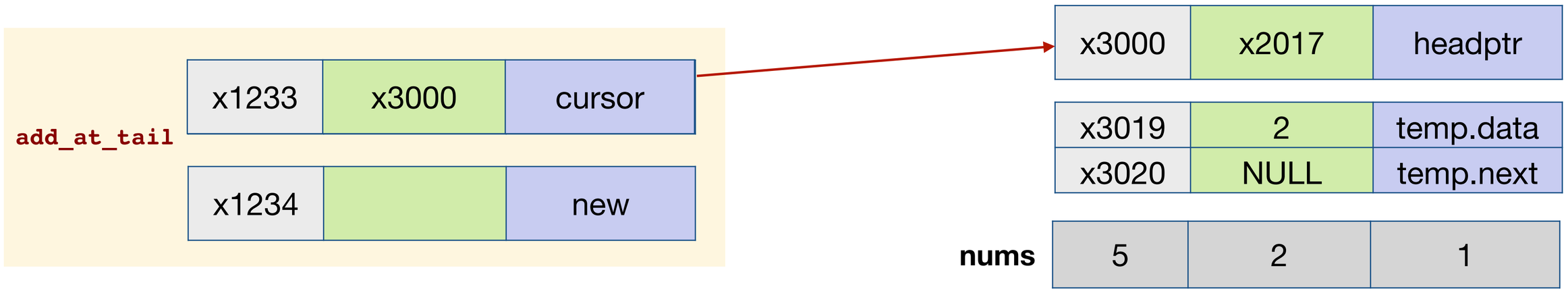
```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&headptr, &temp);
```



```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&headptr, &temp);
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

x3000	x2017	headptr
x3019	2	temp.data
x3020	NULL	temp.next

nums

5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

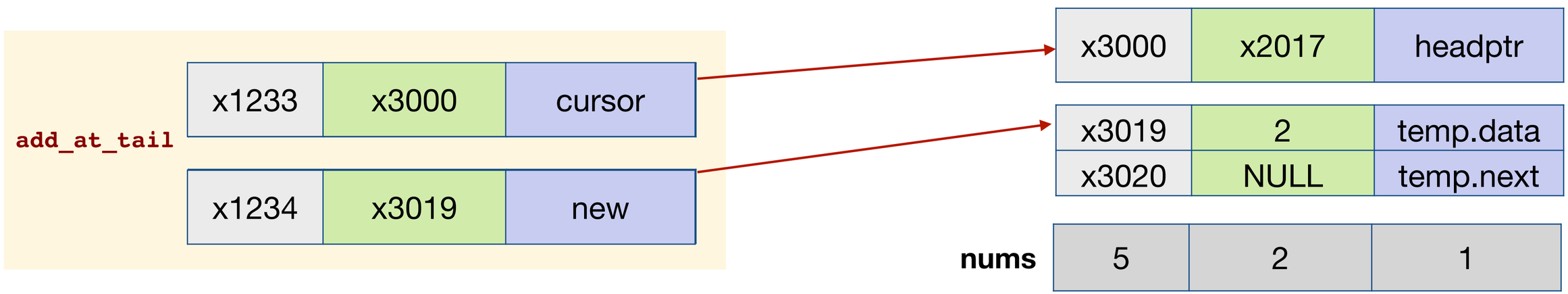
Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&headptr, &temp);
```



```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

Function call : **add_at_tail(&headptr, &temp);**

add_at_tail

x1233	x3000	cursor
x1234	x3019	new

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums

5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

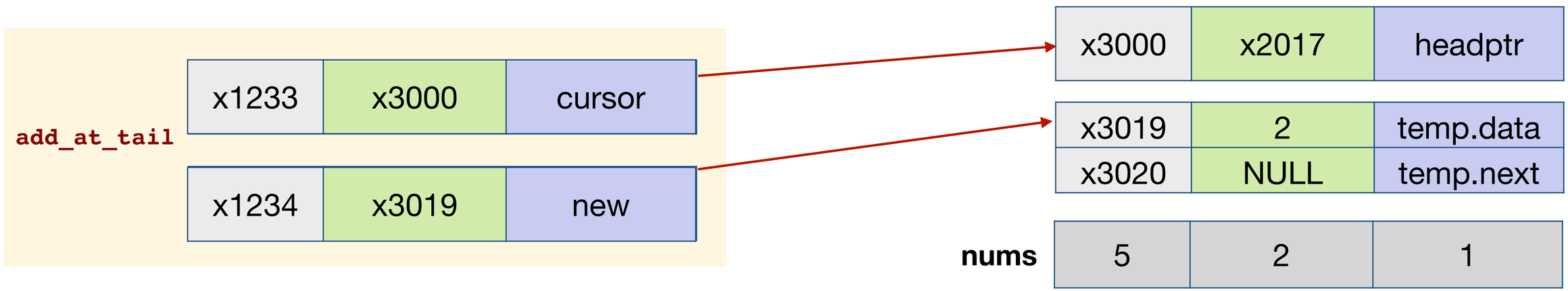
Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

Function call : **add_at_tail(&headptr, &temp);**



```
typedef struct node {
  int data;
  struct node *next;
} node;
```

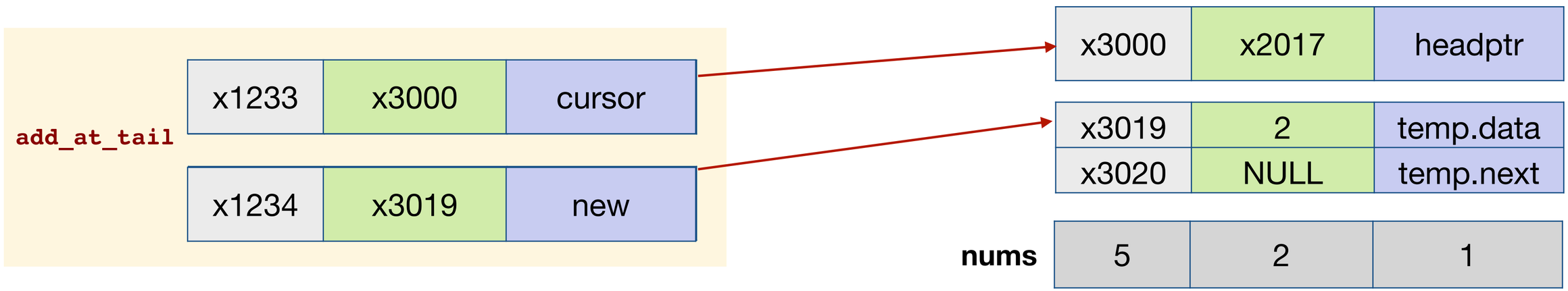
Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&(*cursor)->next,
                            &temp);
```



Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&(*cursor)->next,
                            &temp);
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

x3000	x2017	headptr
x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&(*cursor)->next,
                            &temp);
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&(*cursor)->next,
                            &temp);
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums

5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

```
void add_at_tail(node **cursor, node *new{
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
Function call : add_at_tail(&(*cursor)->next,
                            &temp);
```

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums

5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_tail

x1133	x2018	cursor

x1134	x3019	new

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums

5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums

5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_head

xFF10		cursor
xFF11		new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_head

xFF10		cursor
xFF11		new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

`Function call : add_at_head(cursor, new);`

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_head

xFF10		cursor
xFF11	x3019	new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

`Function call : add_at_head(cursor, new);`

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_head

xFF10		cursor
xFF11	x3019	new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums

5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017		temp.data
x4018		temp.next

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_head

xFF10		cursor
xFF11	**x3019**	new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017		temp.data
x4018		temp.next

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_head

xFF10		cursor
xFF11	**x3019**	new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_head

xFF10		cursor
xFF11	**x3019**	new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	NULL	~~temp~~.next

add_at_head

xFF10		cursor
xFF11	**x3019**	new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	NULL	temp.next

x2017	5	~~temp~~.data
x2018	x4017	~~temp~~.next

add_at_head

xFF10		cursor
xFF11	x3019	new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	NULL	temp.next

x2017	5	~~temp~~.data
x2018	x4017	~~temp~~.next

add_at_head

xFF10	x2018	cursor
xFF11	x3019	new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Function call : **add_at_head(cursor, new);**

x3000	x2017	headptr

x3019	2	~~temp~~.data
x3020	NULL	~~temp~~.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	NULL	temp.next

x2017	5	~~temp~~.data
x2018	x4017	~~temp~~.next

add_at_head

xFF10	x2018	cursor
xFF11		new

add_at_tail

x1133	x2018	cursor
x1134	x3019	new

```
void add_at_head(node **cursor, node *new) {
  node *temp = malloc(sizeof(node));
  temp->data = new->data;
  temp->next = new->next;

  if (*cursor == NULL) {
    *cursor = temp;
  } else {
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

End Result

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

End Result

x2017	x2018
Data	5
Next	x4017

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

End Result

x2017	x2018
Data	5
Next	x4017

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

End Result

x2017	x2018
Data	5
Next	x4017

x4017	x4018
Data	2
Next	NULL

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

Memory addresses are for illustration — it may not match LC3 CC or RTS.

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

We showed:

End Result

x2017	x2018
Data	5
Next	x4017

x4017	x4018
Data	2
Next	NULL

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

We showed:

- ***One way*** to add items at the tail position:

End Result

x2017	x2018
Data	5
Next	x4017

x4017	x4018
Data	2
Next	NULL

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

End Result

x2017	x2018
Data	5
Next	x4017

x4017	x4018
Data	2
Next	NULL

We showed:

- ***One way*** to add items at the tail position:
  - In the process, we also examined how an element gets added at the head.

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

