# ECE 220: Computer Systems & Programming

## Lecture 19: Linked List

Thomas Moon

March 21, 2024

```
typedef struct flightType{
        …
}Flight;

int main()
{
    Flight planes[100];
```

What if we need > 100 planes?

→ Increase the size of array.

But, sometimes we only need 2-3 planes

→ Wasting the rest of unused memory.

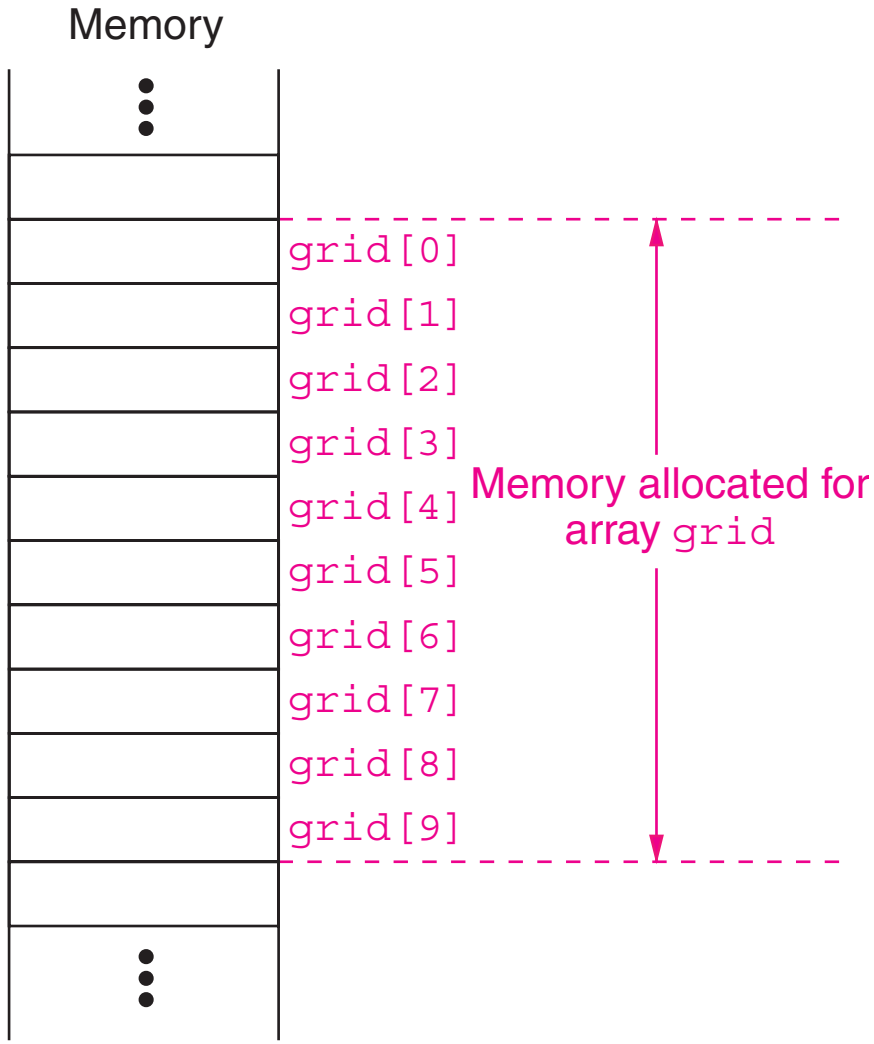
Dynamic Memory Allocation (this lecture)

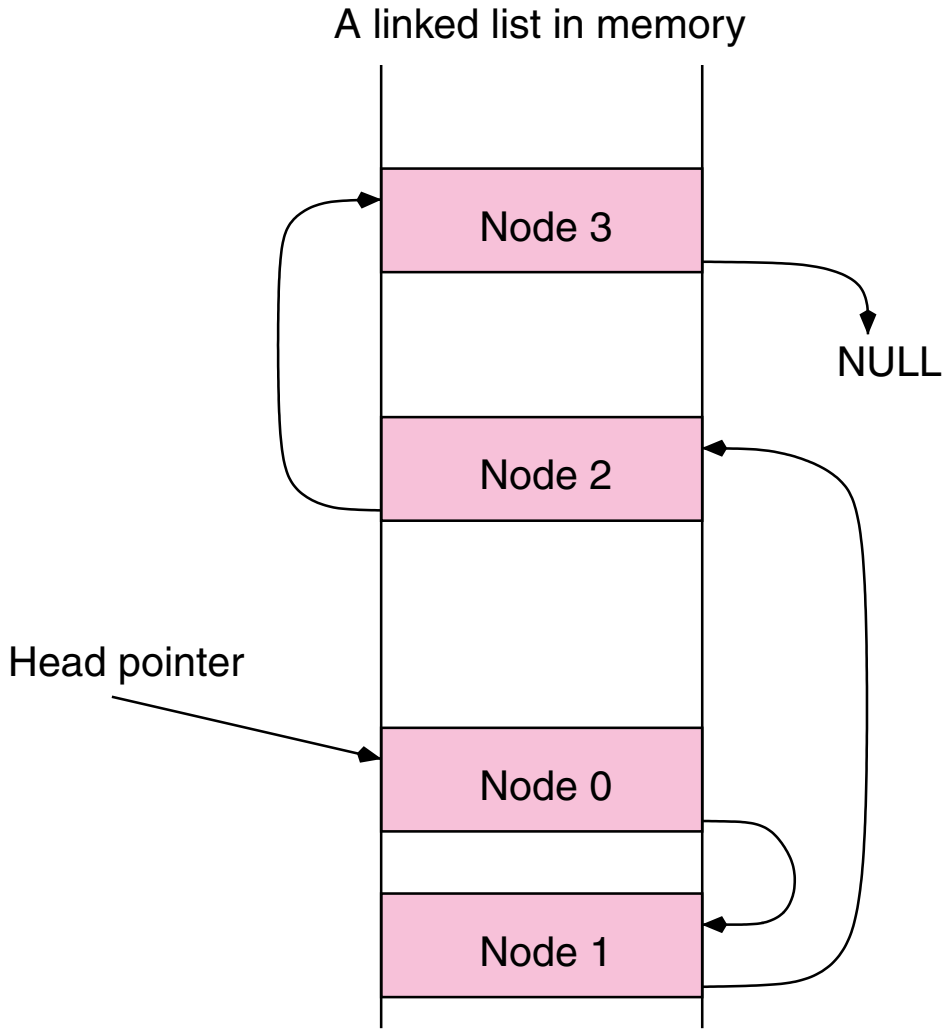
Planes take off & land,
i.e. the data coming in & going out.

Adding or removing an item in the middle.

→ Not efficient in array

Linked List (next lecture)



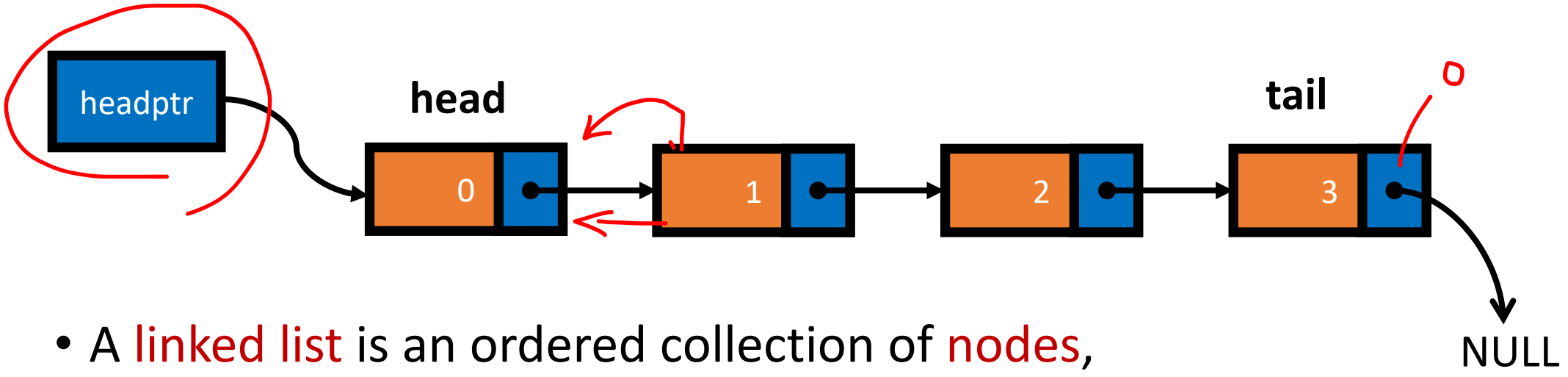

## Array

(can be automatic or dynamic)

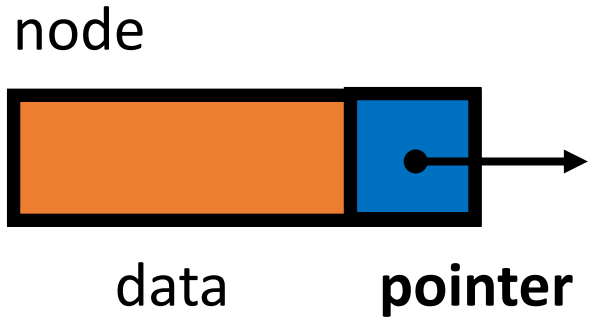
## Linked List

(dynamic only)

