ECE 220: Computer Systems & Programming

Lecture 19: Linked List

Thomas Moon

March 31, 2022

- MP8 due this Thursday.
- Quiz4 is going on.

- Midterm2: 7pm, 4/7/2022. lec7-17
  - https://wiki.illinois.edu/wiki/display/ece220/Exams
  - Conflict: 5:30pm, 4/7/2022
  - Conflict sign-up by 4/3/2022
  - HKN review session: Sunday, 4/3, 3-5pm in ECEB1015

```
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.

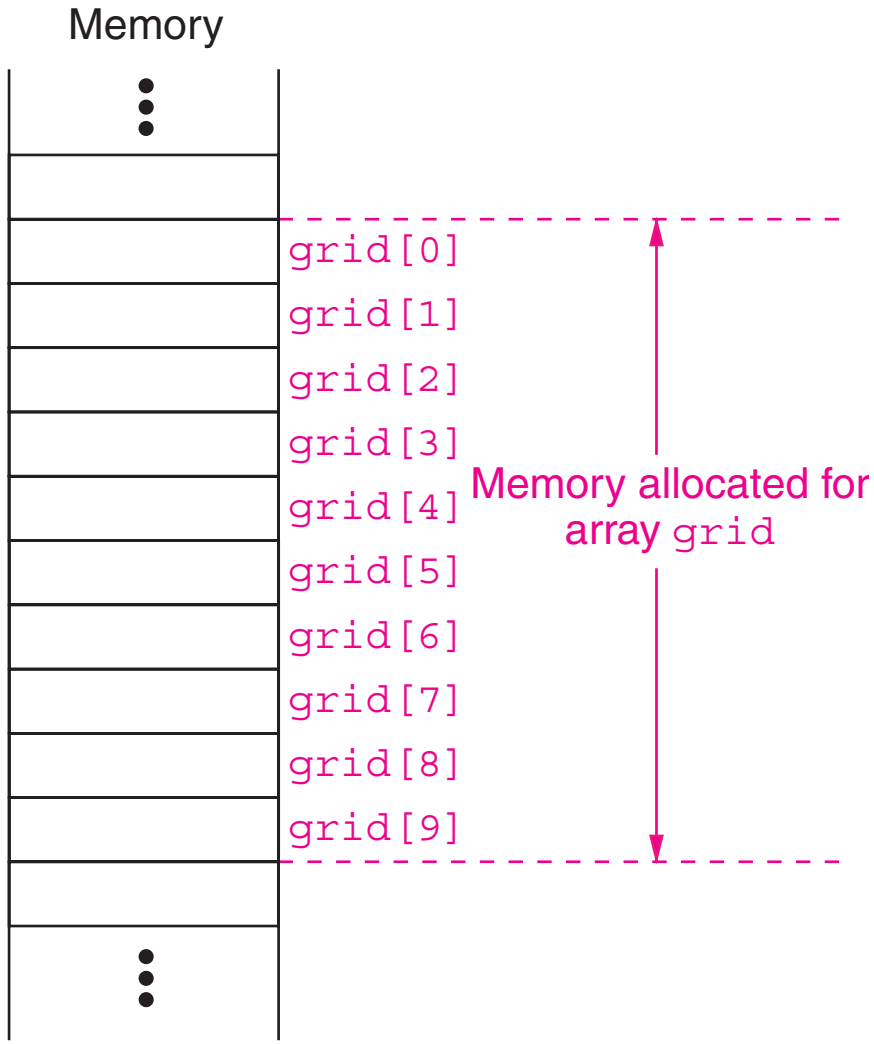
Dynamic Memory Allocation (this lecture)

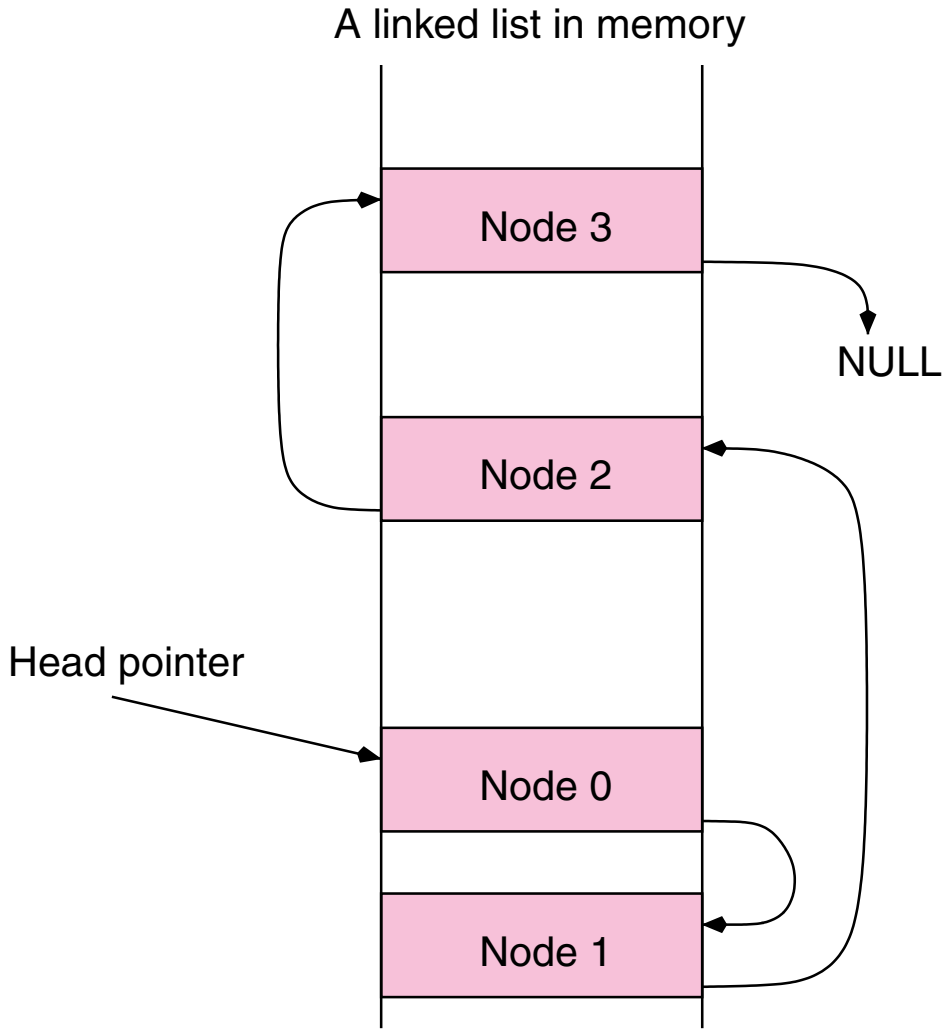
Planes take off & land,
i.e. the data coming in & going out.

Adding or removing an item in the middle.