```
typedef struct node {
  int data;
  struct node *next;
} node;
```

Memory addresses are for illustration — it may not match LC3 CC or RTS.

Review add_at_tail

x4017	2	temp.data
x4018	**NULL**	temp.next

x2017	5	~~temp~~.data
x2018	**x4017**	~~temp~~.next

End Result

x2017	x2018
Data	5
Next	x4017

x4017	x4018
Data	2
Next	NULL

We showed:

- ***One way*** to add items at the tail position:
  - In the process, we also examined how an element gets added at the head.
- Practice practice practice!

x3000	x2017	headptr

x3019	2	temp.data
x3020	NULL	temp.next

nums	5	2	1

Stack using linked lists

Push

Stack using linked lists

Push

Stack using linked lists

- First item in is the last item out - FILO

Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop

Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head

Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head

Stack using linked lists

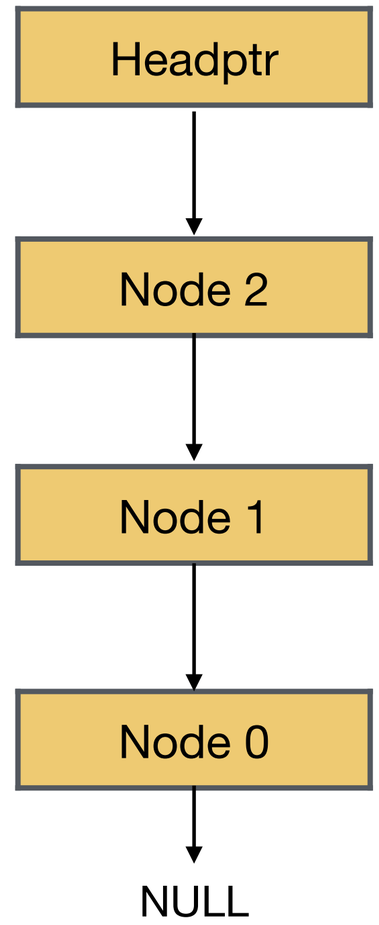
- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head

Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller

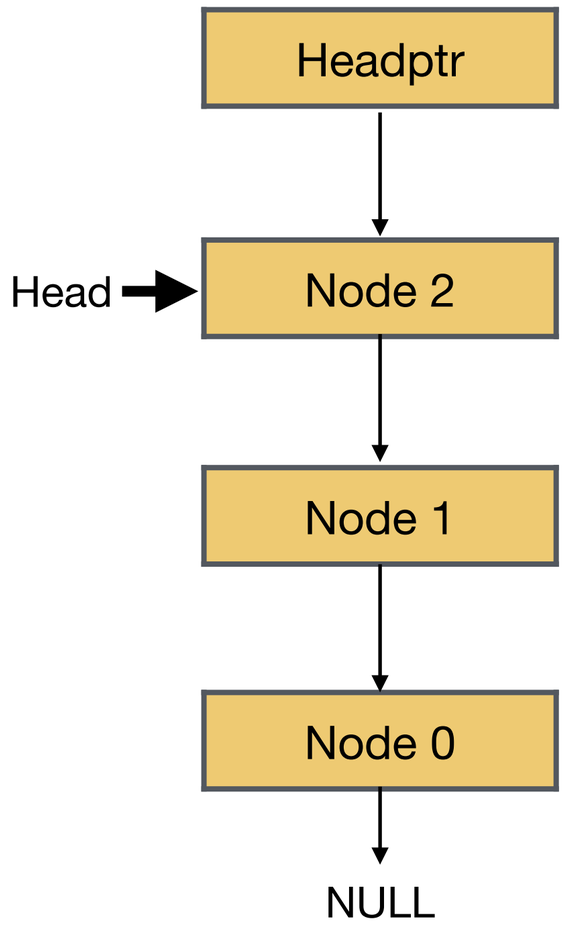
Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



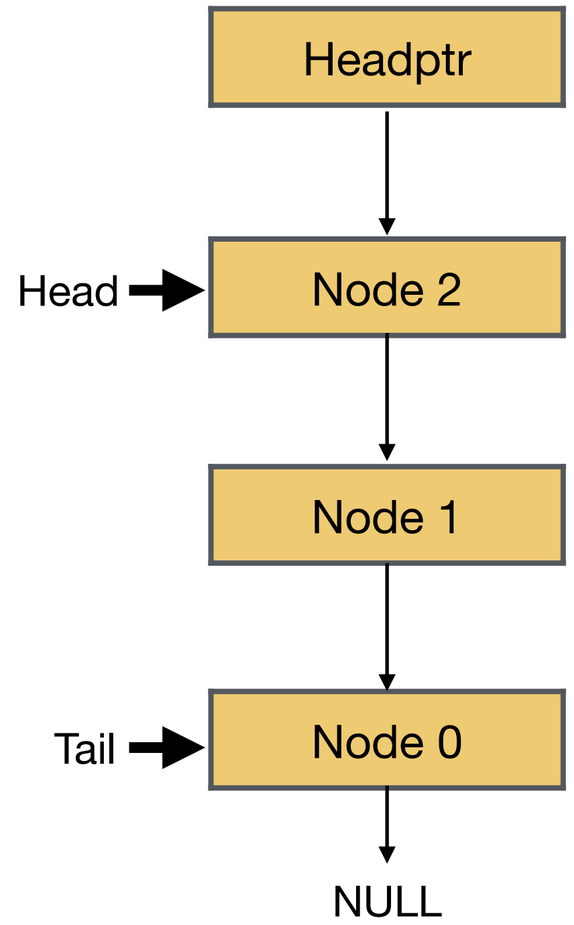
Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller

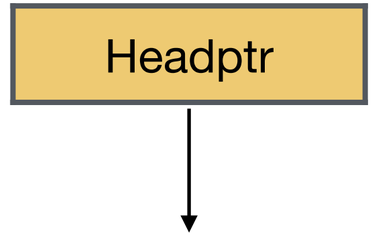


Stack using linked lists

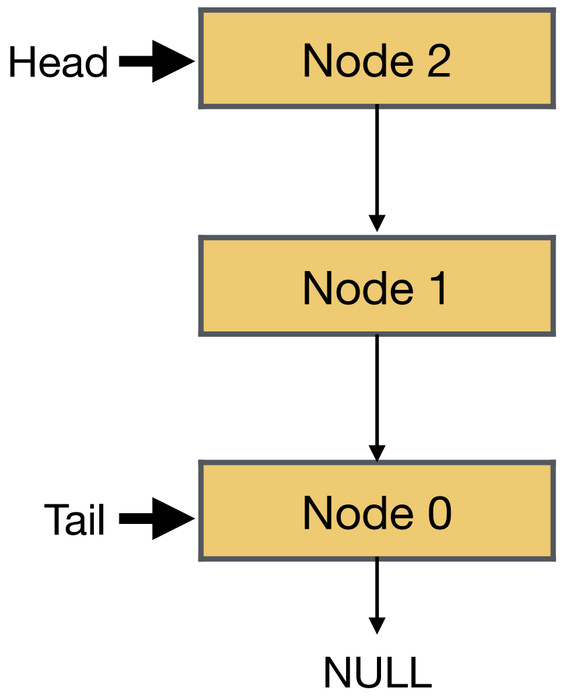
- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



Stack using linked lists

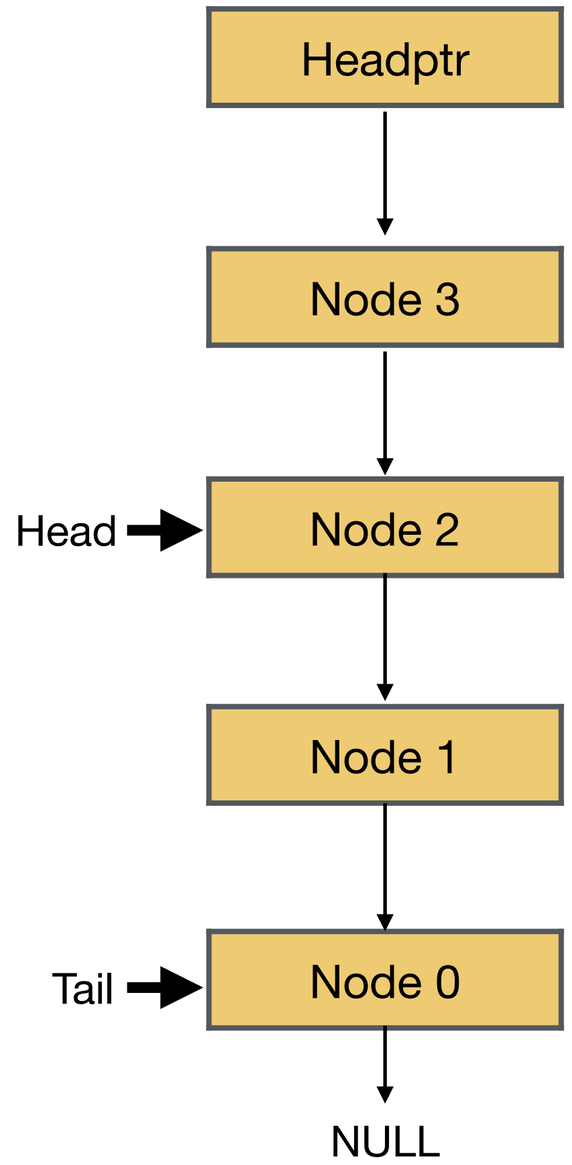


- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



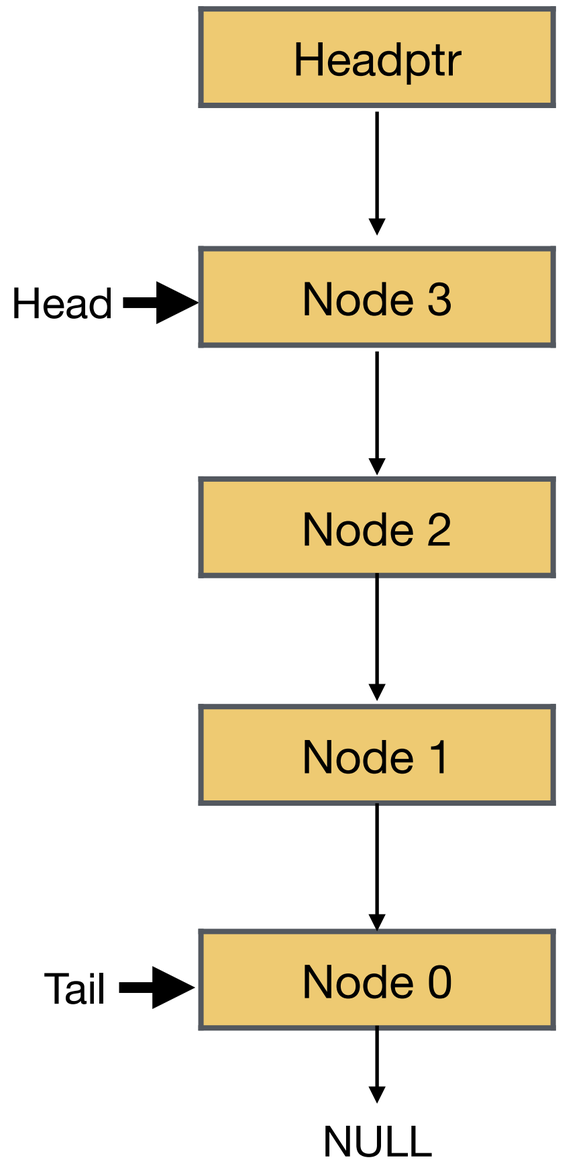
Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



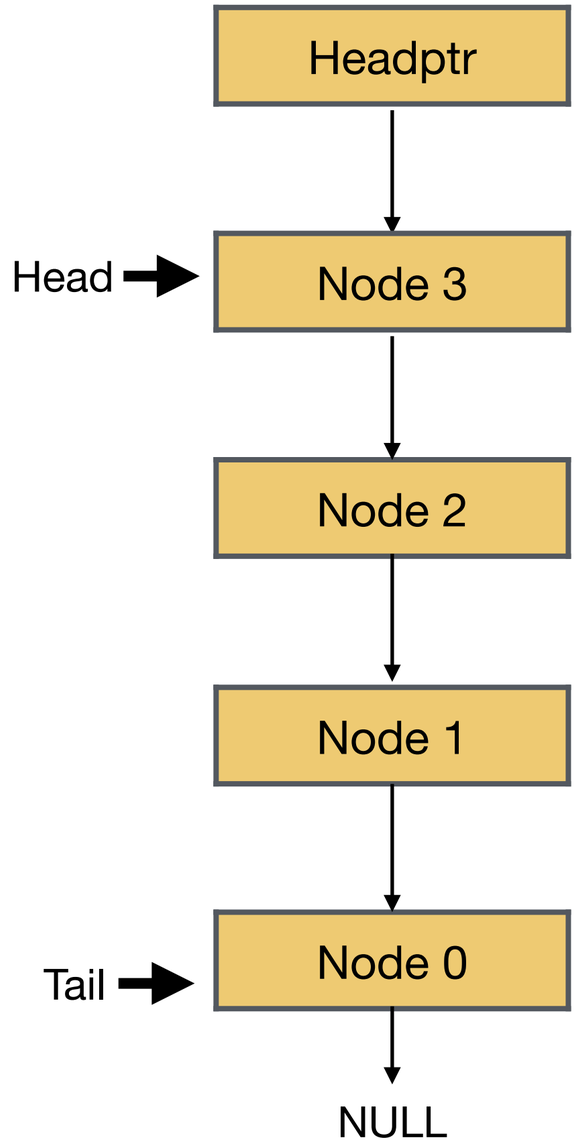
Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



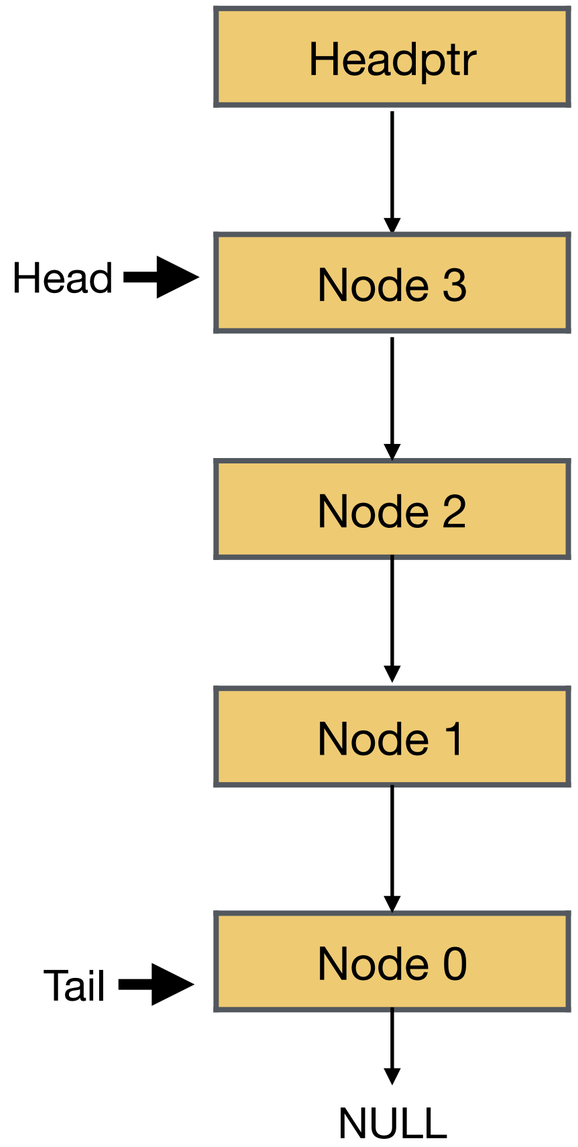
Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



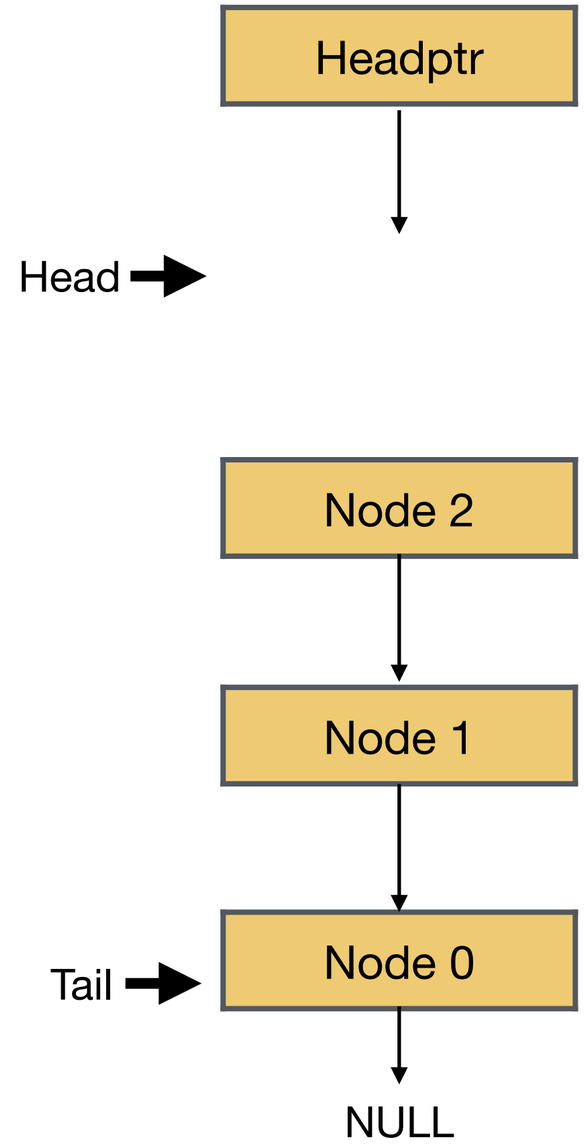
Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



Pop

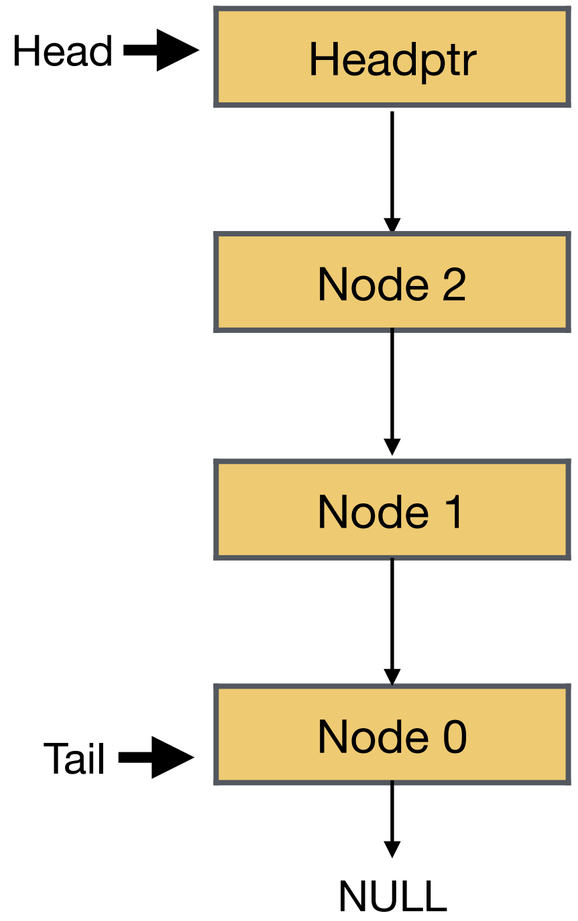
Stack using linked lists



- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller

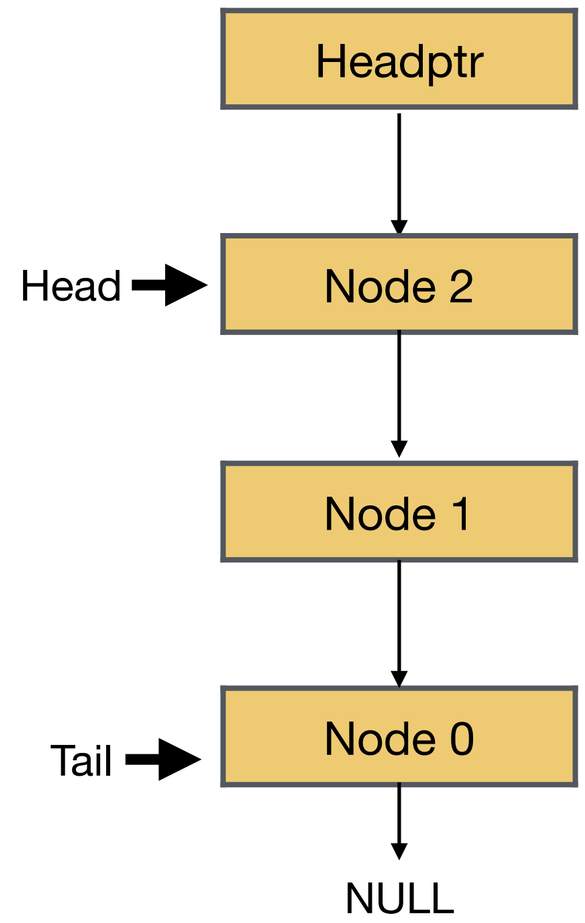
Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



Stack using linked lists

- First item in is the last item out - FILO
- Two operations for data movement: Push & Pop
- Stack top ~ head pointer/head
- Push ~ add at head
- Pop ~ remove from head
  - Need to give popped value to caller



Stack push using linked lists