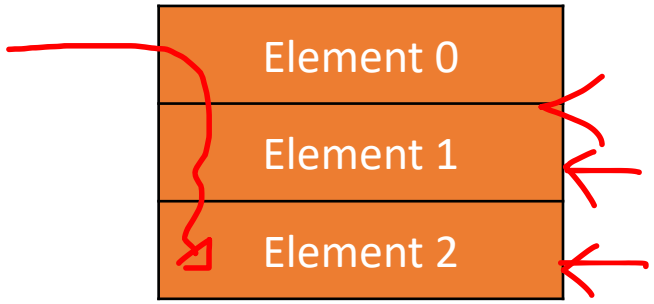
# Linked List

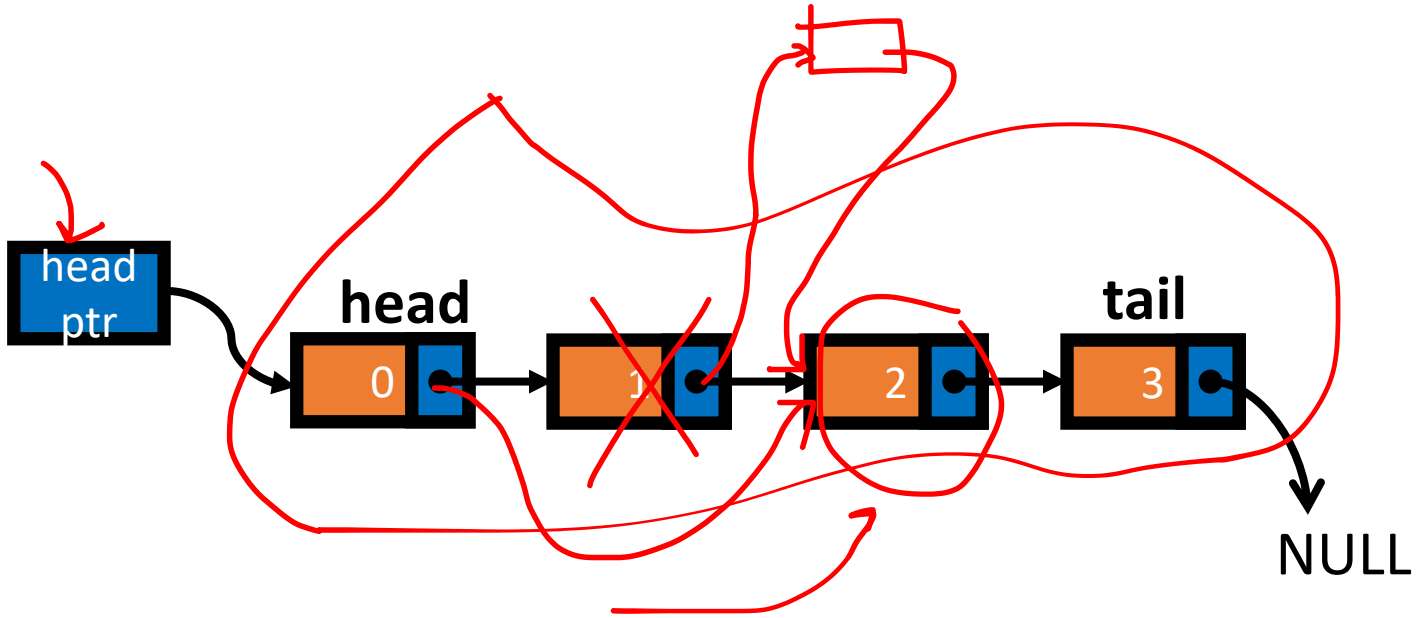



- A linked list is an ordered collection of nodes, each of which contains some data, connected using pointers.
  - The first node in the list is called the head.
  - The last node in the list is called the tail.
- Each node contains at least
  - a piece of data
  - pointer to the next node in the list



# Array vs. Linked List

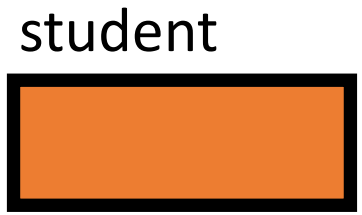


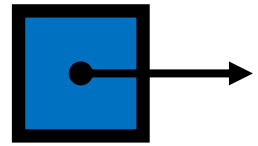




| | Array | Linked List |
|---|---|---|
| Memory Allocation | Automatic/Dynamic | Dynamic |
| Memory Structure | Contiguous | Not necessary consecutive |
| Order of Access | Random | Sequential |
| Insertion/Deletion | Create/delete space, then shift all successive elements | Change pointer address |

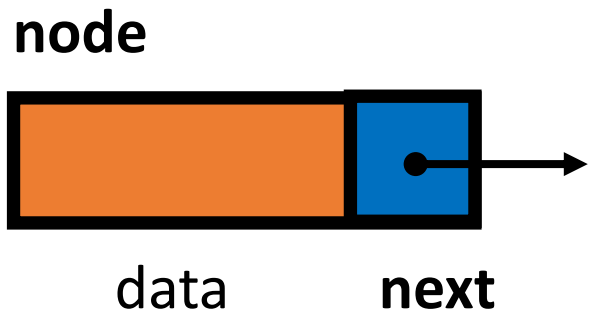
# Example: Student Record

```
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
}student;
```

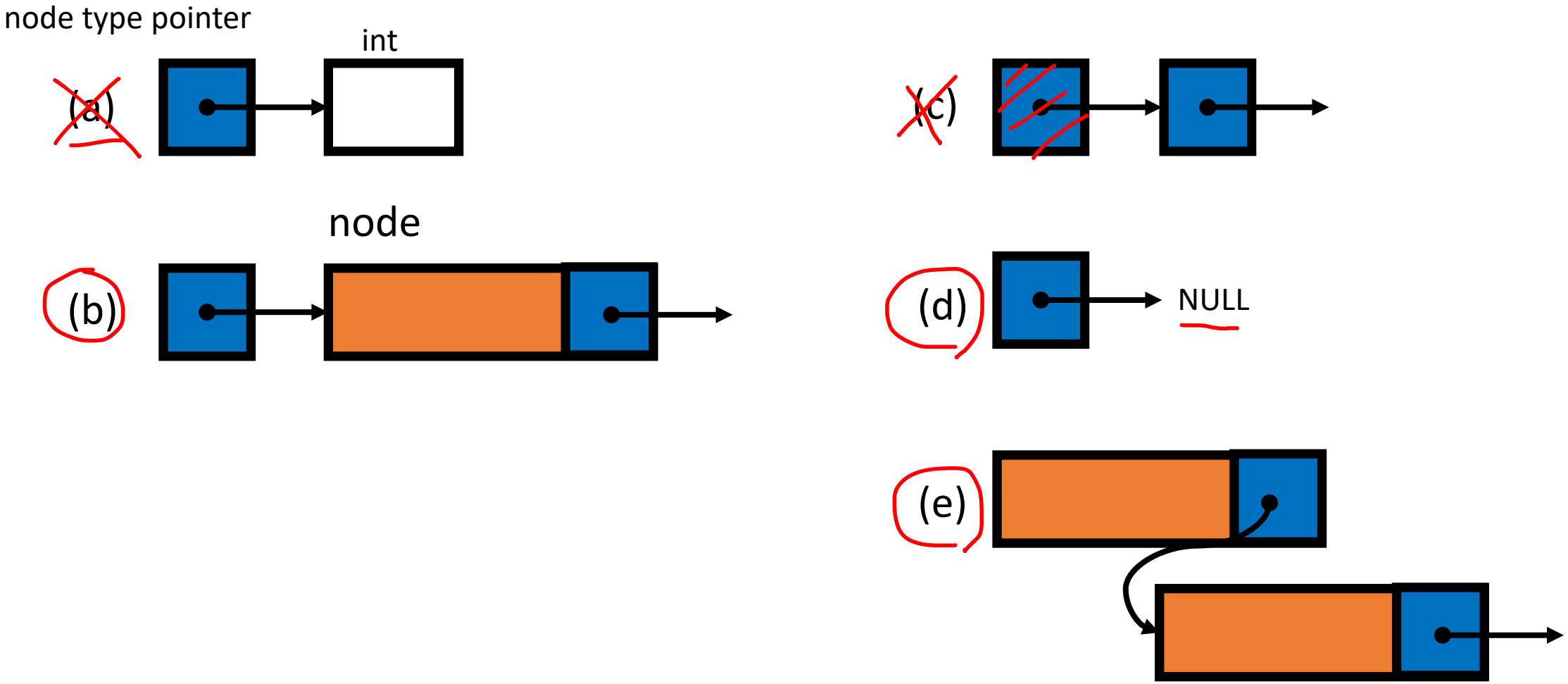


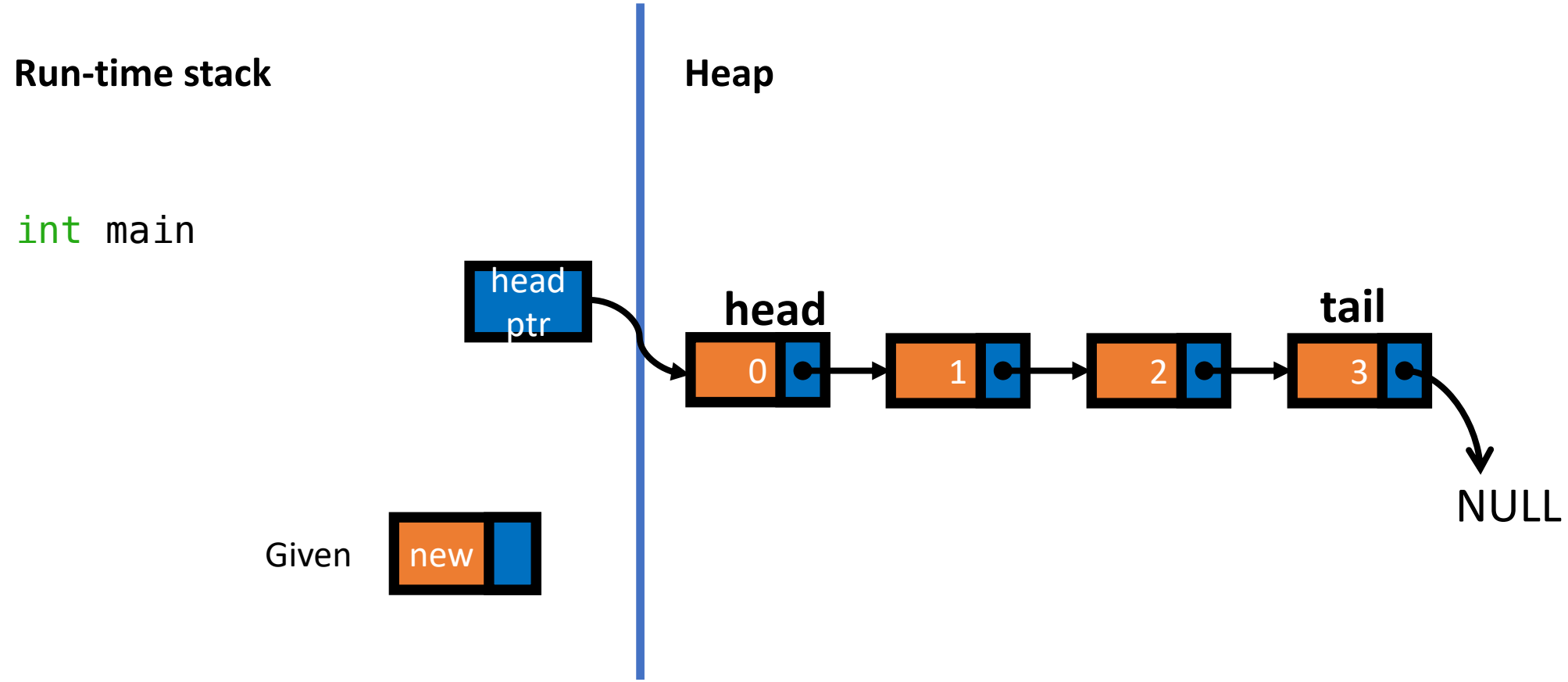

```
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
    struct StudentStruct *next;
}node;
```