→ Not efficient in array

Linked List (next lecture)





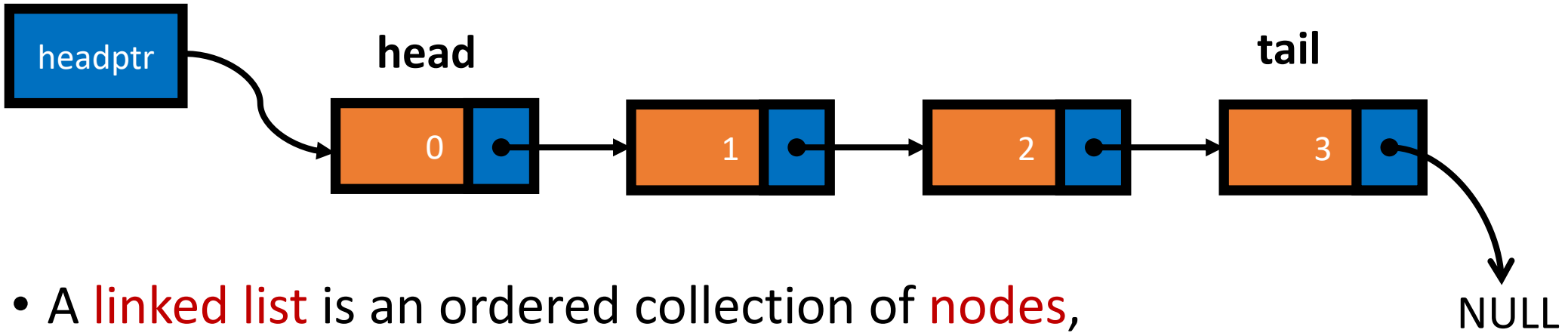
Array

(can be automatic or dynamic)

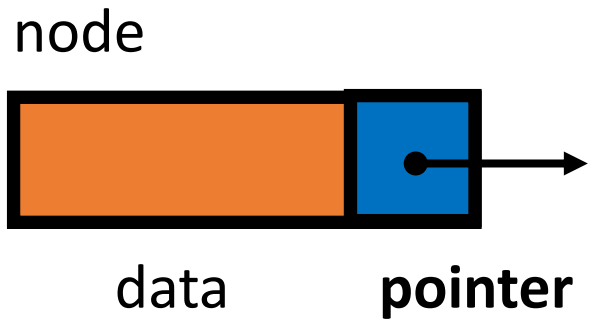
Linked List

(dynamic only)

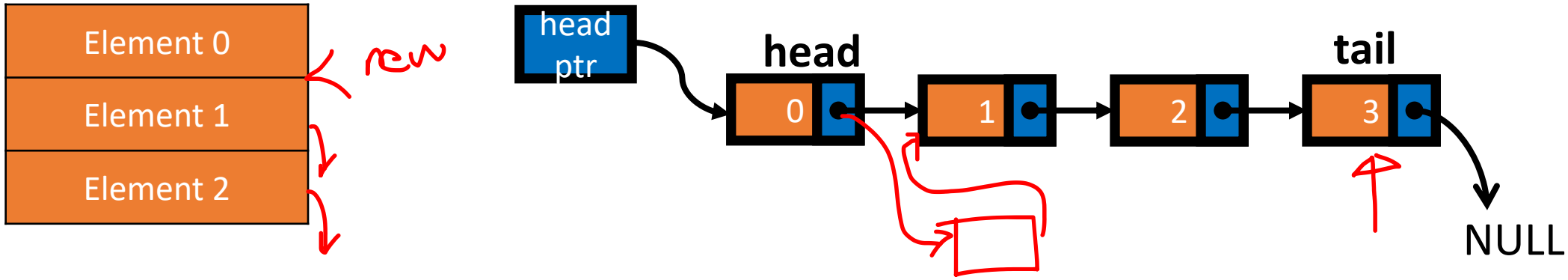
Linked List



- 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



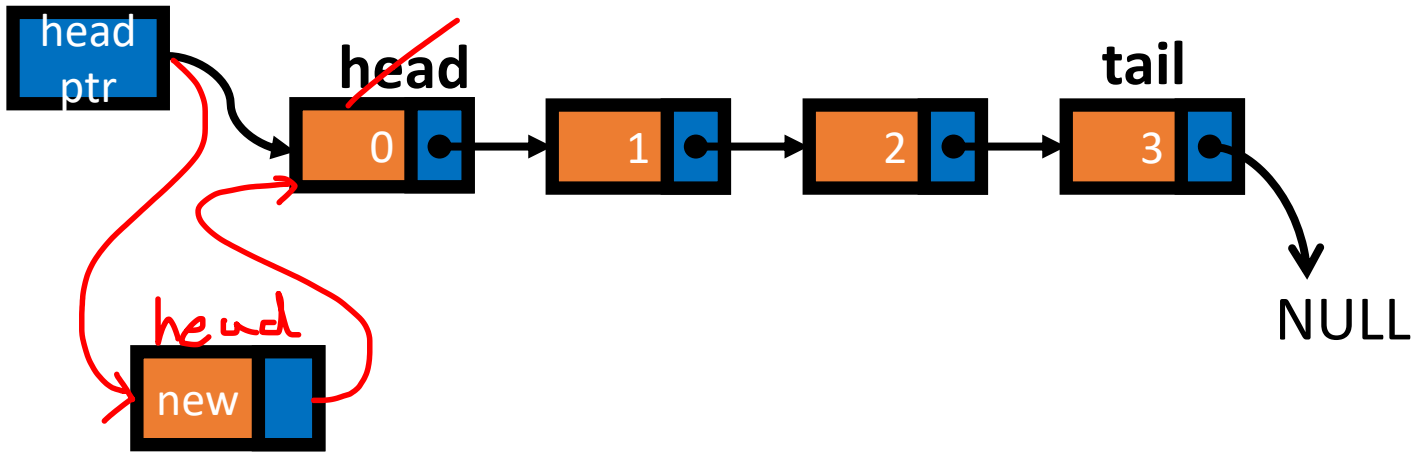
	Array	Linked List
Memory Allocation	Automatic/Dynamic	Dynamic
Memory Structure	Contiguous	Not necessary consecutive
Order of Access	Random	Sequential
Insertion/Deletion	Create/delete space, then shift all successive elements	Change pointer address

Example: Student Record

