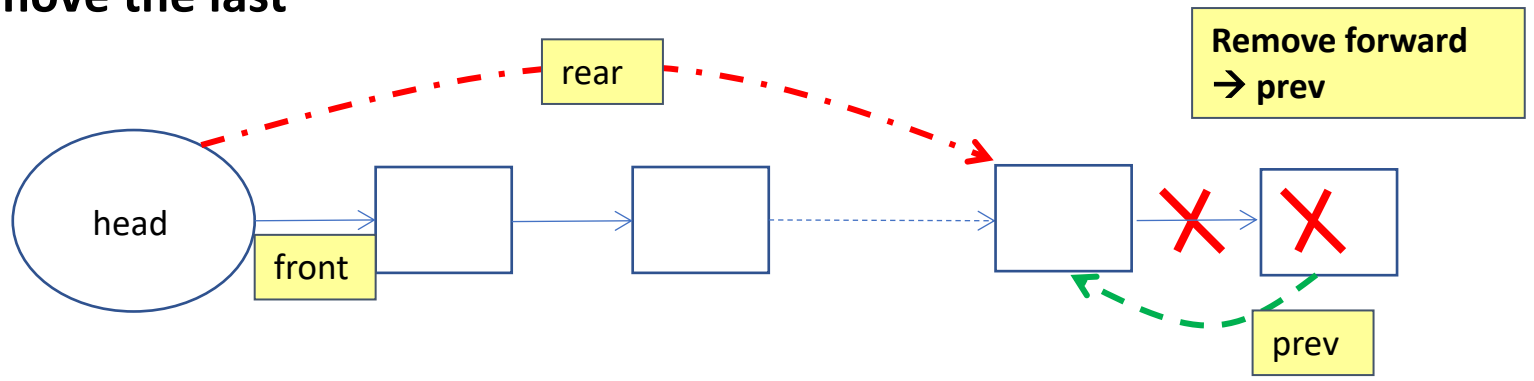
Topic 4: Linked list (remove)

Linked list (remove the last → stack)

Remove the last




```
typedef struct reg{
    int ID;
    int score;
    struct reg *next;
    struct reg *prev;
}tReg;
```

Boundary condition (Add the first or remove the last)

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct reg
{
    int ID;
    int score;
    struct reg *next;
    struct reg *prev;
}tReg;

typedef struct regHead
{
    int count;
    tReg *front;  // (or struct reg)
    tReg *rear;  // (or struct reg)
}tRegHead;

void add_student(tRegHead *head_ptr, int ID, int score);
void rem_student(tRegHead *head_ptr);
```

```c
int main (void)
{
    tReg *stu_ptr;
    tRegHead *head_ptr;
    int i;

    head_ptr=(tRegHead *)malloc(sizeof(tRegHead));
    head_ptr->count = 0;
    head_ptr->front = NULL;
    head_ptr->rear = NULL; // (or head_ptr->front);

    add_student(head_ptr, 20, 40);
    add_student(head_ptr, 52, 100);

    stu_ptr = head_ptr->front;
    for (i =0; i < head_ptr->count; i++)
    {
        printf("ID: %d with score: %d \n",
            stu_ptr->ID, stu_ptr->score);
            stu_ptr = stu_ptr -> next;
    }

    while (head_ptr->count)
    {
        rem_student(head_ptr);
    }
}
```

```c
void add_student(tRegHead *head_ptr, int ID, int score)
{
    tReg *new_stu_ptr;
    new_stu_ptr = (tReg *) malloc (sizeof(tReg));
    new_stu_ptr->ID = ID;
    new_stu_ptr->score = score;
    new_stu_ptr->next = NULL;
    new_stu_ptr->prev = NULL;

     if (head_ptr->count == 0)
    {
        head_ptr->front = new_stu_ptr;
    }
    else
    {
        head_ptr->rear->next = new_stu_ptr;
        new_stu_ptr->prev = head_ptr->rear;
    }
    head_ptr->rear = new_stu_ptr;
    head_ptr->count ++;
}
```

Remove the last

```c
typedef struct reg
{
    int ID;
    int score;
    struct reg *next;
    struct reg *prev;
}tReg;

typedef struct regHead
{
    int count;
    tReg *front;  // (or struct reg)
    tReg *rear;  // (or struct reg)
}tRegHead;

void add_student(tRegHead *head_ptr, int ID, int score);
void rem_student(tRegHead *head_ptr);
```