```
void push(node **cursor, node *new){

  node* temp=(node*) malloc(sizeof(node));
  temp->name=new->name;
  temp->byear=new->byear;
  temp->next=new->next;




  }
}
```

Stack push using linked lists

- Suppose we want to **push a node onto stack.**

```
void push(node **cursor, node *new){

   node* temp=(node*) malloc(sizeof(node));
   temp->name=new->name;
   temp->byear=new->byear;
   temp->next=new->next;









   }
}
```

Stack push using linked lists

- Suppose we want to **push a node onto stack.**
- What needs to be done?

```
void push(node **cursor, node *new){

  node* temp=(node*) malloc(sizeof(node));
  temp->name=new->name;
  temp->byear=new->byear;
  temp->next=new->next;









  }
}
```

Stack push using linked lists

- Suppose we want to **push a node onto stack.**
- What needs to be done?
  - Deal with empty list

```
void push(node **cursor, node *new){

   node* temp=(node*) malloc(sizeof(node));
   temp->name=new->name;
   temp->byear=new->byear;
   temp->next=new->next;

   if (cursor == NULL)
       *cursor = temp;



   }
}
```

Stack push using linked lists

- Suppose we want to **push a node onto stack.**
- What needs to be done?
  - Deal with empty list
  - New node should point to current head.