```
typedef struct StudentStruct{
    int UIN;
    char *netid;
    float GPA;
}student;
```

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

Task1: Insert a new node at the head



Recall1: Solve Swap Problem

```
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;
}
```

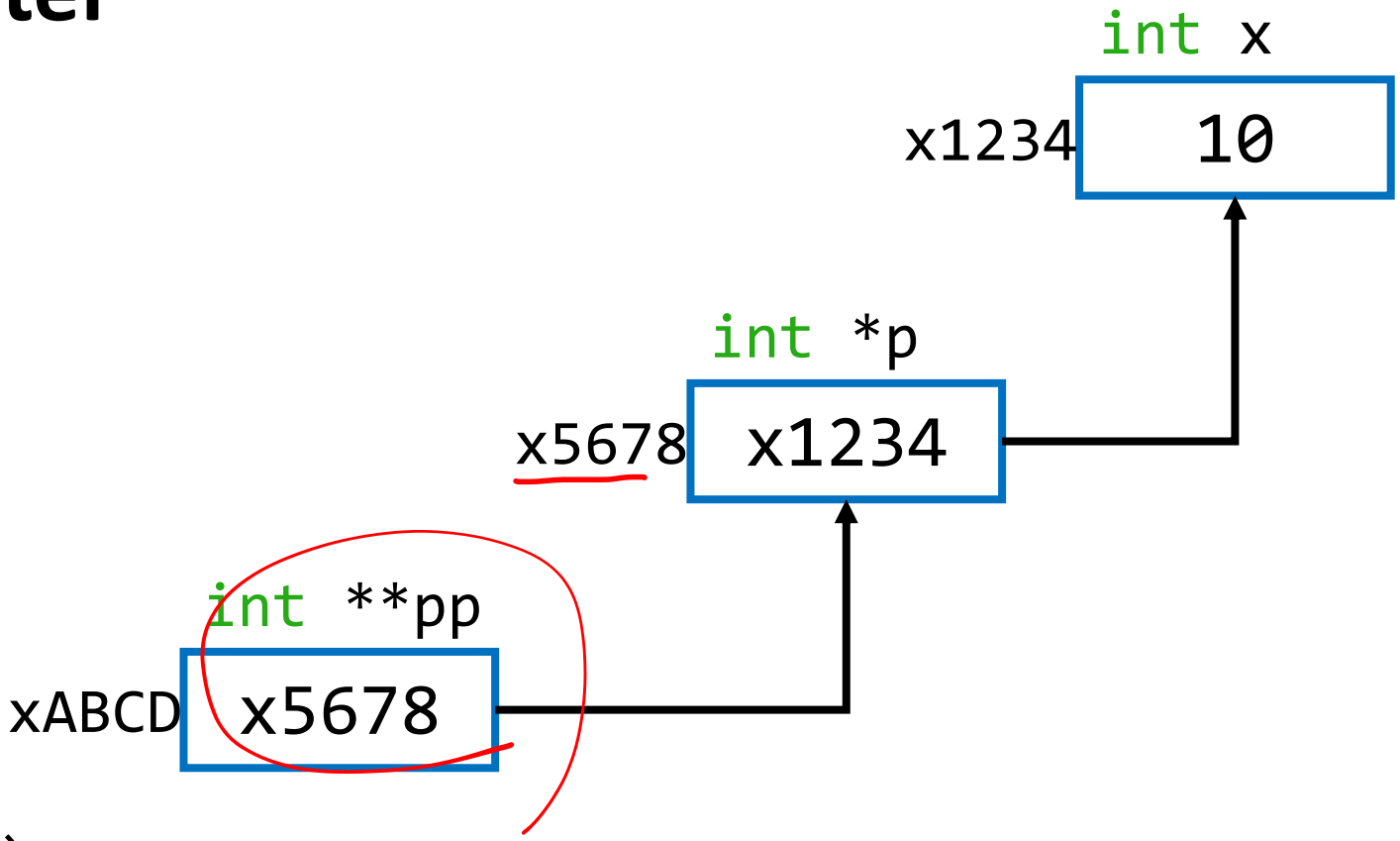
call by value