# Which one is correct?



# Task1: Insert a new node at the head




Steps:

(1)Create a new node in Heap

(2)Insert the new node at the head

**The linked list may be empty or not.

```
void insert_head(????????);

int main(){
    node *headptr = NULL;

    node data;
    // First student
    data.UIN = 1;
       ...

    insert_head(????????);

    // Second student
    data.UIN = 2;
       ...

    insert_head(????????);

}
```

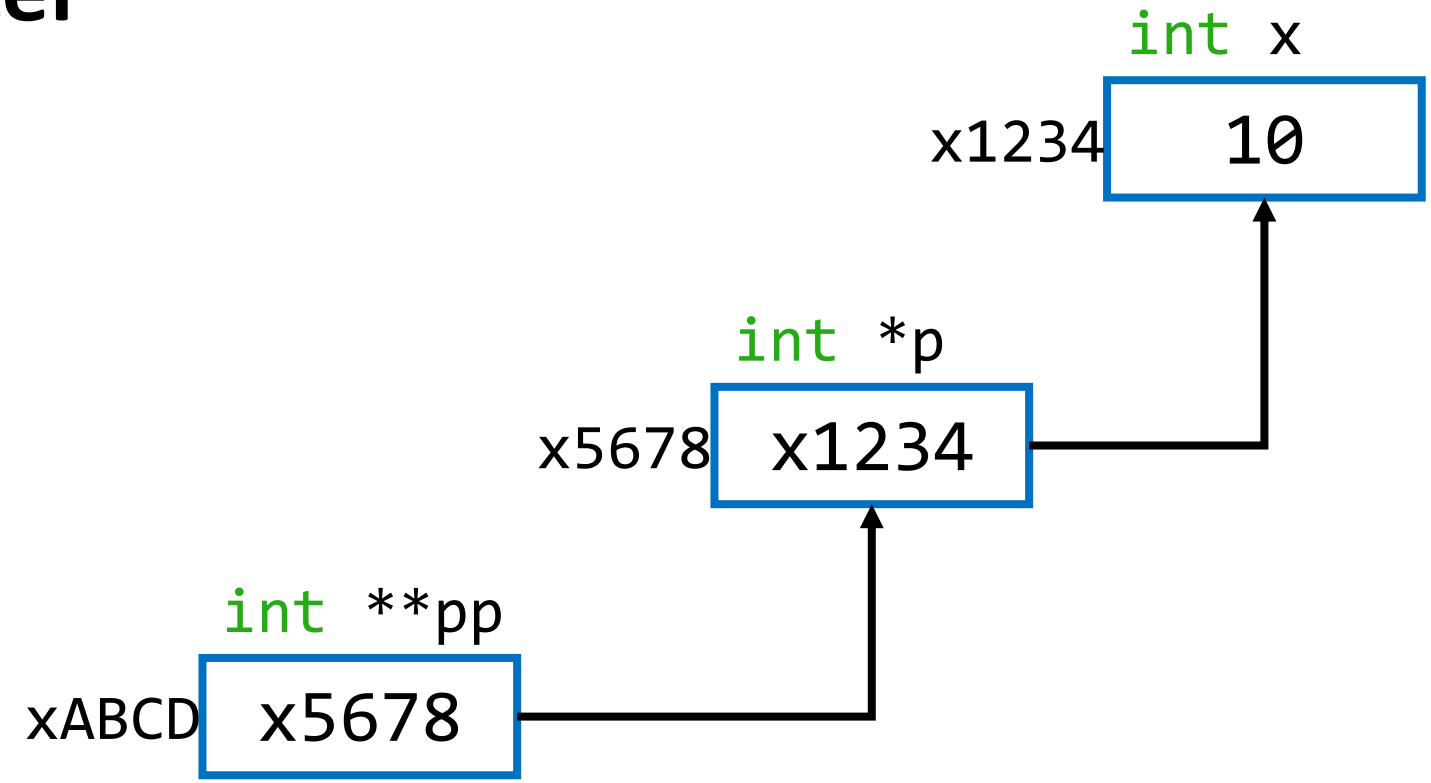
# Recall: Double Pointer

```
int x = 10;
int *p;

p = &x;

int **pp;
pp = &p;


printf(”x%X\n", &pp);
printf(”x%X\n", pp);
printf(”x%X\n", *pp);
printf(”%d\n", **pp);
```



# Recall: Solve Swap Problem

```c
void Swap(int firstVal, int secondVal);
int main()
{
    int valueA = 1;
    int valueB = 2;

    Swap(valueA, valueB);
}

void Swap(int firstVal, int secondVal)
{
    int tempVal;

    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
}
```

*call by value*

```c
void NewSwap(int *firstVal, int *secondVal);
int main()
{
    int valueA = 1;
    int valueB = 2;

    NewSwap(&valueA, &valueB);
}

void NewSwap(int *firstVal, int *secondVal)
{
    int tempVal;

    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
```

int *p;

*call by reference*

# Recall: Pass Structs as Arguments

A.

```
void print_flightName(Flight plane)
{
    printf("flight name: %s\n", plane.flightName);
}
```

vs

which one is more efficient?

B.

```
void print_flightName(Flight *plane)
{
    printf("flight name: %s\n", plane->flightName);
}
```

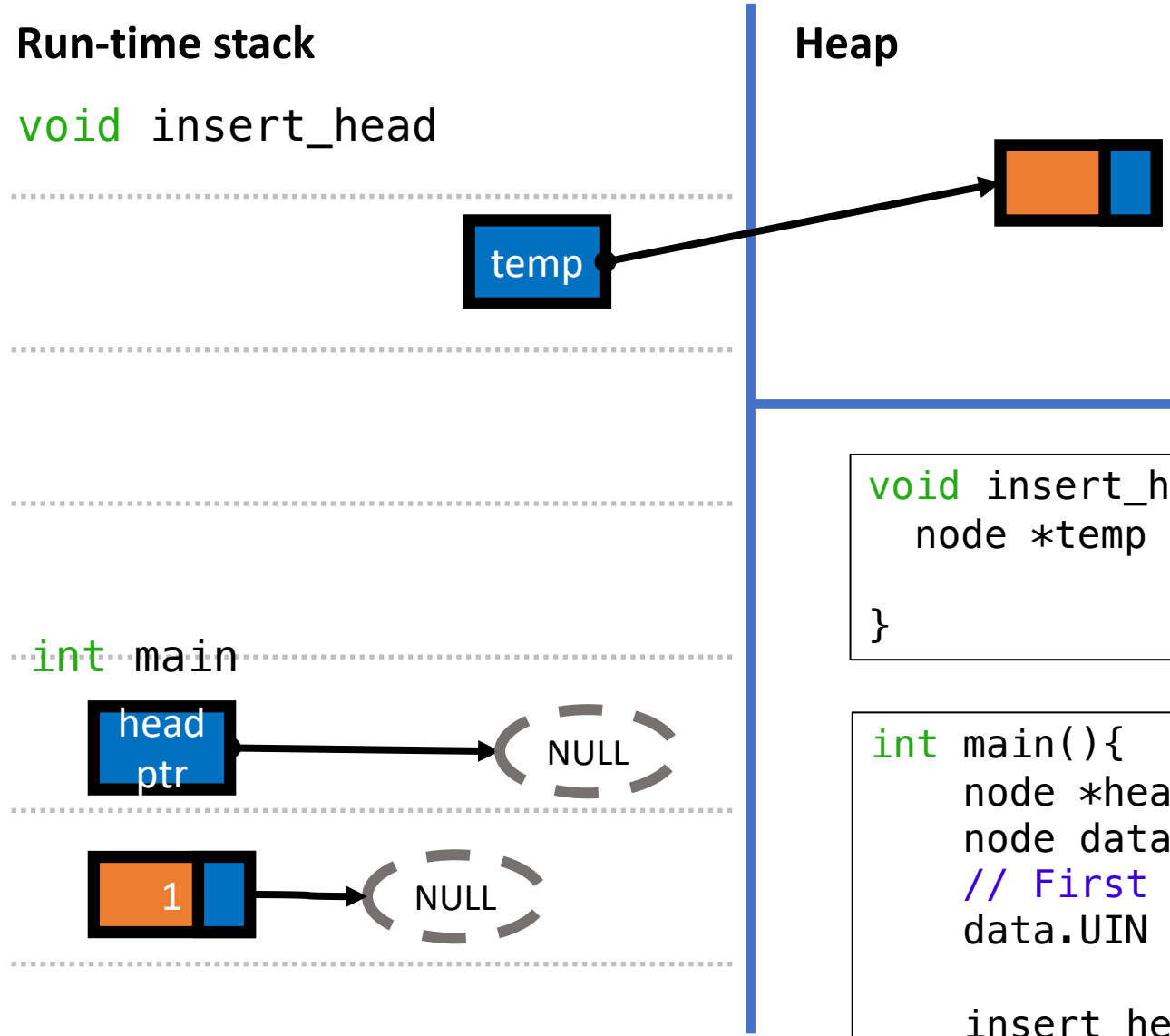
A. Passing by value will push the entire struct members onto the run-time stack.