```c
int main (void)
{
    tReg *stu_ptr;
    tRegHead *head_ptr;
    int i;

    head_ptr=(tRegHead *)malloc(sizeof(tRegHead));
    head_ptr->count = 0;
    head_ptr->front = NULL;
    head_ptr->rear = NULL; // (or head_ptr->front);

    add_student(head_ptr, 20, 40);
    add_student(head_ptr, 52, 100);

    stu_ptr = head_ptr->front;
    for (i =0; i < head_ptr->count; i++)
    {
        printf("ID: %d with score: %d \n",
            stu_ptr->ID, stu_ptr->score);
            stu_ptr = stu_ptr -> next;
    }

    while (head_ptr->count)
    {
        rem_student(head_ptr);
```

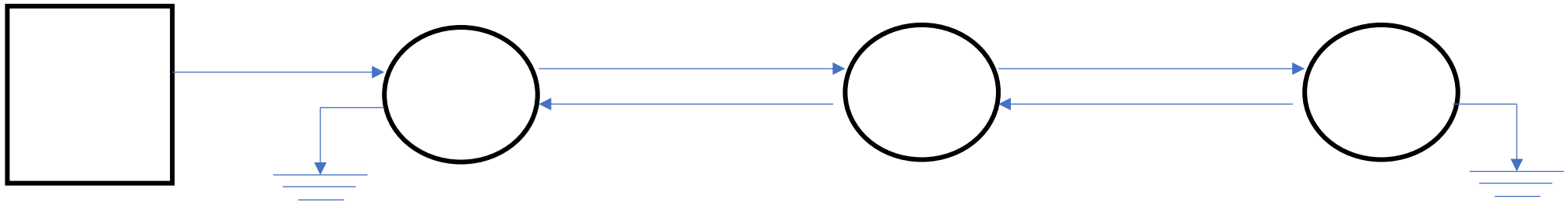
Quiz

```c
void RemStudent (tRegHead *p)
{
    tReg *stu_ptr;

    stu_ptr = XX1;

    if (head_ptr->count > 1)
    {
        head_ptr->rear = XX2;
        head_ptr->rear->next = XX3;
    }
    else
    {
        head_ptr->rear = XX4;
        head_ptr->front = XX5;
    }
    head_ptr->count --;

    printf ("Remove student ID: %d with score: %d \n",
             stu_ptr->ID, stu_ptr->score);

    free (stu_ptr);
}
```

Remove the last

```
void RemStudent (tRegHead *p)
{
    tReg *stu_ptr;

    stu_ptr = head_ptr->rear;

    if (head_ptr->count > 1)
    {
        head_ptr->rear = stu_ptr->prev;
        head_ptr->rear->next = NULL;
    }
    else
    {
        head_ptr->rear = NULL;
        head_ptr->front = NULL;
    }
    head_ptr->count --;

    printf ("Remove student ID: %d with score: %d \n",
             stu_ptr->ID, stu_ptr->score);

    free (stu_ptr);
}
```

Bidirectional linked list: add remove randomly



W9-assignment

Two stages

- First stage:
  - Use your own program last week
  - Modify the print_list function as the right one → your printList function should be the same as mine in this assignment
  - Try first during the class !

```
list->counts: 14
The sorted list: 1 1 2 2 4 4 5 5 8 8 8 8 8 9
The sorted list: 9 8 8 8 8 8 5 5 4 4 2 2 1 1
0
list->counts: 15
The sorted list: 0 1 1 2 2 4 4 5 5 8 8 8 8 8 9
The sorted list: 9 8 8 8 8 8 5 5 4 4 2 2 1 1 0
5
list->counts: 16
The sorted list: 0 1 1 2 2 4 4 5 5 5 8 8 8 8 8 9
The sorted list: 9 8 8 8 8 8 5 5 5 4 4 2 2 1 1 0
```

```c
void print_list(tNumStorHead *list)
{
    tNumStorage *node_ptr;

    printf("\n");
    printf("counts: %d \n", list->counts);
    node_ptr = list->head;
    printf (" From head --> ");
    while (node_ptr != NULL)
    {
        printf("%d ", node_ptr->number);
        node_ptr = node_ptr->next;
    }
    node_ptr = list->tail;
    printf ("\n From tail --> ");
    while (node_ptr != NULL)
    {
        printf("%d ", node_ptr->number);
        node_ptr = node_ptr->prev;
    }
    printf("\n");
}
```

Second stage

- Upload this one only
- Add or remove randomly →→
- Before your delivery, test your program following my procedure !!