```
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;
}
```

call by reference

Recall2: 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);
```



Recall3: 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

```
void insert_head
```

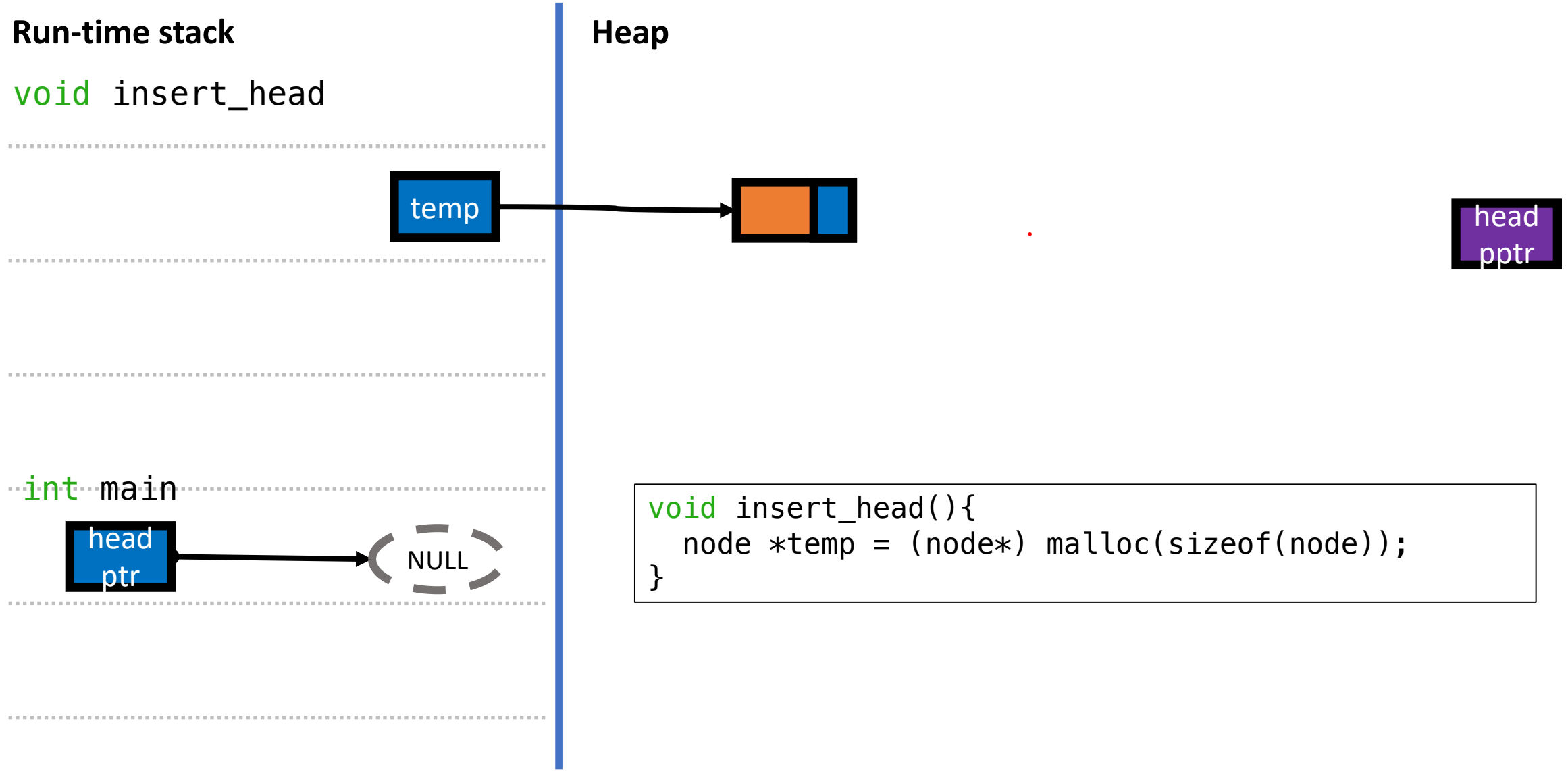
```
int main
```

Heap

Linked list is empty

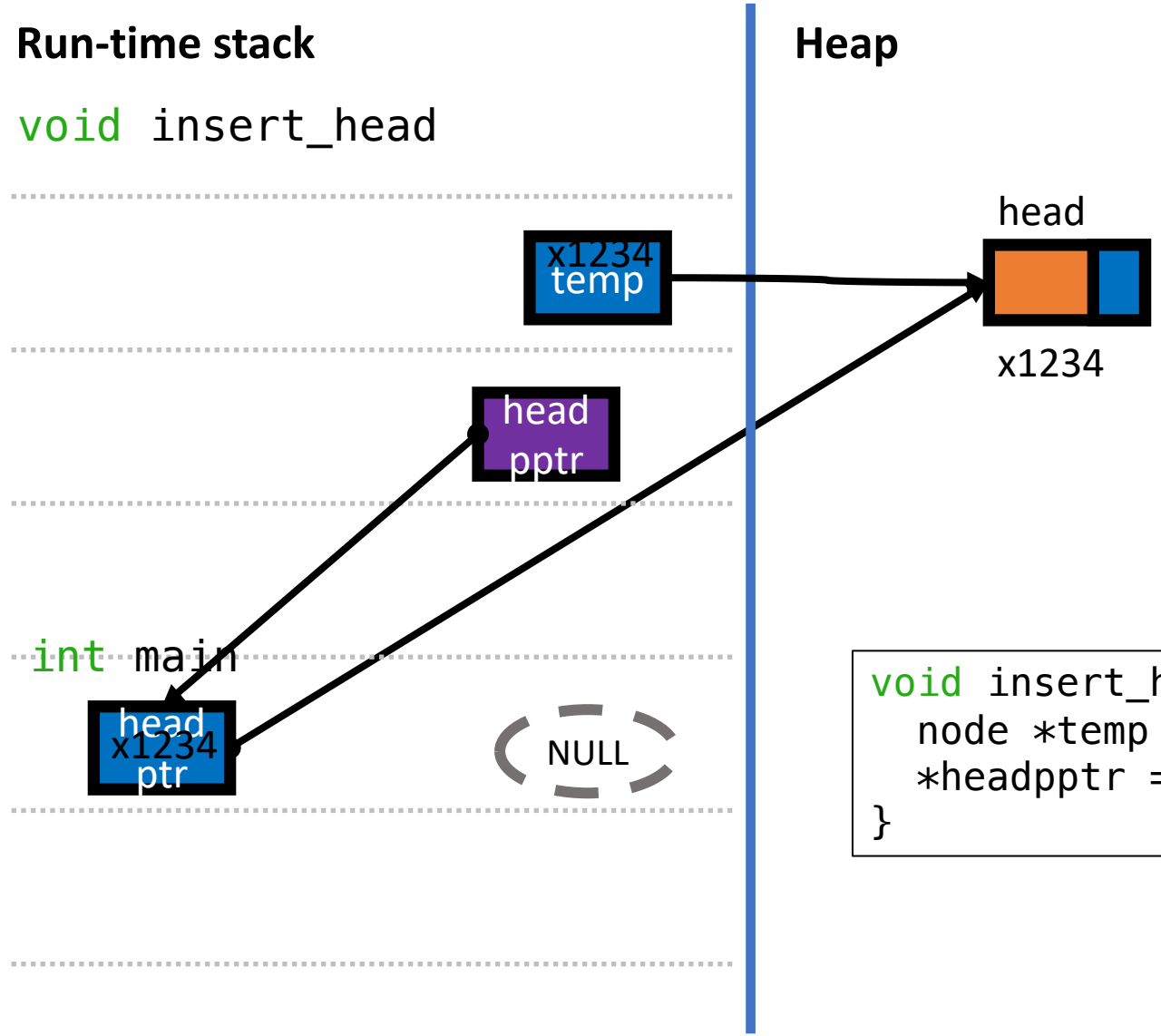
```
void insert_head(){
}
```

Task1: Insert a new node at the head



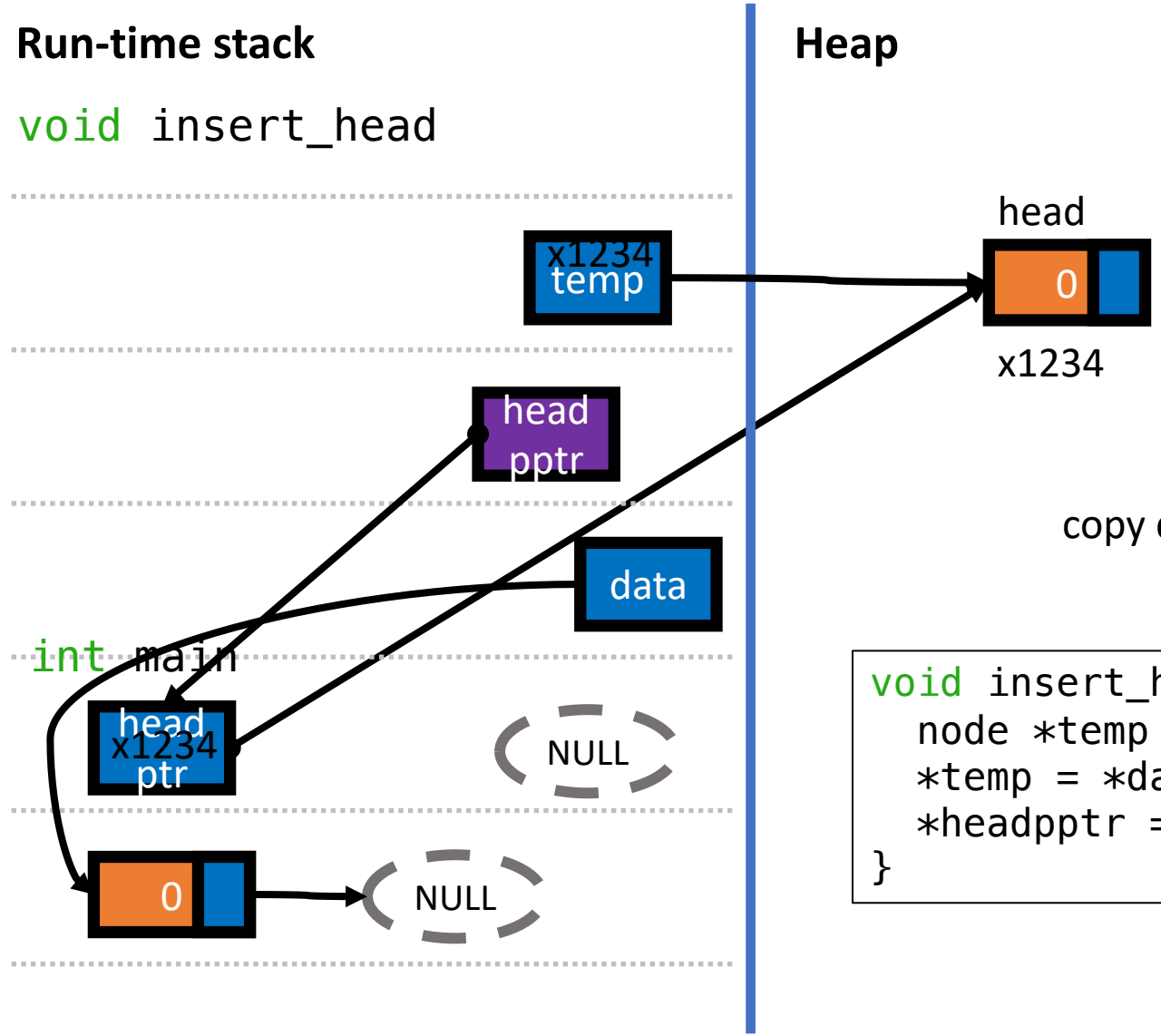

```
void insert_head(){
  node *temp = (node*) malloc(sizeof(node));
}
```