```
void push(node **cursor, node *new){

  node* temp=(node*) malloc(sizeof(node));
  temp->name=new->name;
  temp->byear=new->byear;
  temp->next=new->next;

  if (cursor == NULL)
      *cursor = temp;
  else{
    temp->next = *cursor;

  }
}
```

Stack push using linked lists

- Suppose we want to **push a node onto stack.**
- What needs to be done?
  - Deal with empty list
  - New node should point to current head.
  - Current head should be updated to new node.

```
void push(node **cursor, node *new){

  node* temp=(node*) malloc(sizeof(node));
  temp->name=new->name;
  temp->byear=new->byear;
  temp->next=new->next;

  if (cursor == NULL)
      *cursor = temp;
  else{
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Stack push using linked lists

- Suppose we want to **push a node onto stack.**
- What needs to be done?
  - Deal with empty list
  - New node should point to current head.
  - Current head should be updated to new node.

Same as insert at head!

```
void push(node **cursor, node *new){

  node* temp=(node*) malloc(sizeof(node));
  temp->name=new->name;
  temp->byear=new->byear;
  temp->next=new->next;

  if (cursor == NULL)
      *cursor = temp;
  else{
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Stack pop using linked lists

- To **pop** a node from stack, we have to delete node from head:

***Similar* to delete at head**

```
node * pop(node **headptr){
```

```
}
```

Stack pop using linked lists

- To **pop** a node from stack, we have to delete node from head:
  - Deal with empty list

***Similar* to delete at head**

```
node * pop(node **headptr){
  if (*headptr==NULL)
    return NULL;
```

```
}
```

Stack pop using linked lists

- To **pop** a node from stack, we have to delete node from head:
  - Deal with empty list
  - Make a *copies* of the head pointer

Similar to delete at head

```
node * pop(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
  node * new=(node*) malloc(sizeof(node));
  new->name=(*headptr)->name;
  new->byear=(*headptr)->byear;
  new->next = NULL;

  node *old_head = *headptr;



}
```

Stack pop using linked lists

- To **pop** a node from stack, we have to delete node from head:
  - Deal with empty list
  - Make a *copies* of the head pointer
  - Shift the head pointer to its next item

Similar to delete at head

```
node * pop(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
  node * new=(node*) malloc(sizeof(node));
  new->name=(*headptr)->name;
  new->byear=(*headptr)->byear;
  new->next = NULL;

  node *old_head = *headptr;
  *headptr = (*headptr)->next;


}
```

Stack pop using linked lists

- To **pop** a node from stack, we have to delete node from head:
  - Deal with empty list
  - Make a *copies* of the head pointer
  - Shift the head pointer to its next item
  - Call `free` on a copy of the head pointer

Similar **to delete at head**

```
node * pop(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
  node * new=(node*) malloc(sizeof(node));
  new->name=(*headptr)->name;
  new->byear=(*headptr)->byear;
  new->next = NULL;

  node *old_head = *headptr;
  *headptr = (*headptr)->next;
  free(old_head);


}
```

Stack pop using linked lists

- To **pop** a node from stack, we have to delete node from head:
  - Deal with empty list
  - Make a *copies* of the head pointer
  - Shift the head pointer to its next item
  - Call `free` on a copy of the head pointer
  - ***Return the popped copy to caller***

***Similar* to delete at head**

```
node * pop(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
    node * new=(node*) malloc(sizeof(node));
    new->name=(*headptr)->name;
    new->byear=(*headptr)->byear;
    new->next = NULL;

    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);