B. Pass only one pointer.

# Task1: Insert a new node at the head

**Run-time stack**

**Heap**




```
void insert_head(){
  node *temp = malloc(sizeof(node));

}
```

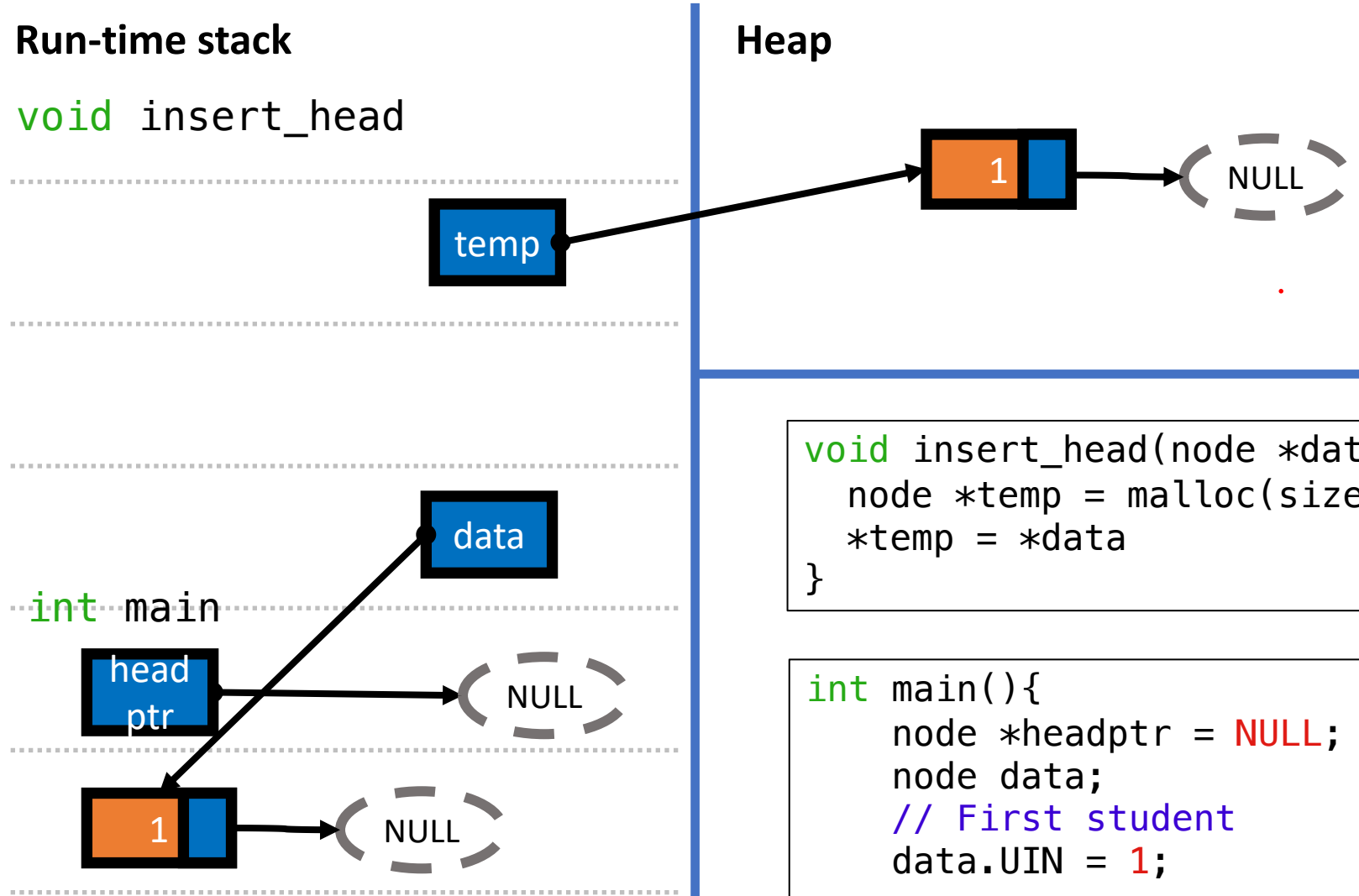
```
int main(){
    node *headptr = NULL;
    node data;
    // First student
    data.UIN = 1;

    insert_head(???);
```

# Task1: Insert a new node at the head

**Run-time stack**

**Heap**

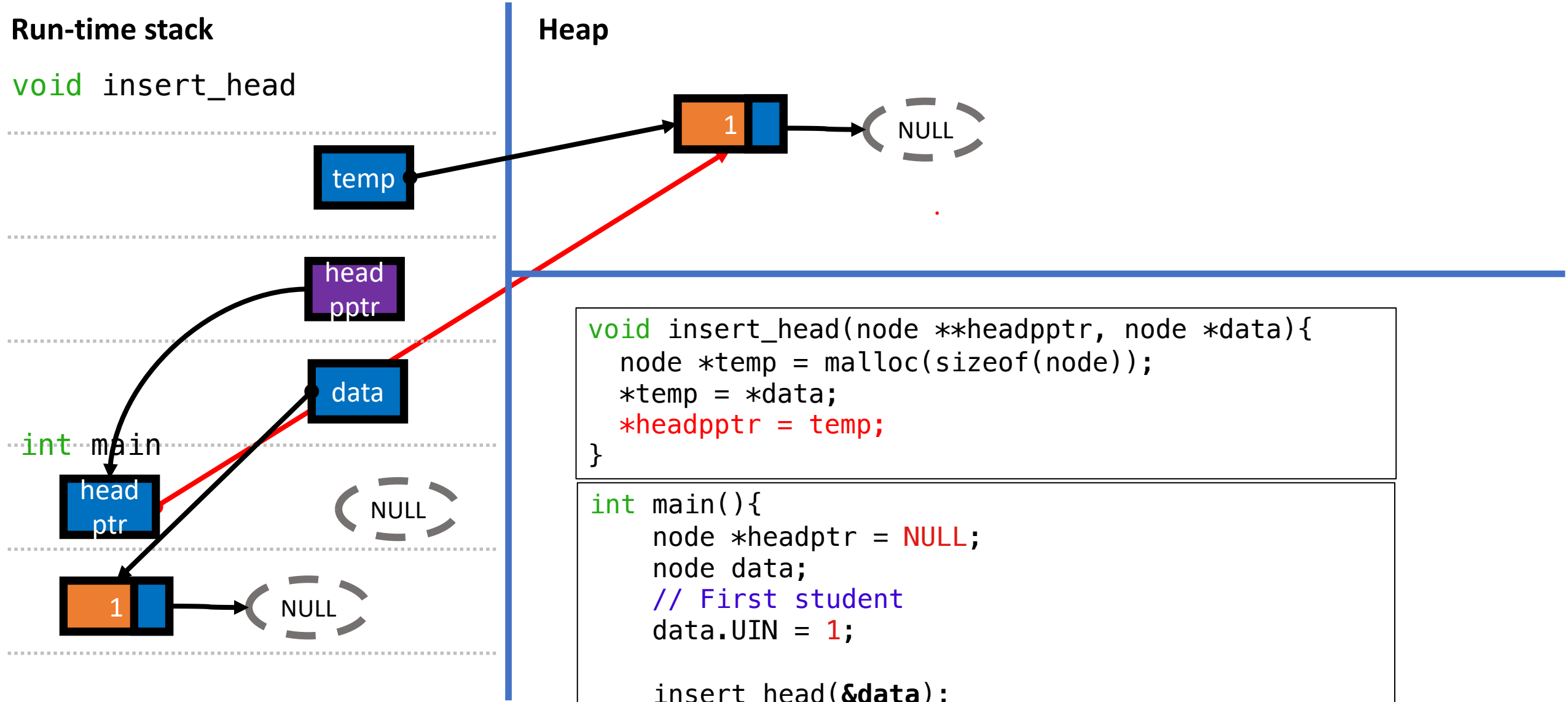



```
void insert_head(node *data){
  node *temp = malloc(sizeof(node));
  *temp = *data
}
```

```
int main(){
    node *headptr = NULL;
    node data;
    // First student
    data.UIN = 1;

    insert_head(&data);
```

# Task1: Insert a new node at the head

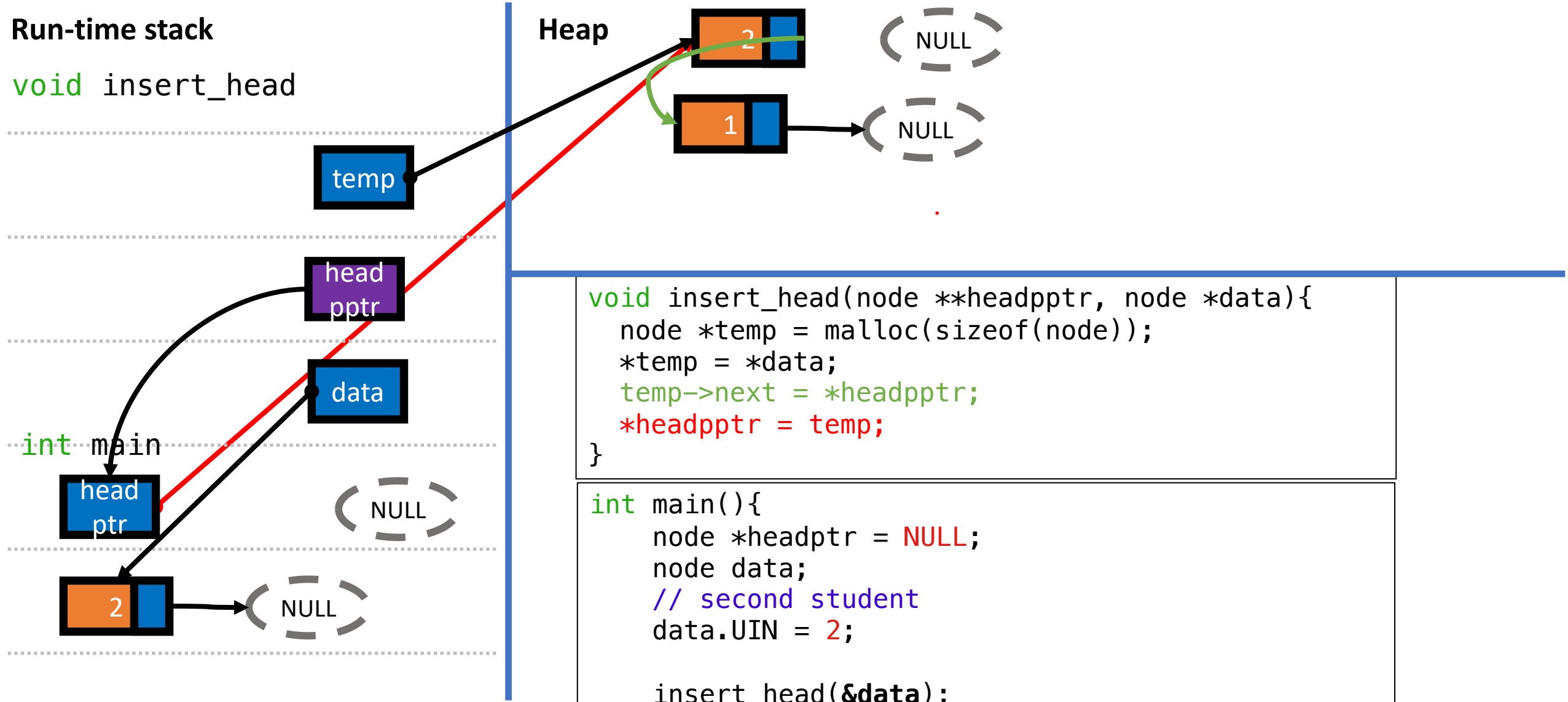



```
void insert_head(node **headpptr, node *data){
  node *temp = malloc(sizeof(node));
  *temp = *data;
  *headpptr = temp;
}
```

```
int main(){
    node *headptr = NULL;
    node data;
    // First student
    data.UIN = 1;

    insert_head(&data);
```