Task1: Insert a new node at the head




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

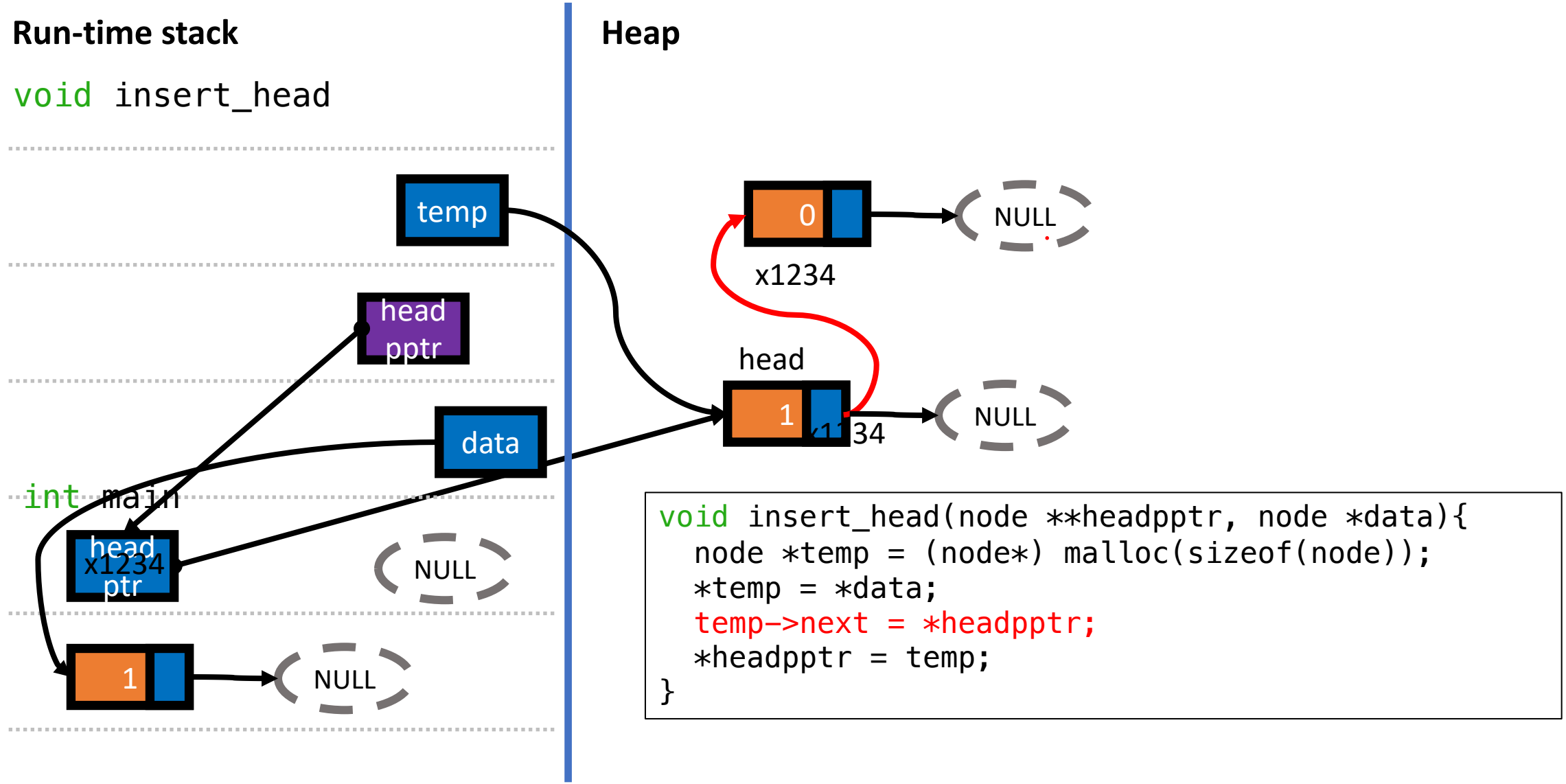
Task1: Insert a new node at the head



copy data from node in main

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

Task1: Insert a new node at the head

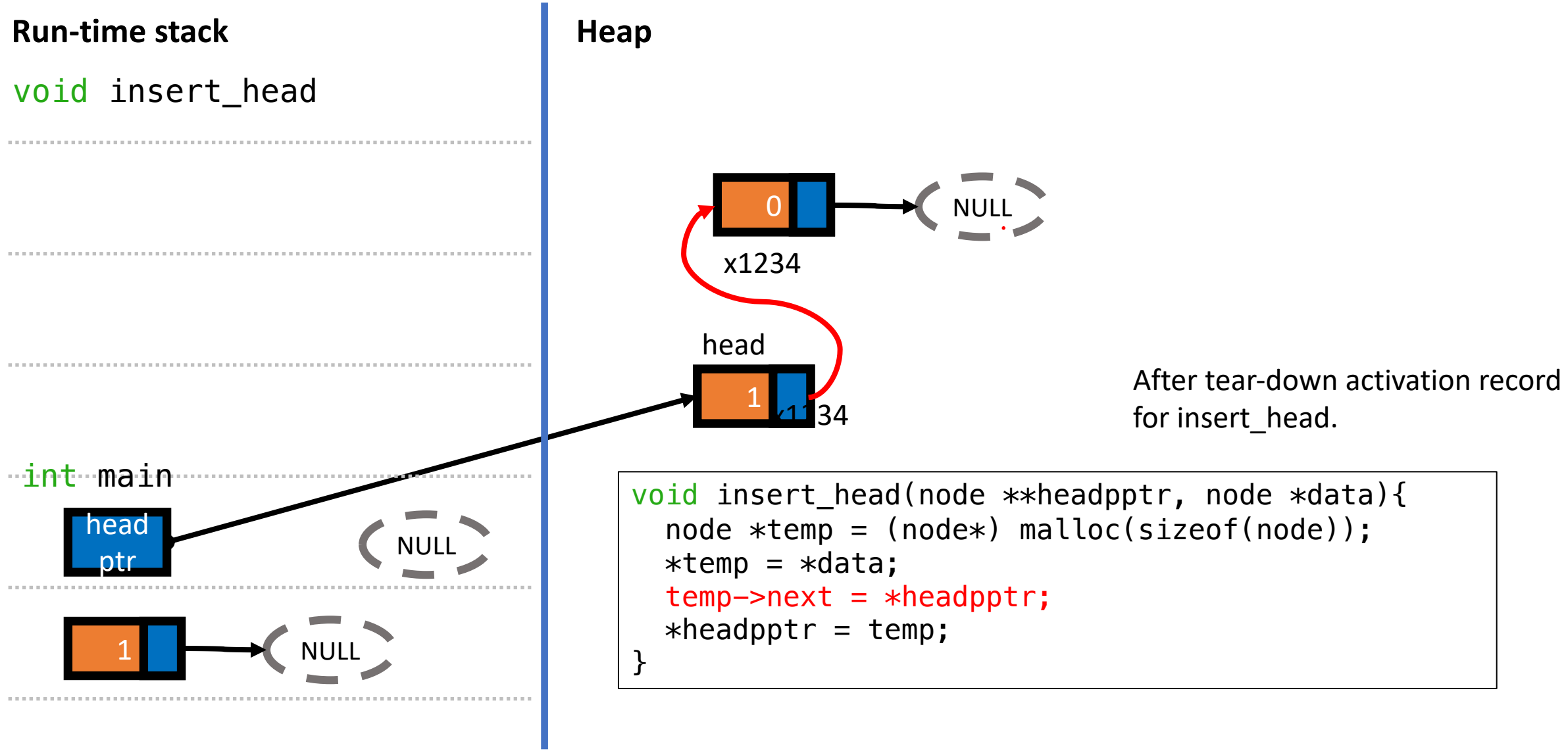



```
void insert_head(node **headpptr, node *data){
  node *temp = (node*) malloc(sizeof(node));
  *temp = *data;
  temp->next = *headpptr;
  *headpptr = temp;
}
```

Task1: Insert a new node at the head

Run-time stack

Heap



After tear-down activation record for insert_head.

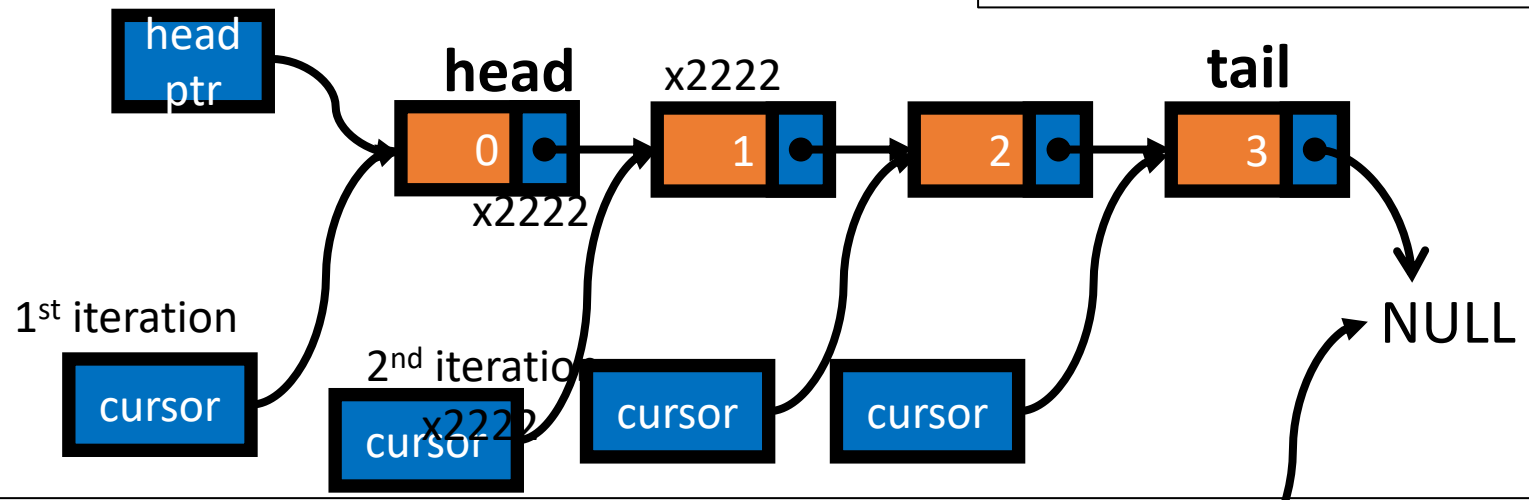
```
void insert_head(node **headpptr, node *data){
  node *temp = (node*) malloc(sizeof(node));
  *temp = *data;
  temp->next = *headpptr;
  *headpptr = temp;
}
```