    return new;
  }
}
```

Queue using linked lists

Queue using linked lists

- First item in is the first item out - FIFO

Queue using linked lists

- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue

Queue using linked lists

- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue
  - Dequeued item must be available for use by caller

Queue using linked lists

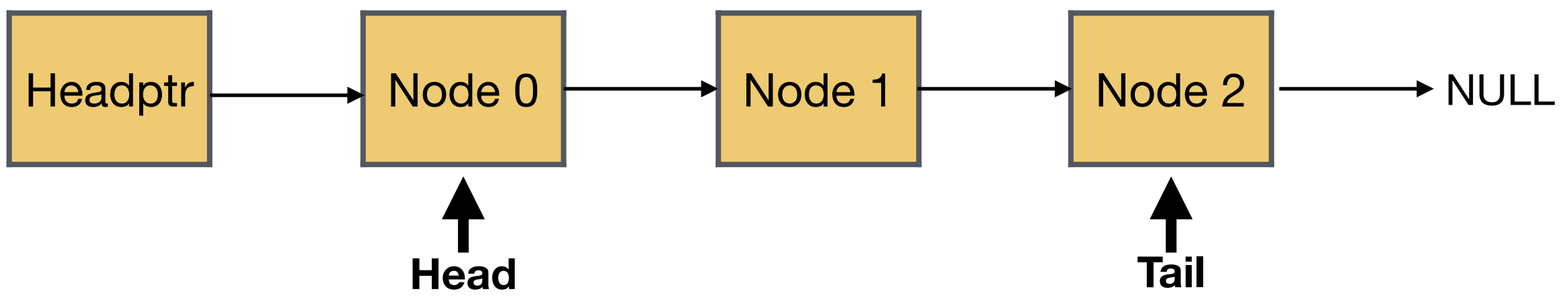
- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue
  - Dequeued item must be available for use by caller

Enqueue

Queue using linked lists

- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue
  - Dequeued item must be available for use by caller

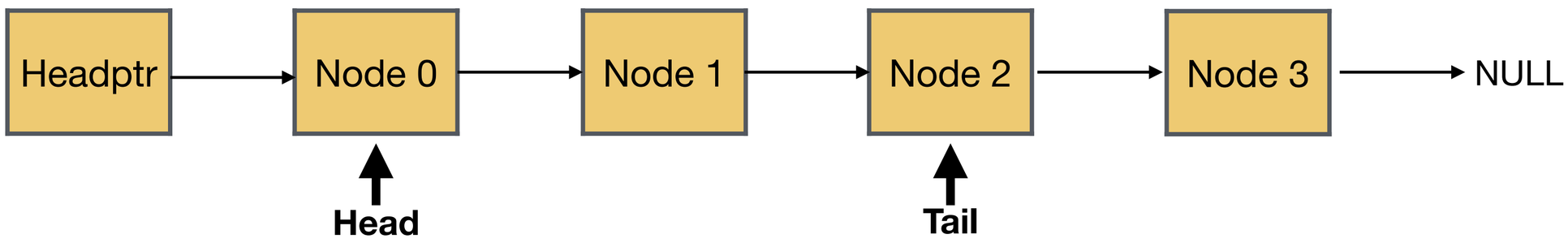
Enqueue



Queue using linked lists

- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue
  - Dequeued item must be available for use by caller

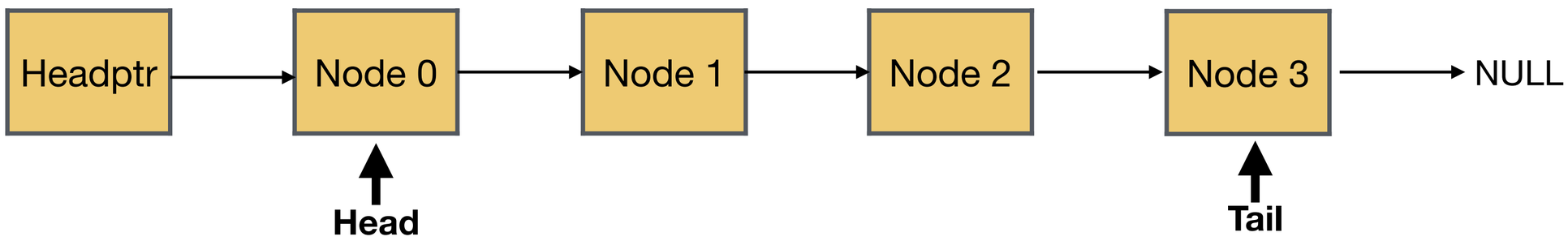
Enqueue



Queue using linked lists

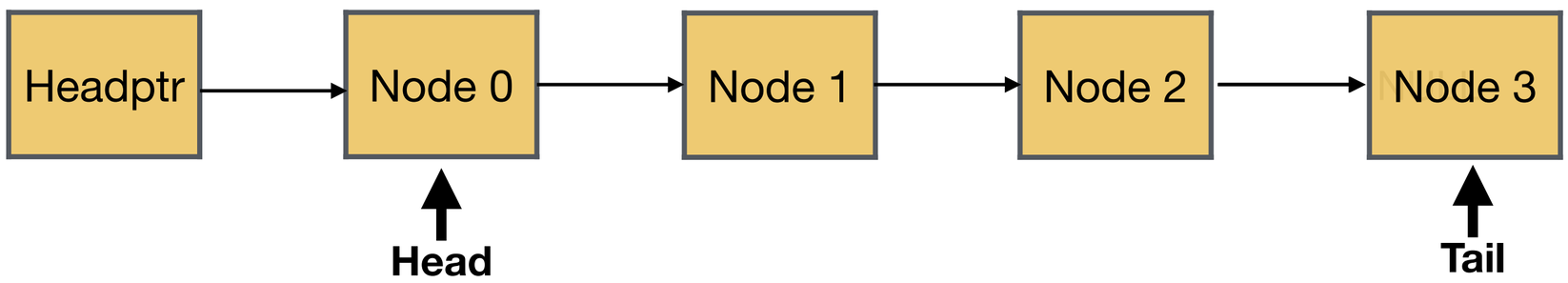
- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue
  - Dequeued item must be available for use by caller

Enqueue



Queue using linked lists

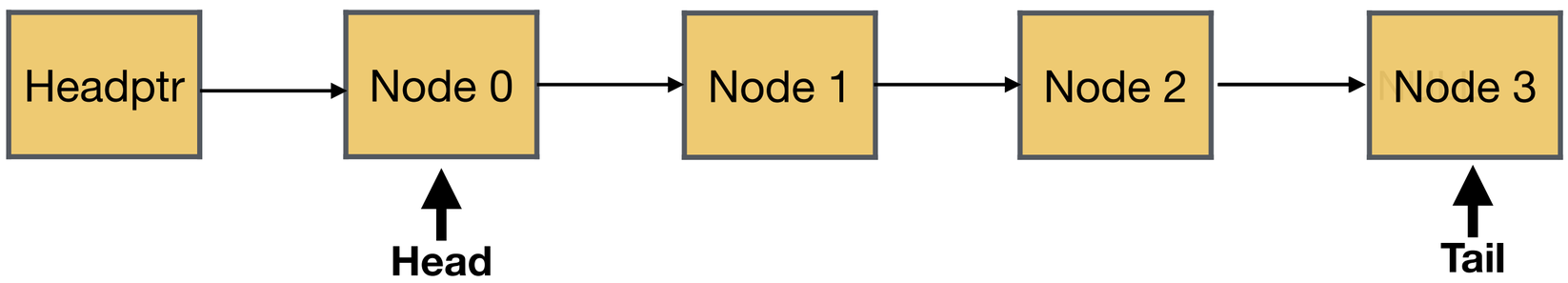
- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue
  - Dequeued item must be available for use by caller



Queue using linked lists

- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue
  - Dequeued item must be available for use by caller

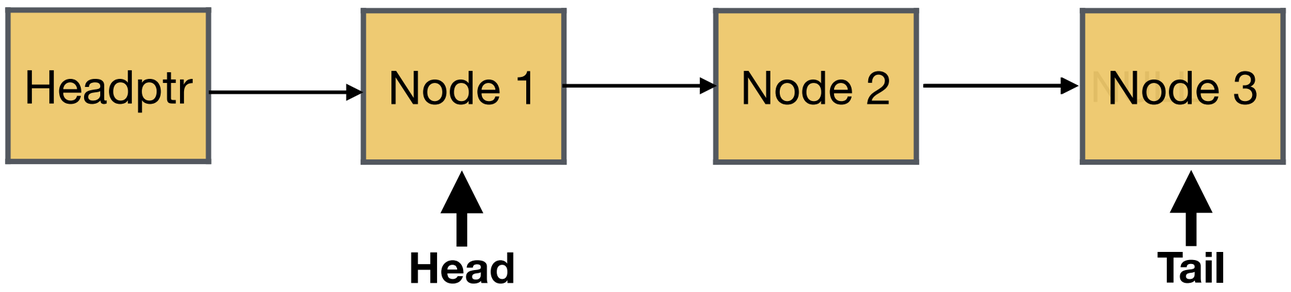
Dequeue



Queue using linked lists

- First item in is the first item out - FIFO
- Two operations for data movement: enqueue & dequeue
  - Dequeued item must be available for use by caller

Dequeue



Enqueue using linked lists

- To add (enqueue) a **node** to a queue

```
void enqueue(node **cursor, node *new){

}
```

Enqueue using linked lists

- To add (enqueue) a **node** to a queue
- If queue empty, add right away, else

```
void enqueue(node **cursor, node *new){
  if (*cursor == NULL){
    node * temp = (node *) malloc(sizeof(node));
    temp->name = new->name;
    temp->byear = new->byear;
    temp->next = new->next;
    *cursor = temp;
  }


}
```

Enqueue using linked lists

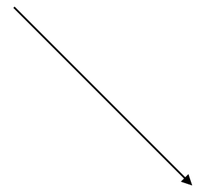
- To add (enqueue) a **node** to a queue
  - If queue empty, add right away, else
  - Go till the end