# Task1: Insert a new node at the head

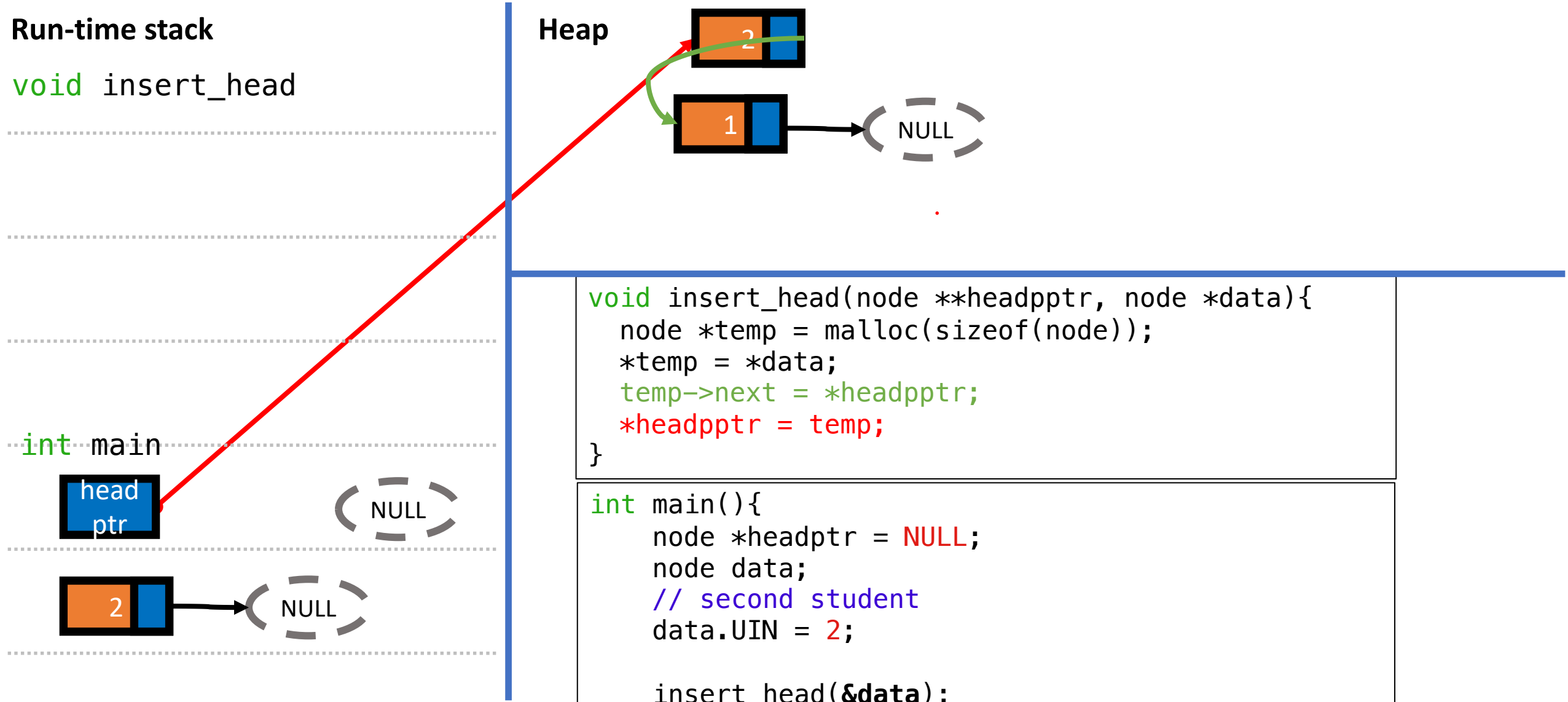



```
void insert_head(node **headpptr, node *data){
  node *temp = malloc(sizeof(node));
  *temp = *data;
  temp->next = *headpptr;
  *headpptr = temp;
}
```

```
int main(){
    node *headptr = NULL;
    node data;
    // second student
    data.UIN = 2;

    insert_head(&data);
```

# Task1: Insert a new node at the head




```
void insert_head(node **headpptr, node *data){
  node *temp = malloc(sizeof(node));
  *temp = *data;
  temp->next = *headpptr;
  *headpptr = temp;
}
```

```
int main(){
    node *headptr = NULL;
    node data;
    // second student
    data.UIN = 2;

    insert_head(&data);
```

**double pointer** because we need to modify (**write**) the head pointer in main

**single pointer** because we need to **read** the data node in main

```
void insert_head(node **headpptr, node *data);

int main(){
    node *headptr = NULL;

    node data;
    data.UIN = 0;
       ...


    insert_head(&headptr, &data);
```

# Task2: Print all nodes

```
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
    struct StudentStruct *next;
}node;
```

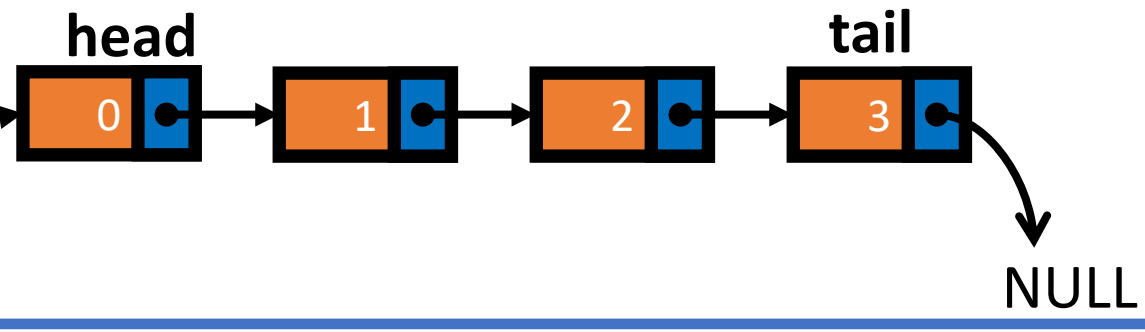
**Run-time stack**

```
void printStudents
```

```
int main
```

head ptr

**Heap**




```
void printStudents(____________){
    printf("UIN netid GPA\n");
    while(_____________________){
        printf("%d %s %f\n", _________);
        ________________;
    }
}
```

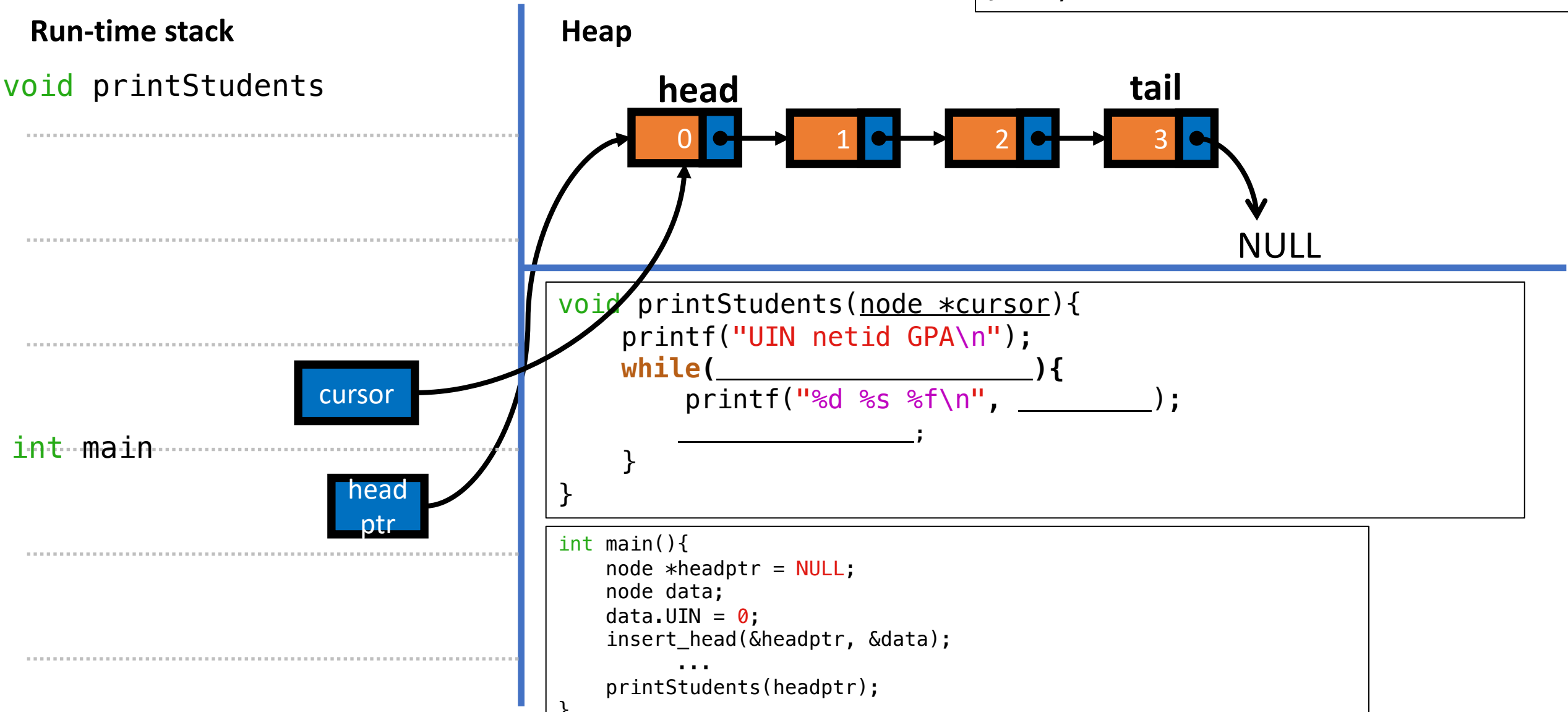
```
int main(){
    node *headptr = NULL;
    node data;
    data.UIN = 0;
    insert_head(&headptr, &data);
        ...
    printStudents(headptr);
}
```

# Task2: Print all nodes

```
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
    struct StudentStruct *next;
}node;
```