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

int main(){
    node *headptr = NULL;

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


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

double pointer because we need to modify the head pointer in main

Task2: Print all nodes

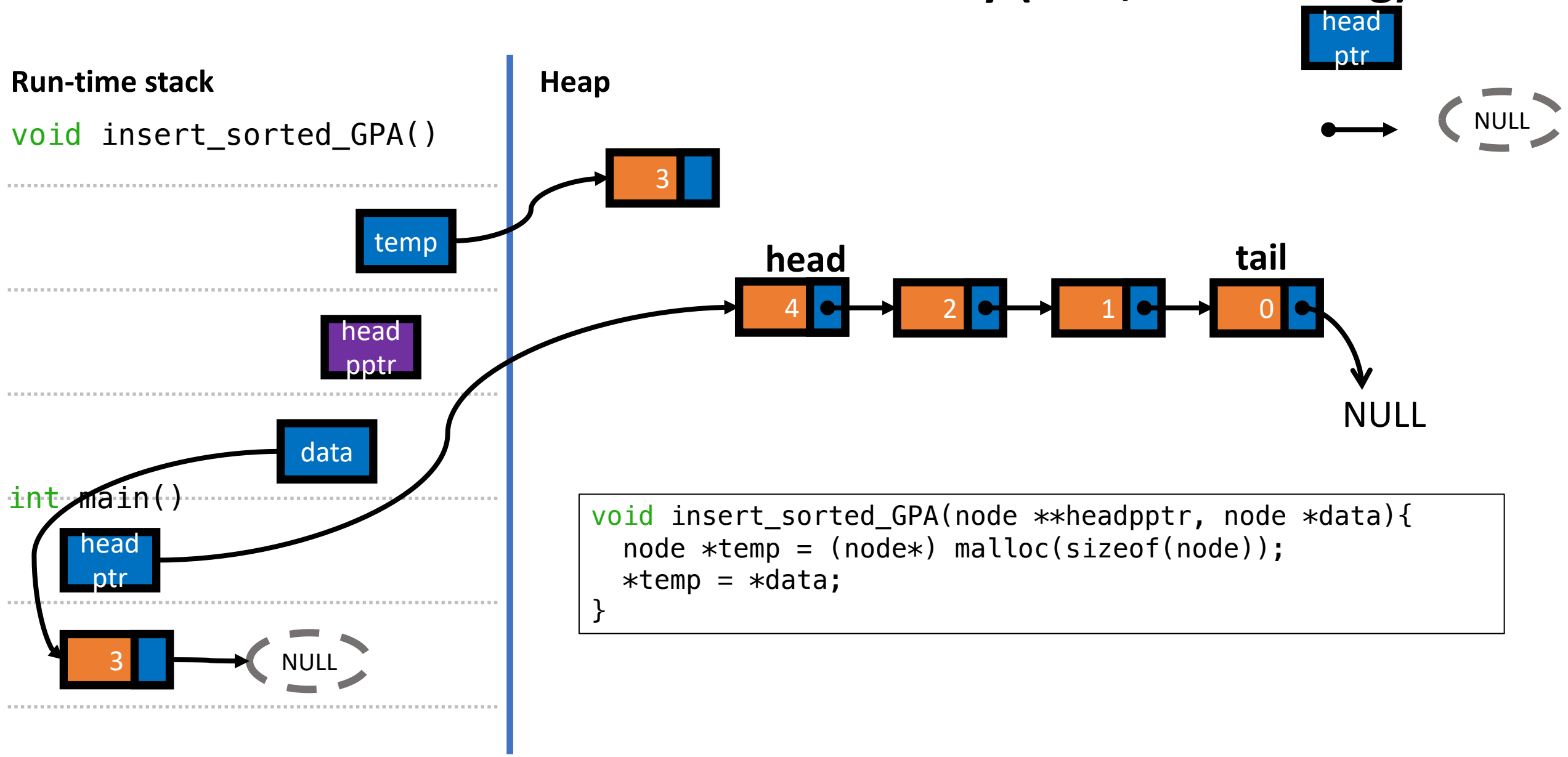
```
typedef struct StudentStruct{
    int UIN;
    char *netid;
    float GPA;
    struct StudentStruct *next;
}node;
```




```
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;
    }
}
```

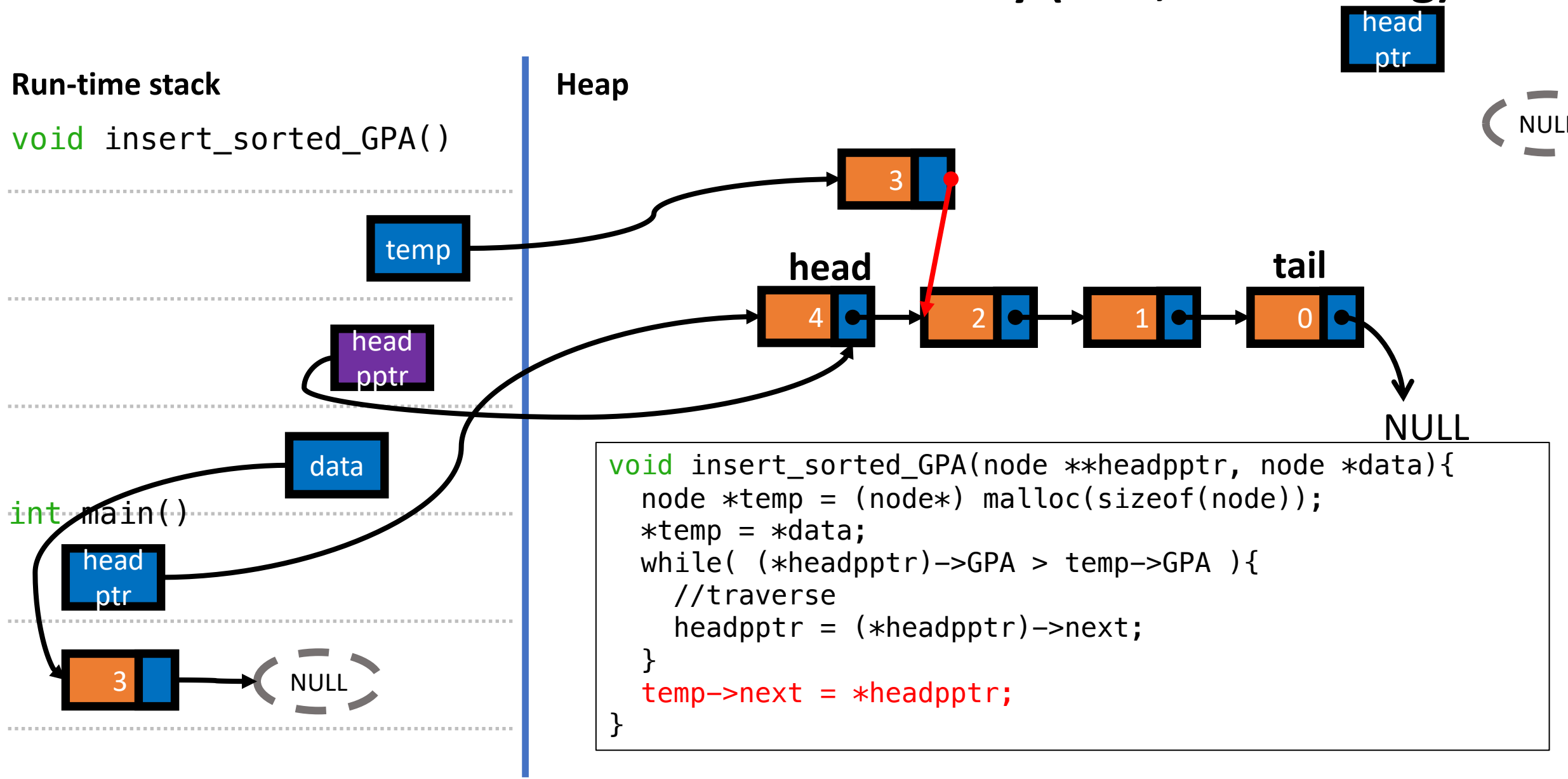
```
printStudents(headptr);
```

Task3: Insert a new node in a sorted way (GPA, descending)