```
1. Add a number or 2. Delete a number: 1
Add a number: 1

count: 1
From head --> 1
From tail --> 1

1. Add a number or 2. Delete a number: 1
Add a number: 2
 Specify a target location: 1
 1. Before or 2. After the location 1: 2

count: 2
From head --> 1 2
From tail --> 2 1

1. Add a number or 2. Delete a number: 3
 No such choose !
1. Add a number or 2. Delete a number: 1
Add a number: 3
 Specify a target location: 1
 1. Before or 2. After the location 1: 1

count: 3
From head --> 3 1 2
From tail --> 2 1 3

1. Add a number or 2. Delete a number: 1
Add a number: 4
 Specify a target location: 3
 1. Before or 2. After the location 3: 2

count: 4
From head --> 3 1 2 4
From tail --> 4 2 1 3
```

```
1. Add a number or 2. Delete a number: 1
Add a number: 5
 Specify a target location: 2
 1. Before or 2. After the location 2: 1

count: 5
From head --> 3 5 1 2 4
From tail --> 4 2 1 5 3

1. Add a number or 2. Delete a number: 2
 Specify a target location: 3

count: 4
From head --> 3 5 2 4
From tail --> 4 2 5 3

1. Add a number or 2. Delete a number: 2
 Specify a target location: 4

count: 3
From head --> 3 5 2
From tail --> 2 5 3

1. Add a number or 2. Delete a number: 1
Add a number: 6
 Specify a target location: 3
 1. Before or 2. After the location 3: 2

count: 4
From head --> 3 5 2 6
From tail --> 6 2 5 3

1. Add a number or 2. Delete a number: 2
 Specify a target location: 4

count: 3
From head --> 3 5 2
From tail --> 2 5 3
```

```
1. Add a number or 2. Delete a number: 1
 Add a number: 6
  Specify a target location: 3
  1. Before or 2. After the location 3: 1

count: 4
 From head --> 3 5 6 2
 From tail --> 2 6 5 3

1. Add a number or 2. Delete a number: 2
  Specify a target location: 1

count: 3
 From head --> 5 6 2
 From tail --> 2 6 5

1. Add a number or 2. Delete a number: 2
  Specify a target location: 2

count: 2
 From head --> 5 2
 From tail --> 2 5

1. Add a number or 2. Delete a number: 1
 Add a number: 6
  Specify a target location: 2
  1. Before or 2. After the location 2: 2

count: 3
 From head --> 5 2 6
 From tail --> 6 2 5
```

```
1. Add a number or 2. Delete a number: 2
  Specify a target location: 2

count: 2
 From head --> 5 6
 From tail --> 6 5

1. Add a number or 2. Delete a number: 1
 Add a number: 9
  Specify a target location: 2
  1. Before or 2. After the location 2: 1

count: 3
 From head --> 5 9 6
 From tail --> 6 9 5

1. Add a number or 2. Delete a number: 2
  Specify a target location: 1

count: 2
 From head --> 9 6
 From tail --> 6 9

1. Add a number or 2. Delete a number: 2
  Specify a target location: 1

count: 1
 From head --> 6
 From tail --> 6

1. Add a number or 2. Delete a number: 2
  Specify a target location: 1

count: 0
 From head -->
 From tail -->
```

```
1. Add a number or 2. Delete a number: 1
 Add a number: 1

count: 1
 From head --> 1
 From tail --> 1

1. Add a number or 2. Delete a number: 1
 Add a number: 2
  Specify a target location: 1
  1. Before or 2. After the location 1: 2

count: 2
 From head --> 1 2
 From tail --> 2 1

1. Add a number or 2. Delete a number: 1
 Add a number: 3
  Specify a target location: 1
  1. Before or 2. After the location 1: 1

count: 3
 From head --> 3 1 2
 From tail --> 2 1 3
```