**Run-time stack**

**Heap**




```
void printStudents(node *cursor){
    printf("UIN netid GPA\n");
    while(_____________________){
        printf("%d %s %f\n", _________);
        _________________;
    }
}
```

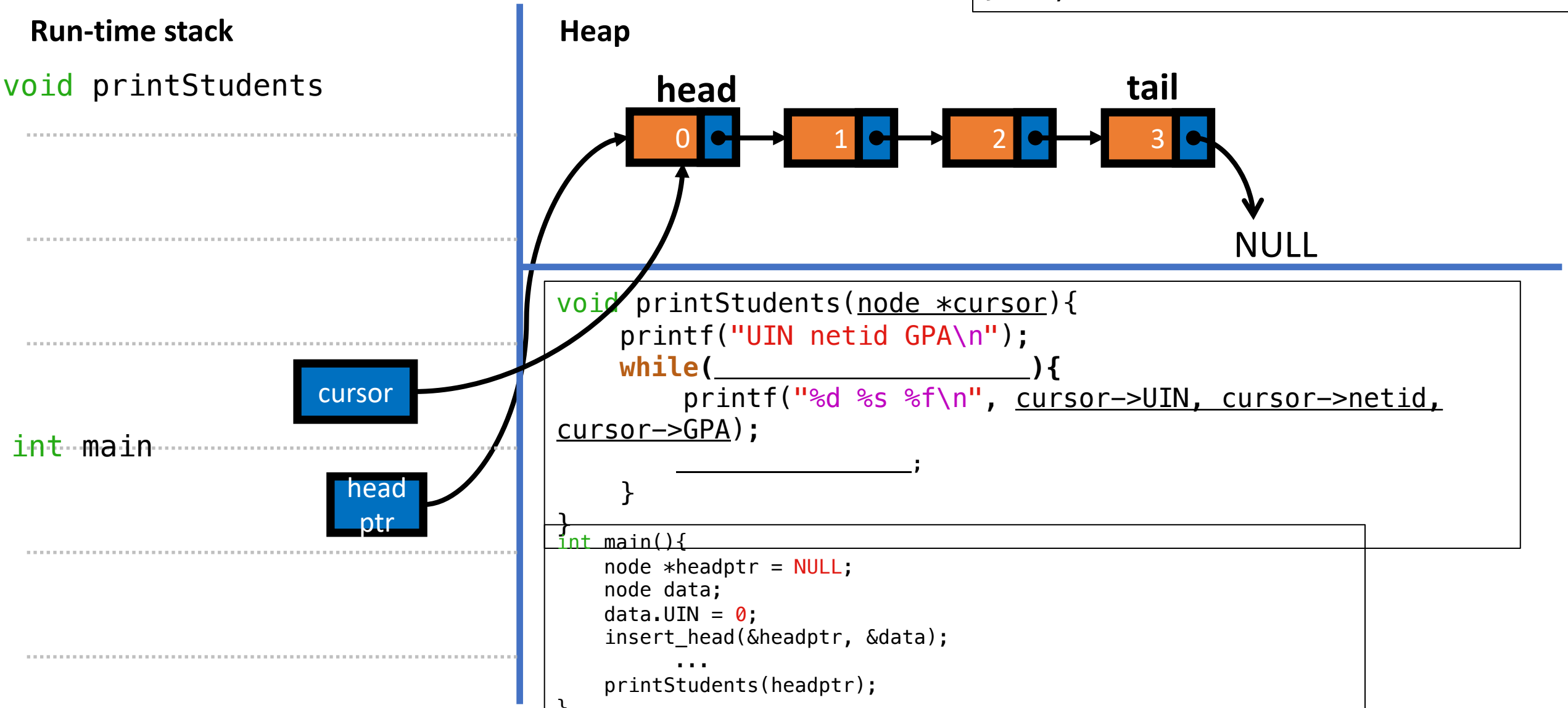
```
int main(){
    node *headptr = NULL;
    node data;
    data.UIN = 0;
    insert_head(&headptr, &data);
        ...
    printStudents(headptr);
}
```

# Task2: Print all nodes

```c
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
    struct StudentStruct *next;
}node;
```

**Run-time stack**

**Heap**

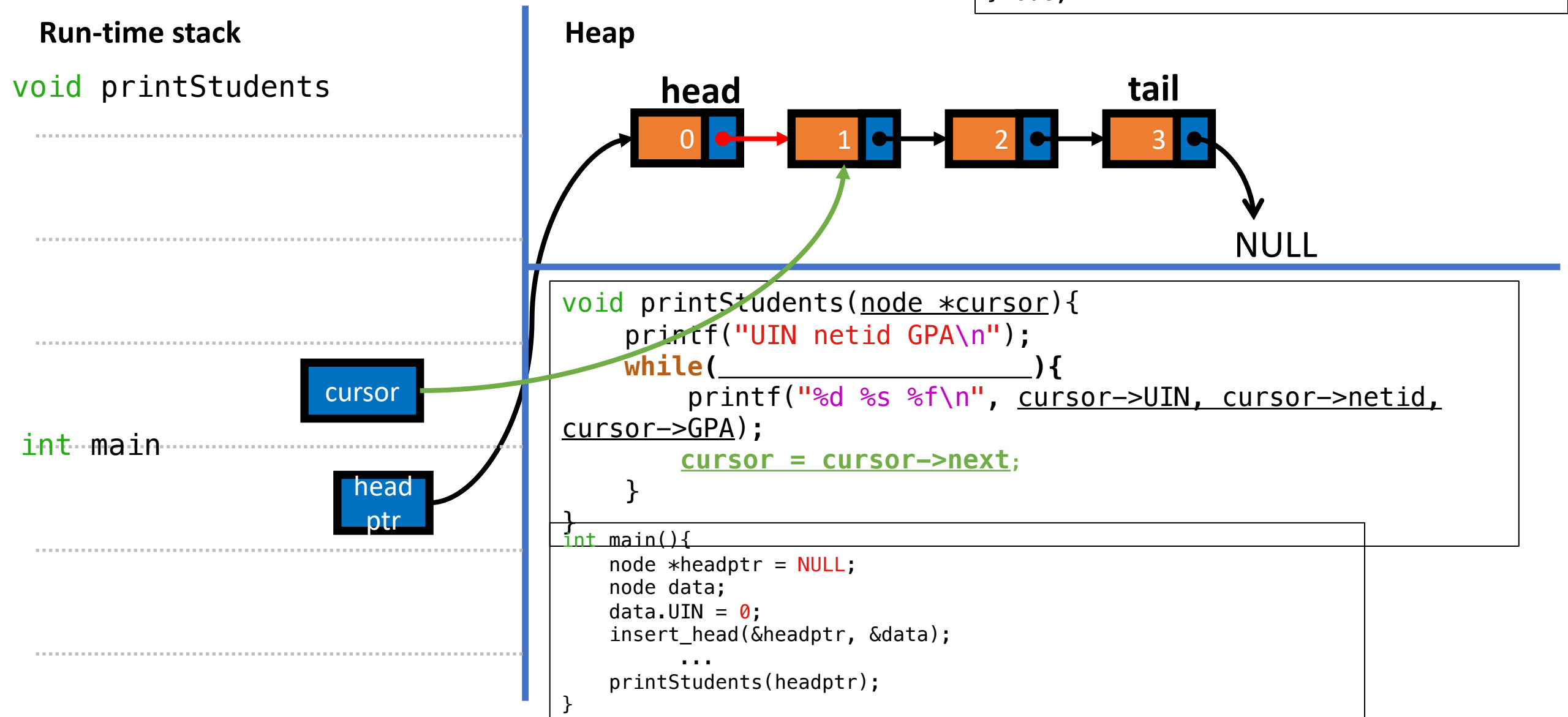



```c
void printStudents(node *cursor){
    printf("UIN netid GPA\n");
    while(_____________________){
        printf("%d %s %f\n", cursor->UIN, cursor->netid,
cursor->GPA);
        _____________________;
    }
}
```

```c
int main(){
    node *headptr = NULL;
    node data;
    data.UIN = 0;
    insert_head(&headptr, &data);
        ...
    printStudents(headptr);
}
```

# Task2: Print all nodes

```
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
    struct StudentStruct *next;
}node;
```

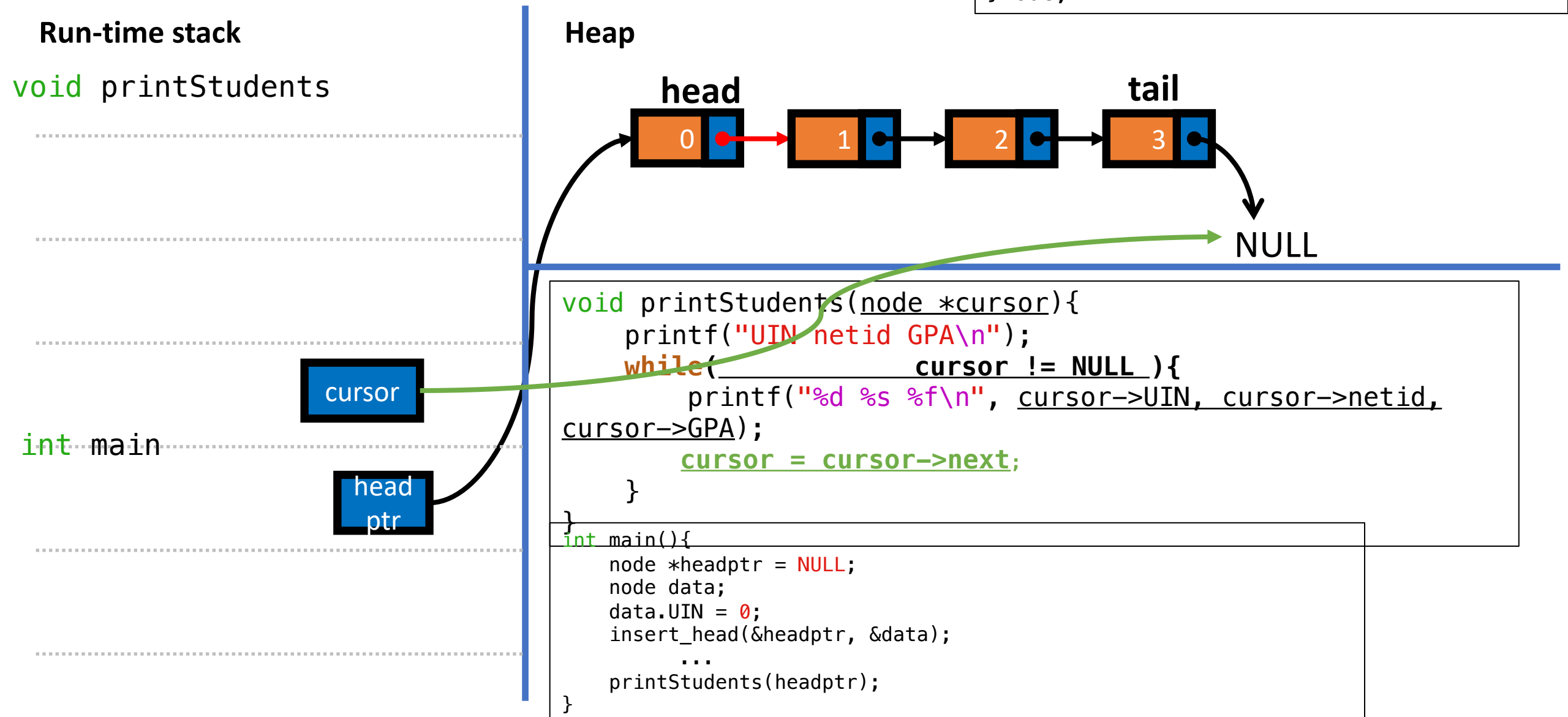



```
void printStudents(node *cursor){
    printf("UIN netid GPA\n");
    while(_____________________){
        printf("%d %s %f\n", cursor->UIN, cursor->netid,
cursor->GPA);
        cursor = cursor->next;
    }
}
```

```
int main(){
    node *headptr = NULL;
    node data;
    data.UIN = 0;
    insert_head(&headptr, &data);
        ...
    printStudents(headptr);
}
```