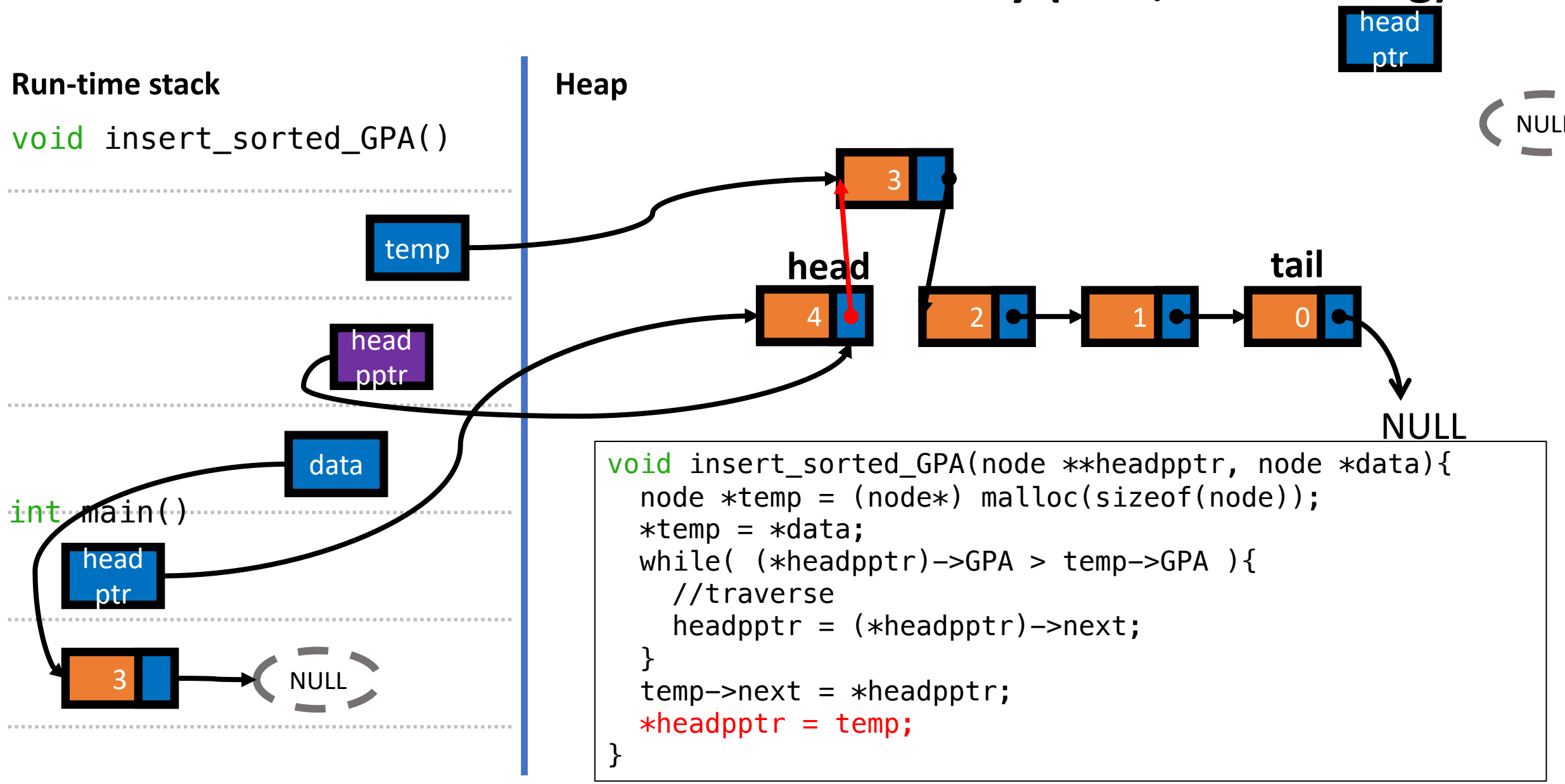

```
void insert_sorted_GPA(node **headpptr, node *data){
  node *temp = (node*) malloc(sizeof(node));
  *temp = *data;
}
```

Task3: Insert a new node in a sorted way (GPA, descending)



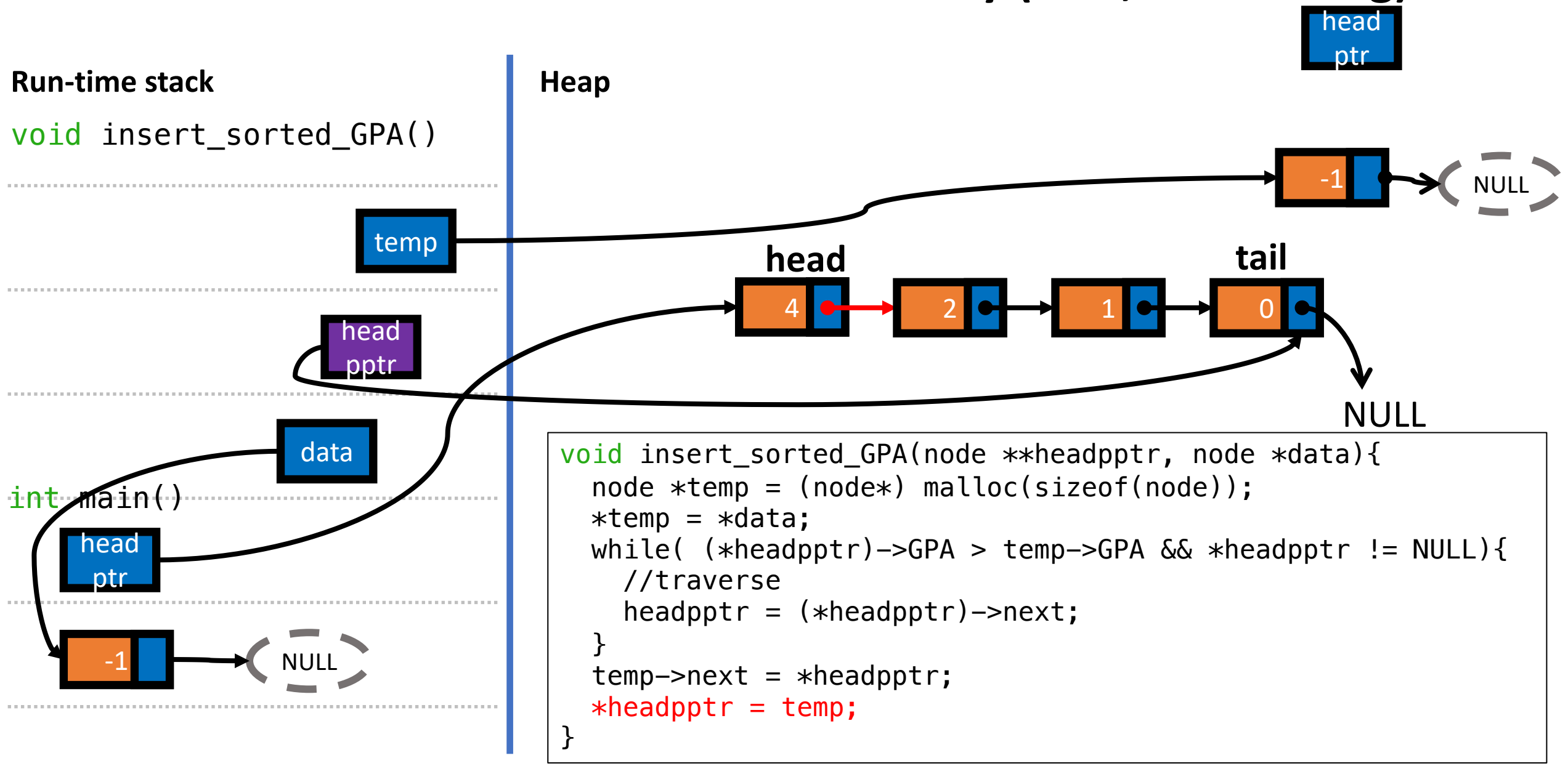

```
void insert_sorted_GPA(node **headpptr, node *data){
  node *temp = (node*) malloc(sizeof(node));
  *temp = *data;
  while( (*headpptr)->GPA > temp->GPA ){
    //traverse
    headpptr = (*headpptr)->next;
  }
  temp->next = *headpptr;
}
```