```
void enqueue(node **cursor, node *new){
  if (*cursor == NULL){
    node * temp = (node *) malloc(sizeof(node));
    temp->name = new->name;
    temp->byear = new->byear;
    temp->next = new->next;
    *cursor = temp;
  }
  else
    enqueue(&(*cursor)->next, new);
}
```

Enqueue using linked lists

- To add (enqueue) a **node** to a queue
  - If queue empty, add right away, else
  - Go till the end

```
void enqueue(node **cursor, node *new){
  if (*cursor == NULL){
    node * temp = (node *) malloc(sizeof(node));
    temp->name = new->name;
    temp->byear = new->byear;
    temp->next = new->next;
    *cursor = temp;
  }
  else
    enqueue(&(*cursor)->next, new);
}
```



Same as insert at tail

Dequeue using linked lists

```
node * dequeue(node **headptr){

}
```

Dequeue using linked lists

- To delete (dequeue) a **node** from the queue

```
node * dequeue(node **headptr){
```

```
}
```

Dequeue using linked lists

- To delete (dequeue) a **node** from the queue
  - If head empty do nothing, else,

```
node * dequeue(node **headptr){
  if (*headptr==NULL)
    return NULL;

}
```

Dequeue using linked lists

- To delete (dequeue) a **node** from the queue
  - If head empty do nothing, else,
  - Save copy of current head

```
node * dequeue(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
  node* new=(node*) malloc(sizeof(node));
  new->name=(*headptr)->name;
  new->byear=(*headptr)->byear;

  node *old_head = *headptr;



}
```

Dequeue using linked lists

- To delete (dequeue) a **node** from the queue
  - If head empty do nothing, else,
  - Save copy of current head
  - Advance head pointer and free the memory used by old head

```
node * dequeue(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
  node* new=(node*) malloc(sizeof(node));
  new->name=(*headptr)->name;
  new->byear=(*headptr)->byear;

  node *old_head = *headptr;
  *headptr = (*headptr)->next;
  free(old_head);


}
```

Dequeue using linked lists

- To delete (dequeue) a **node** from the queue
  - If head empty do nothing, else,
  - Save copy of current head
  - Advance head pointer and free the memory used by old head
  - Pass/return dequeued item to caller

```
node * dequeue(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
    node* new=(node*) malloc(sizeof(node));
    new->name=(*headptr)->name;
    new->byear=(*headptr)->byear;

    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);

    return new;
  }
}
```

Dequeue using linked lists

- To delete (dequeue) a **node** from the queue
  - If head empty do nothing, else,
  - Save copy of current head
  - Advance head pointer and free the memory used by old head
  - Pass/return dequeued item to caller

```
node * dequeue(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
    node* new=(node*) malloc(sizeof(node));
    new->name=(*headptr)->name;
    new->byear=(*headptr)->byear;

    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);

    return new;
  }
}
```

***Similar to* delete at head!**

Relations

Relations

Linked List	Stack	Queue
Add at head	Push	

```
void push(node **cursor, node *new){
  node* temp=(node*) malloc(sizeof(node));
  temp->name=new->name;
  temp->byear=new->byear;
  temp->next=new->next;

  if (cursor == NULL)
      *cursor = temp;
  else{
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

```
void add_at_head(node **cursor, node *new){

  node * temp = (node *) malloc(sizeof(node));
  temp->name = new->name;
  temp->next = new->next;
  temp->next=new->next;

  if (*cursor == NULL)
    *cursor = temp;
  else{
    temp->next = *cursor;
    *cursor = temp;
  }
}
```

Relations

Linked List	Stack	Queue
Add at head	Push	
Delete at head	Pop	Dequeue

```c
void del_head(node **headptr){
  if (*headptr==NULL)
    return;
  else{
    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);
  }
}
```

```c
node * dequeue(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
    node* new=(node*) malloc(sizeof(node));
    new->name=(*headptr)->name;
    new->byear=(*headptr)->byear;

    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);

    return new;
  }
}
```

```c
node * pop(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
    node * new=(node*)
malloc(sizeof(node));
    new->name=(*headptr)->name;
    new->byear=(*headptr)->byear;
    new->next = NULL;

    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);

    return new;
  }
}
```

Relations

Linked List	Stack	Queue
Add at head	Push	
Delete at head	Pop	Dequeue
Add at tail		Enqueue

```c
void del_head(node **headptr){
  if (*headptr==NULL)
    return;
  else{
    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);
  }
}
```

```c
node * dequeue(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
    node* new=(node*) malloc(sizeof(node));
    new->name=(*headptr)->name;
    new->byear=(*headptr)->byear;

    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);

    return new;
  }
}
```

```c
node * pop(node **headptr){
  if (*headptr==NULL)
    return NULL;
  else{
    node * new=(node*)
malloc(sizeof(node));
    new->name=(*headptr)->name;
    new->byear=(*headptr)->byear;
    new->next = NULL;

    node *old_head = *headptr;
    *headptr = (*headptr)->next;
    free(old_head);

    return new;
  }
}
```

Relations

Linked List	Stack	Queue
Add at head	Push	
Delete at head	Pop	Dequeue
Add at tail		Enqueue

```
void add_at_tail(node **cursor, node *new){
  if (*cursor == NULL)
    add_at_head(cursor, new);
  else
    add_at_tail(&(*cursor)->next, new);
}
```

```
void enqueue(node **cursor, node *new){
  if (*cursor == NULL){
    node * temp = (node *) malloc(sizeof(node));
    temp->name = new->name;
    temp->byear = new->byear;
    temp->next = new->next;
    *cursor = temp;
  }
  else
    enqueue(&(*cursor)->next, new);
}
```

Exercise(s)

- Given a *sorted* linked list, implement binary search on the list

```
node * binary_search(*headptr, char * key)
```

  - Return a NULL pointer if the element is not found
  - Otherwise return a pointer to the element.

- Hint: Write a function to get the middle element in a linked list

How do you find the middle element in a linked list?

Finding middle of a linked list

```c
#include <stdio.h>

int main(void){
  int i, target, j;
  printf("Enter a target number:\t");
  scanf("%d", &target);
  for (j=0, i=0; j<target; i++, j++)
    j++;
  printf("Midway to target is %d", i);
}
```

Exercise for “later”

- Given two ***sorted*** linked lists write a function that takes the two head pointers and returns a pointer to a ***merged*** list
- Sort order **must be maintained.** Basic idea …
  - Traverse both lists until one of them ends, then copy over the remaining list
  - During traversal add new nodes in sorted order