# Task2: Print all nodes

```
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
    struct StudentStruct *next;
}node;
```




```
void printStudents(node *cursor){
    printf("UIN netid GPA\n");
    while(cursor != NULL ){
        printf("%d %s %f\n", cursor->UIN, cursor->netid,
cursor->GPA);
        cursor = cursor->next;
    }
}
```

```
int main(){
    node *headptr = NULL;
    node data;
    data.UIN = 0;
    insert_head(&headptr, &data);
        ...
    printStudents(headptr);
}
```

# Task3: Delete all nodes

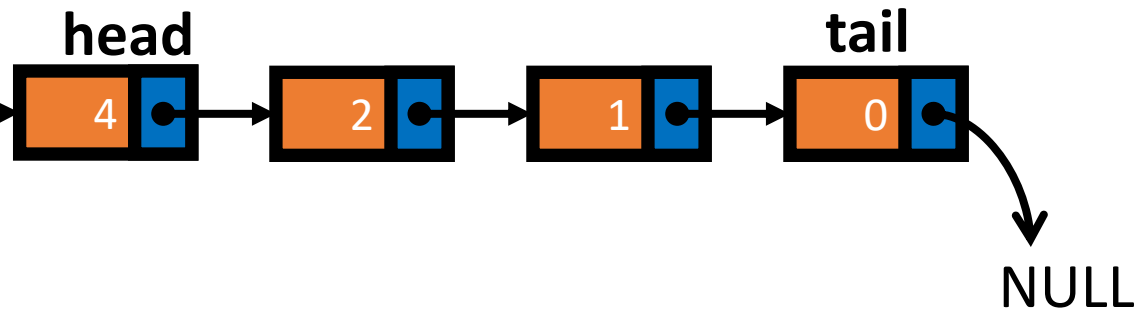
**Run-time stack**

```
void delete_all()
```

```
int main()
```

head ptr

**Heap**




```
void delete_all(){
}
```

# Task3: Delete all nodes

**Run-time stack**

```
void delete_all()
```

temp

head pptr

```
int main()
```

head ptr

**Heap**

head

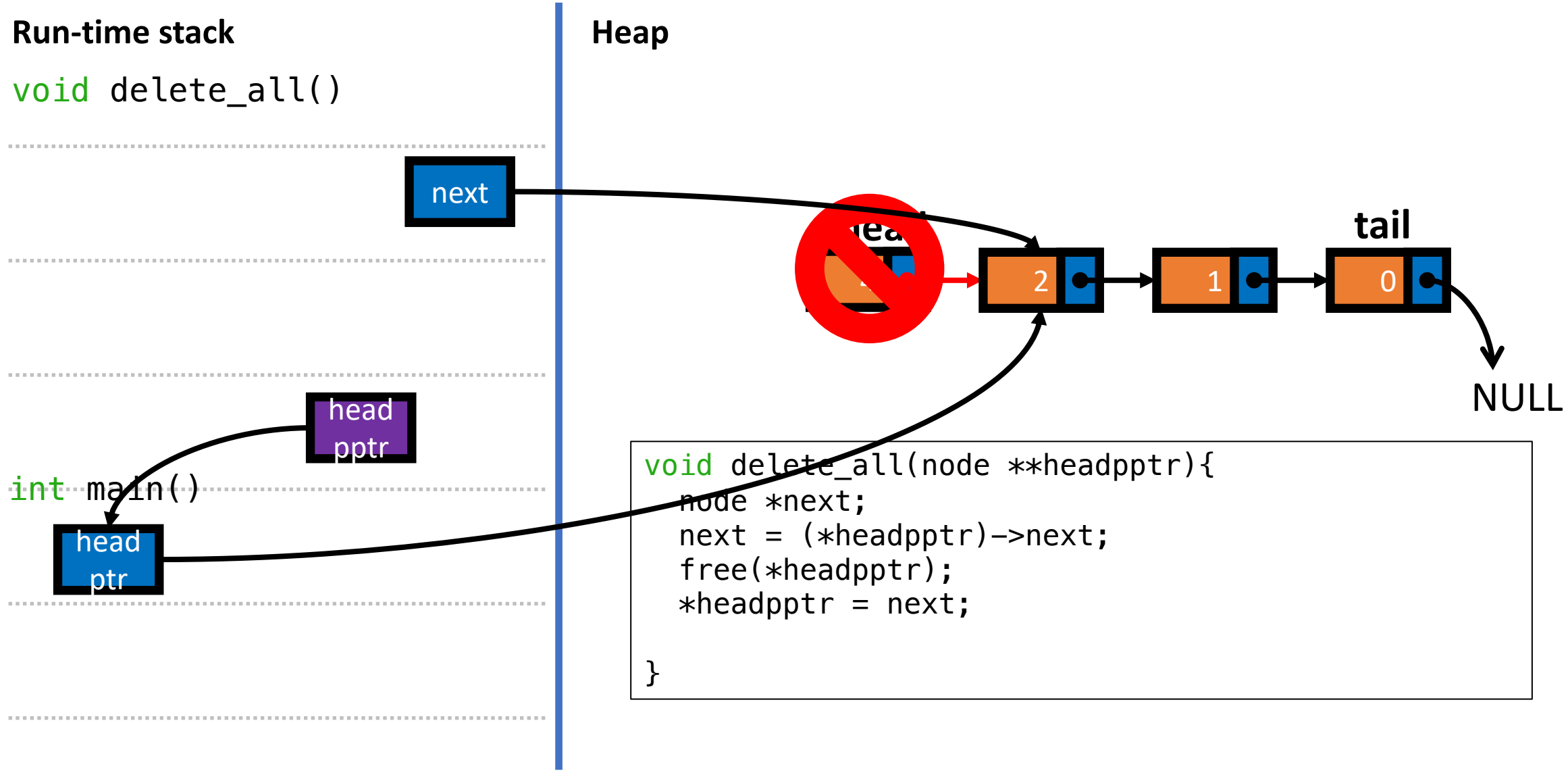
tail

4 → 2 → 1 → 0 → NULL

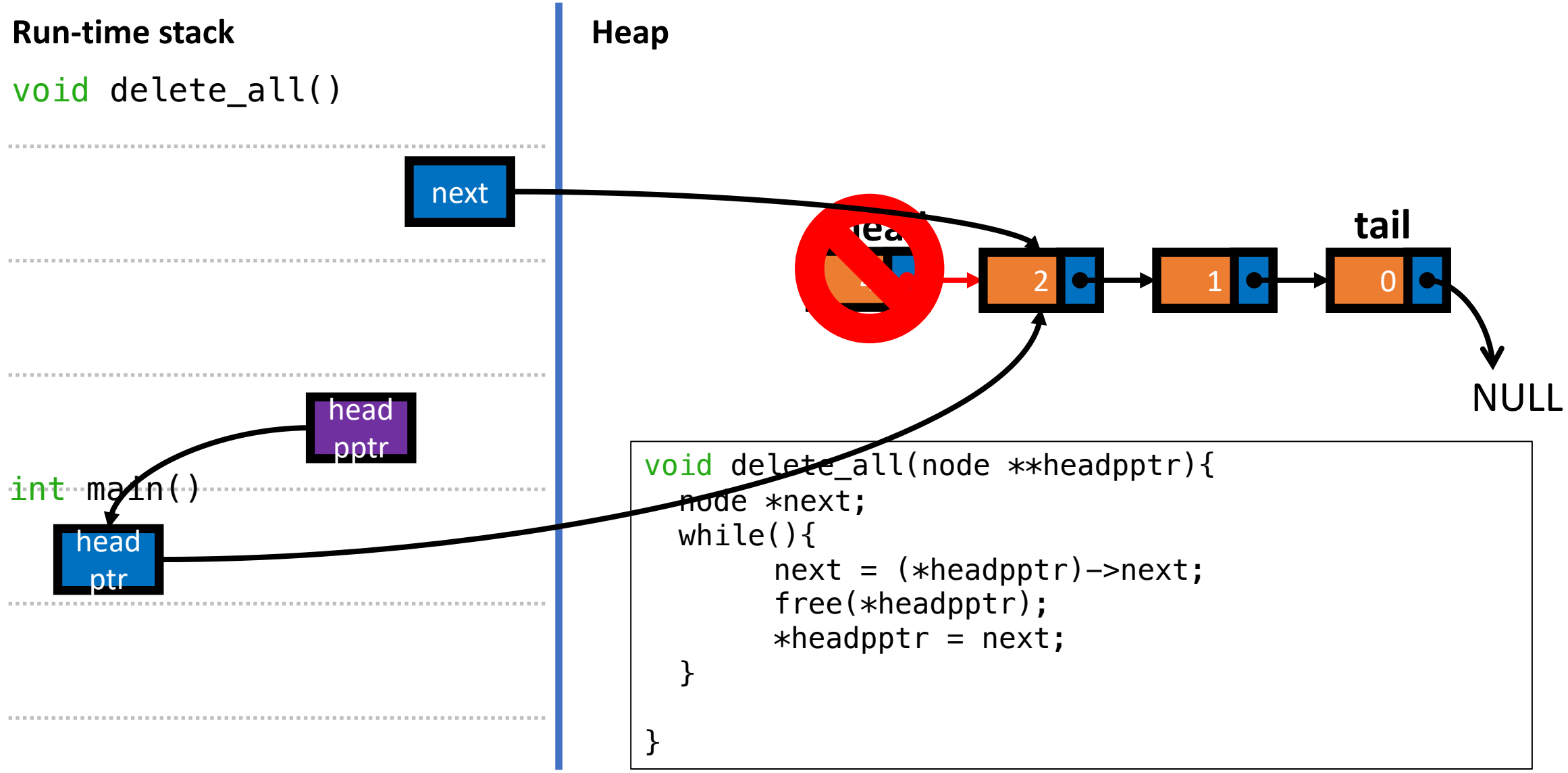
```
void delete_all(node **headpptr){
  node *temp;
  temp = (*headpptr)->next;

}
```

