Forward Linked Lists - Node Operations

- There are two main node operations:

  1. Generating a new node
  2. Deleting a node

- We will define a function with no parameters, named as `Getnode`, and in that function, we will use `malloc` function to allocate enough memory locations for one node within the heap, each time we need to generate a new node. `Getnode` will return the starting address of those locations.

```
/* Get a node from the heap */
node_t *Getnode(void) {
     node_t *node;
     node = (node_t *)malloc(sizeof(node_t));
     node->next = NULL;
     return (node);
}
```

- If we call this function as:

```
p = Getnode();
```

  it will generate a new node and store its address in `p`.

- As you know, `free` function returns memory cells to the heap so they can be reused later in response to calls to `malloc`.

```
free(p);
```

Linked List Operations

- There are three main operations:

  1. Initialization of an empty list
  2. Adding a new node to a list
  3. Deleting a node from a list

- As you have already learned, initialization of an empty list is very simple. Just assign NULL to its head.

```
headp = NULL;
```

Adding a New Node to a List

- The steps of adding a new node to a list are as follows:

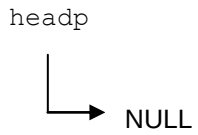
  1. Create a new node (using `Getnode` function)
  2. Assign data to this new node
  3. Link this new node to the list
  4. Link the list to this new node

- We will examine the add operation in four cases:

  1. Adding a node to an empty list
  2. Adding a node between two nodes
  3. Adding a node at the end of a list
  4. Adding a node at the beginning of a list

Example 1: Add a new node, with data 5, to an empty list.

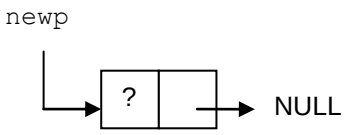
- In an empty list, `headp` points to NULL, as follows.



- We will use `newp` as the address of the new node. We must first of all create it as follows:

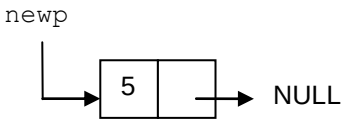
```
node_t *newp;
newp = Getnode();
```

- Now, we created a new node, and `newp` contains the address of this new node, but its `data` member is yet unknown. We can visualize it as follows:



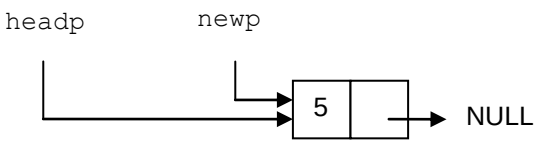
- We need to assign 5 to its `data` member:

```
newp->data = 5;
```



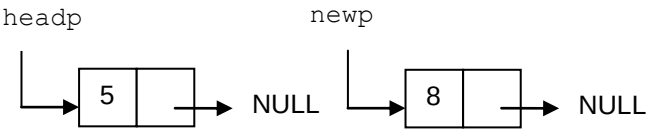
- Now, we have two lists, an empty list `headp`, and a list with one node `newp`. The only thing we need to do is to link them, so that `headp` also points to this node:

```
headp = newp;
```



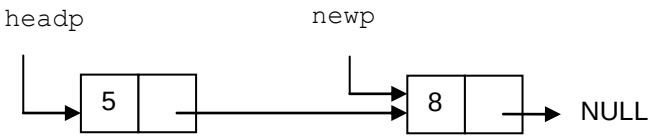
Example 2: Add a second node, with data 8.

```
newp = Getnode();
newp->data = 8;
```

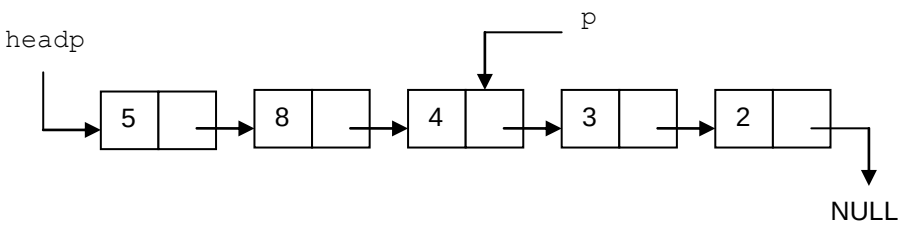


- Now, we have two lists, our previous list `headp`, and a list with one node `newp`. The only thing we need to do is to link them, so that `headp->next` points to this node:

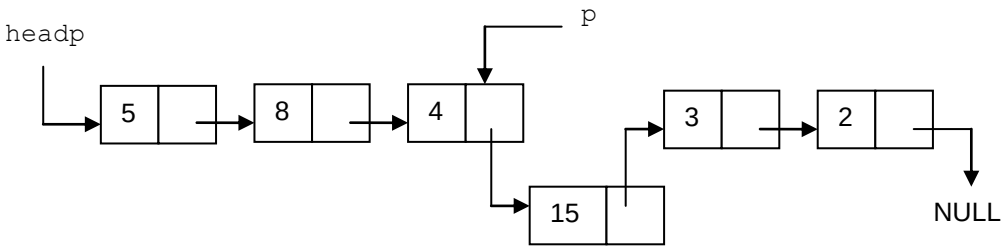
```
headp->next = newp;
```



Example 3: Add a new node, with data 15, after the node pointed by `p` in the following list.