Task3: Insert a new node in a sorted way (GPA, descending)



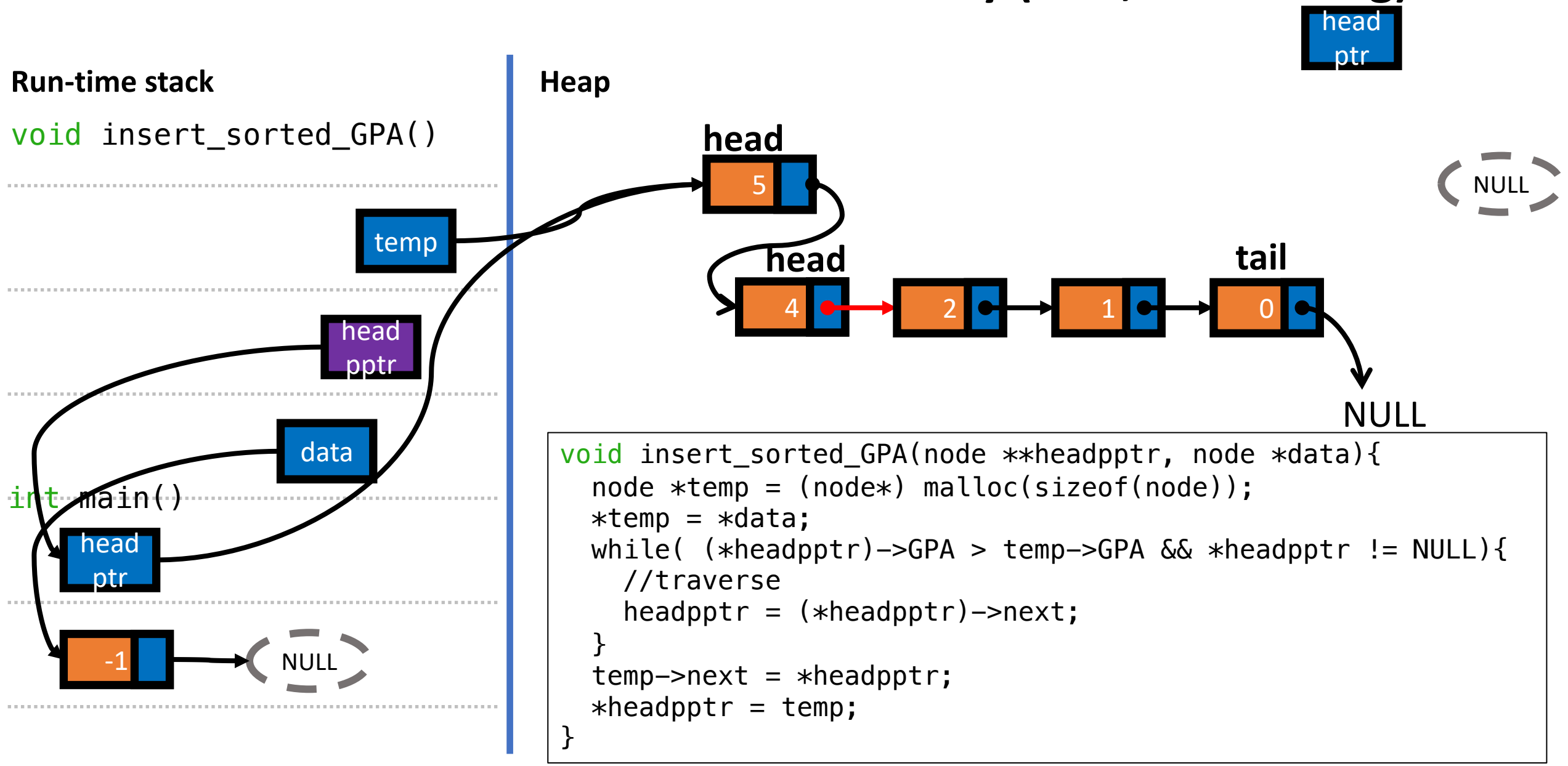

```
void insert_sorted_GPA(node **headpptr, node *data){
  node *temp = (node*) malloc(sizeof(node));
  *temp = *data;
  while( (*headpptr)->GPA > temp->GPA ){
    //traverse
    headpptr = (*headpptr)->next;
  }
  temp->next = *headpptr;
  *headpptr = temp;
}
```

Task3: Insert a new node in a sorted way (GPA, descending)




```
void insert_sorted_GPA(node **headpptr, node *data){
  node *temp = (node*) malloc(sizeof(node));
  *temp = *data;
  while( (*headpptr)->GPA > temp->GPA && *headpptr != NULL){
    //traverse
    headpptr = (*headpptr)->next;
  }
  temp->next = *headpptr;
  *headpptr = temp;
}
```

Task3: Insert a new node in a sorted way (GPA, descending)

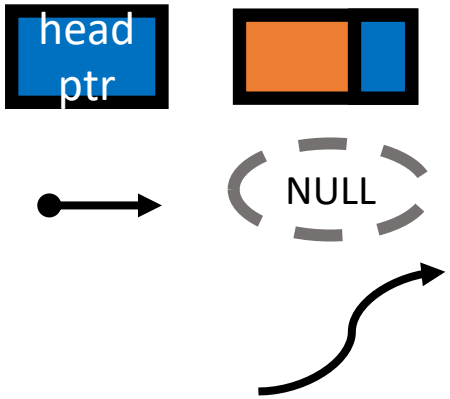


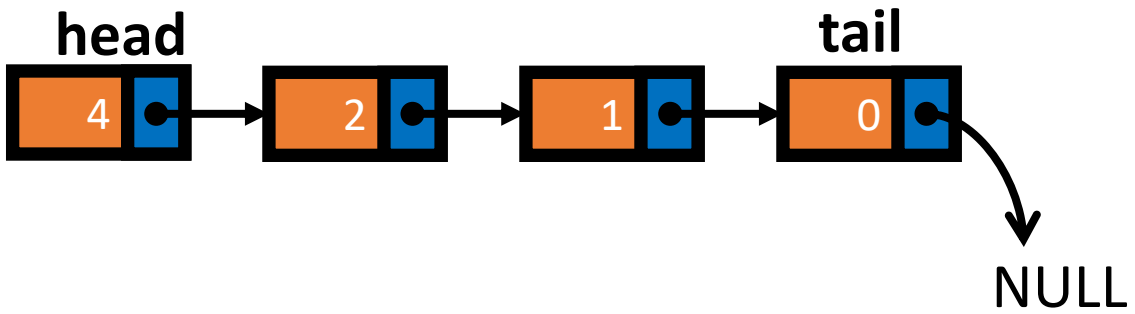

```
void insert_sorted_GPA(node **headpptr, node *data){
  node *temp = (node*) malloc(sizeof(node));
  *temp = *data;
  while( (*headpptr)->GPA > temp->GPA && *headpptr != NULL){
    //traverse
    headpptr = (*headpptr)->next;
  }
  temp->next = *headpptr;
  *headpptr = temp;
}
```

Task4: Delete all nodes

Run-time stack

Heap






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