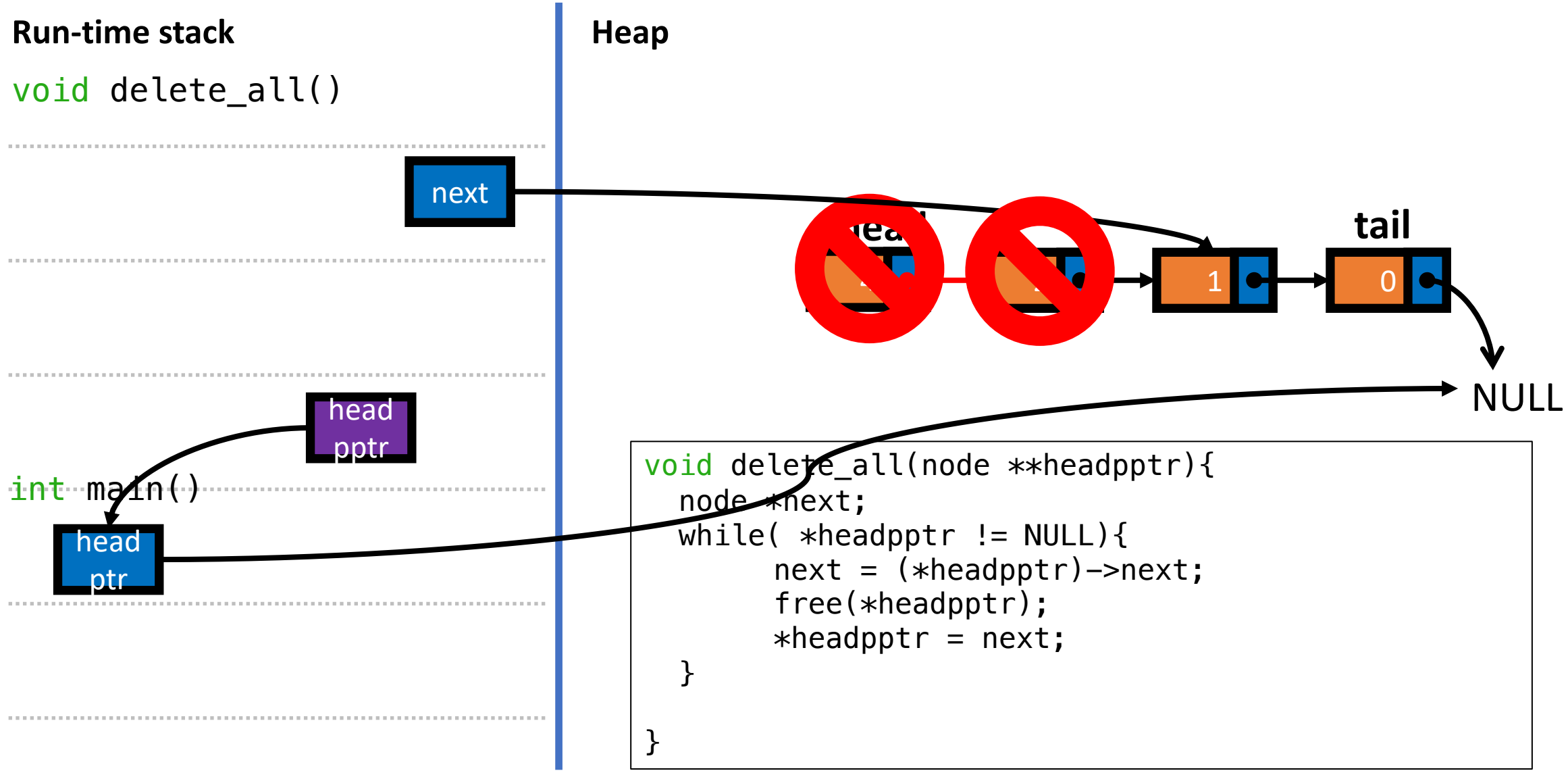
# Task3: Delete all nodes



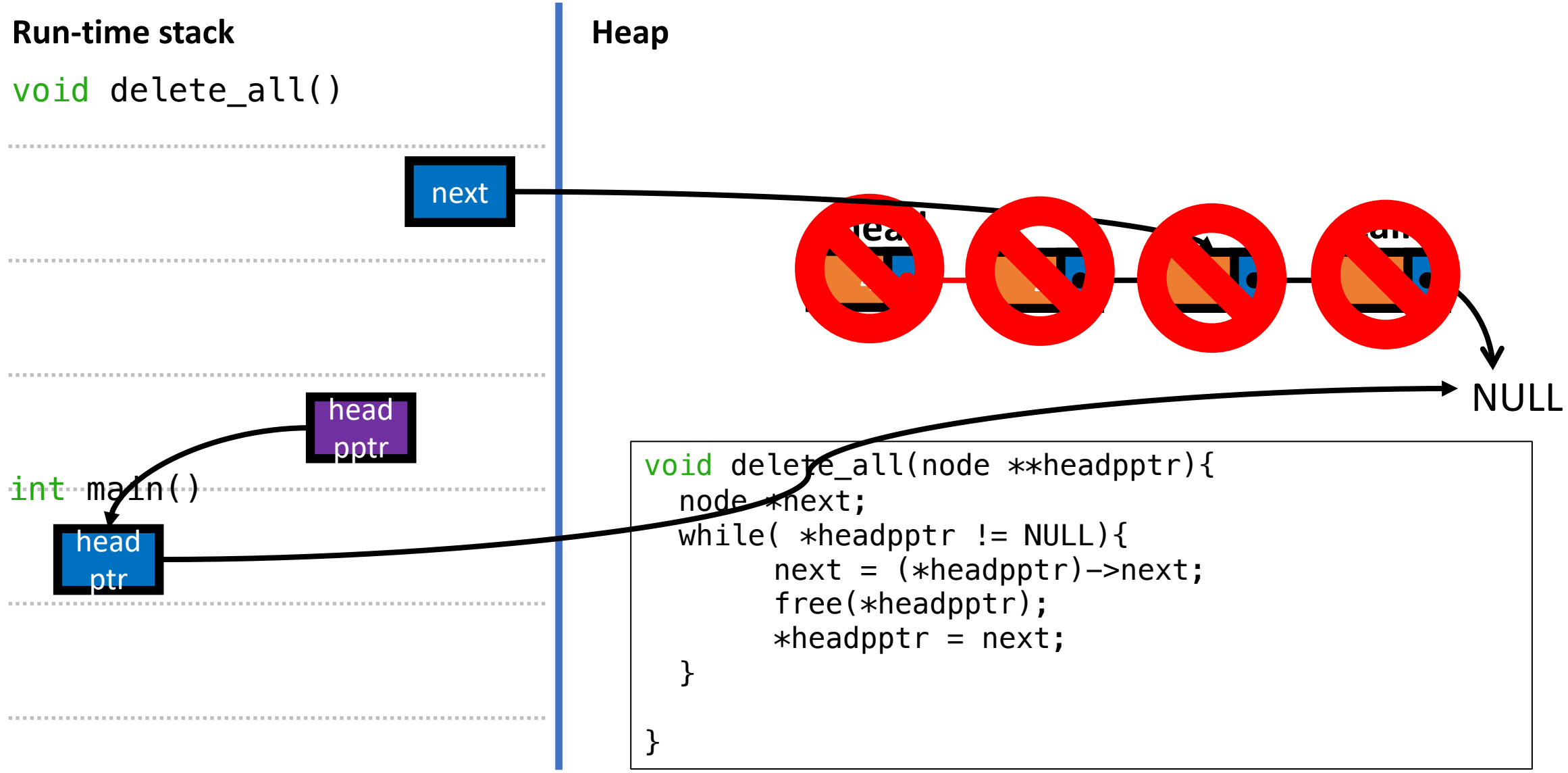
# Task3: Delete all nodes



# Task3: Delete all nodes



# Task3: Delete all nodes

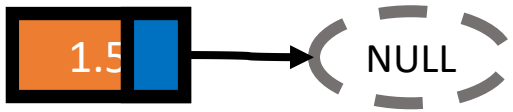


# Task4: Insert a new node in a sorted way (GPA, descending)

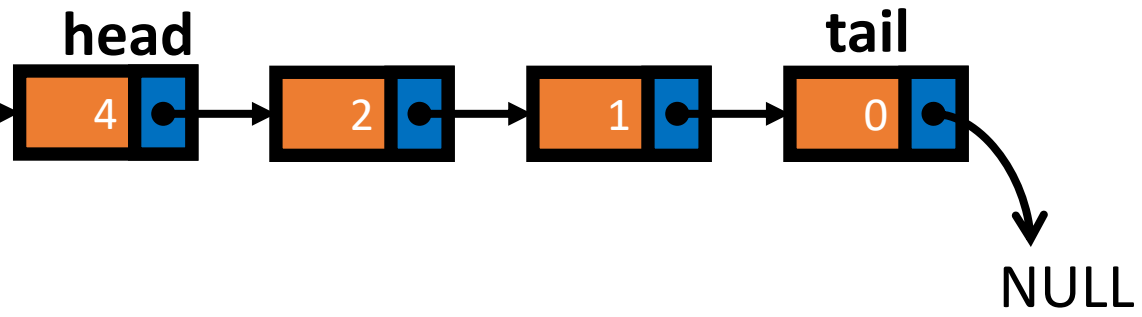
**Run-time stack**

```
void insert_sorted_GPA()
```

```
int main()
```


**Heap**




```
void insert_sorted_GPA(){
}
```