- After the operation the list should look like as follows:




```
newp = Getnode();
newp->data = 15;
newp->next = p->next;
p->next = newp;
```

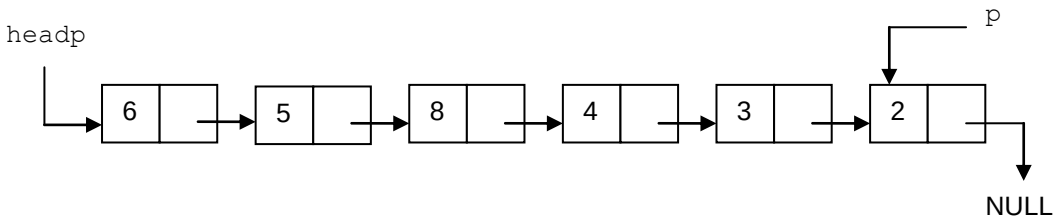
- Notice that, the order of the last two statements can not be changed.

- Let's write this as a function, thus let's define a function that adds a node with a certain data item between two nodes. We need to know the address of the node before the new one, so that we can add the new node after it.

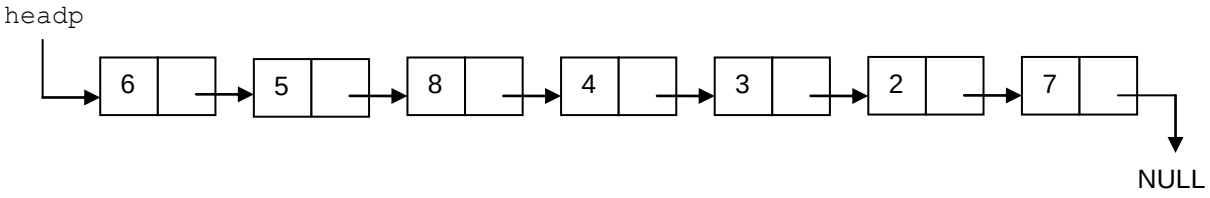
```
void add_after(node_t *p, int item)
{
     node_t *newp;
     /* Create a new node */
     newp = Getnode();
     /* Put the data item into the new node */
     newp->data = item;
     /* Link the new node to the list */
     newp->next = p->next;
     /* Link the list to the new node */
     p->next = newp;
}
```

- We are sure that this function works correctly, if a node is to be inserted between two nodes or to a list with a single node. Let's check, whether it also works correctly, if we want to insert a new node at the end of a list.

- Assume that we want to insert a new node with data 7 to the end of the following list.



- After the operation the list should look like as follows:



- Of course, we should first of all find the address of the last node, so that we can insert the new node after it. Assume that it is found and put in `p`, and the function is called as

```
add_after(p, 7);
```

- After creating a new node and putting 7 as the data item into that node, the `add_after` function assigns `p->next` to `newp->next`. Since `p` shows the last node, `p->next` is NULL. That means, the function will assign NULL to `newp->next`, and then change `p->next` from NULL to `newp`. Therefore, the function also works correctly if we want to insert a new node at the end of a list.

- Notice that, we solved two of the adding problems (second and third ones) with `add_after` function.

- Can we use the same function to solve also the fourth one, thus, to add a new node to the beginning of a list? No, because the function needs `p`, thus the address of the node that should come before the new node, as a parameter. Therefore, we need to define another function for that case, as follows:

```
node_t *add_beginning(node_t *headp, int item)
{
     node_t *newp;
     /* Create a new node */
     newp = Getnode();
     /* Put the data item into the new node */
     newp->data = item;
     /* Link the new node to the list */
     newp->next = headp;
     /* Return the starting address of the new list */
     return (newp);
}
```

- Can we use this function to solve the first adding problem, thus to add a node to an empty list? Yes.

```
headp = add_beginning(headp, 5);
```

- Therefore, when we are dealing with forward linked lists, we need to define two different add functions: `add_beginning`, which adds a new node to an empty list, or to the beginning of any list, and `add_after`, which adds a new node between two nodes, or to the end of any list.

Example: Write a function that creates a list, using the data in an array.

```
/* Creating a list from an array */
node_t *create_list(int ar[], int size) {
     node_t *headp, *p;
     int i;
     headp = NULL;
     headp = add_beginning(headp, ar[0]);
     p = headp;
     for (i = 1; i < size; i++) {
         add_after(p, ar[i]);
         p = p->next;
     }
     return (headp);
}
```

- In the above function, we add the first element of the array to the beginning of the list, the rest to the end. What would happen if we added all of them to the beginning as follows:

```
/* Creating a list from an array */
node_t *create_list2(int ar[], int size)
{
     node_t *headp;
     int i;
     headp = NULL;
     for (i = 0; i < size; i++)
         headp = add_beginning(headp, ar[i]);
     return (headp);
}
```

Home Exercise: Write a program segment to create a linked list using the integer data given by the user. Use 0 as the sentinel value.

Home Exercise: Write a program segment to create a sorted linked list using the integer data given by the user. Use 0 as the sentinel value. Trace your solution for the following sets of input: **5 4 8 7 4 9 5 0**, **3 0**, **0**