# CSI2P(I) 2019 Spring

Final Exercise

# 12145 - Species of Knuckles

# Problem

- Sample Input:
  - 5
  - abcca
- Sample Output:
  - I'm the god of Knuckles!

- **aa**b -> bbb
- **aa**bcc -> bbbcc -> bbbbb
- If any character appears above twice, the answer is YES !

# Hint

- Problem has memory limitation
  - Declare one 26-dimension array representing characters’ frequency


  - ‘a’ ~ ‘z’
- After processing input, count each element of array and output answer

# 12146 - Too Many Things to Buy

# Problem

- Input:
  - n : how many items
  - k : must buy at least k item **at first day**
  - Item’ s prices at first day
  - Item’ s prices at second day
- Output:
  - Miuimum price

# Problem

- Sample Input:
  - 5 2
  - 6 8 9 5 4
  - 5 1 10 3 2
- Sample Output:
  - 21

- In the end, you still have to buy all items
- At first day, you have to buy **k** items that is **cheaper** than ones at second day in order to save money
- We have to sort items according to its **relative prices** between two days -> Buy first k cheaper items at first day

| First day | **6** | 8 | **9** | 5 | 4 |
|---|---|---|---|---|---|
| Second day | 5 | **1** | 10 | **3** | **2** |
| Relative price | **-1** | **-7** | **1** | **-2** | **-2** |

# 12152 - youbike racer

# 12152 - youbike racer

Input:

6 5 ---> 6 checkpoint, able to pass through 5 units

2 4 7 8 9 14 ---> Location of each checkpoint

A

Output:

2

# 12152 - youbike racer


- A is able to pass checkpoint 4, however, it is impossible to reach checkpoint 7.
  - 5 > 2, no need to stop to change to a new bike
  - 5 > 4, same reason as above
  - 5 < 7, cannot reach 7
  - We have to change bike at the previous checkpoint, i.e., checkpoint 4
  - Now, the bike have the ability to ride (4 + 5) units.
  - 9 < 9, no need to change
  - 9 < 14, we cannot reach 14 unless the bike is a new one
  - Change the bike at its previous checkpoint, i.e., checkpoint 9
  - 9 + 5 = 14, we’ re good to go.

# 12152 - youbike racer

```
int pos = k, ans = 0, flag = 1;
for(int i = 0 ; i < n-1 ; i++)
    if( ckpt[i+1] - ckpt[i] > k ){
        flag = 0;
        break;
    }
if( ckpt[0] > k || !flag ){
    printf("The Legend Falls.\n");
    return 0;
}
for(int i = 1 ; i < n ; i++){
   if( pos < ckpt[i] ){
        pos = ckpt[i-1] + k;
        ans++;
    }
}
printf("%d\n", ans);
```

# 12289 - after rain

# 12289 - after rain

Functions:

insert <color> <dest> :  insert Black 13 ---> insert Black after the 13-th location

erase1 <dest> :  erase1 4 ---> erase the 4-th in the sequence

erase2 <color> : erase2 White ---> erase all white in the sequence

reverse <dest1> <dest2> ---> { ‘A’ , ‘B’ , ‘C’ } -> { ‘C’ , ‘B’ , 'A'}

~~show ---> a simple show function~~

# 12289 - after rain

Focus on the limitation on each functions.

For example,

“erase1 n”，what if n is larger than the length of the sequence?

……

See 12289 for more details.

# Insert

Data Structure: Linked list, Method: insert

```
void insert(Node** head, char* in, int id){
    Node *newN = (Node*)malloc(sizeof(Node));
    strcpy(newN->color, in);
    newN->next = NULL;
    Node *ptr = *head;
    for(int i = 0 ; i < id && ptr->next != NULL ; i++){
        ptr = ptr->next;
    }
    newN->next = ptr->next;
    ptr->next = newN;
}
```

# Erase1

Method: remove

```
void erase1(Node** head, int id){
    Node *ptr = *head, *prev = NULL, *del;
    for(int i = 0 ; i < id && ptr->next != NULL ; i++){
        prev = ptr;
        ptr = ptr->next;
    }
    if(prev == NULL) return;
    del = ptr;
    prev->next = ptr->next;
    free(del);
}
```

# Erase2

```
void erase2(Node** head, char* color){
    Node *ptr = *head, *prev;
    while(ptr != NULL){
        if( !strcmp(ptr->color, color) ){
            Node *del = ptr;
            prev->next = ptr->next;
            free(del);
            ptr = prev;
        }
        prev = ptr;
        ptr = ptr->next;
    }
}
```

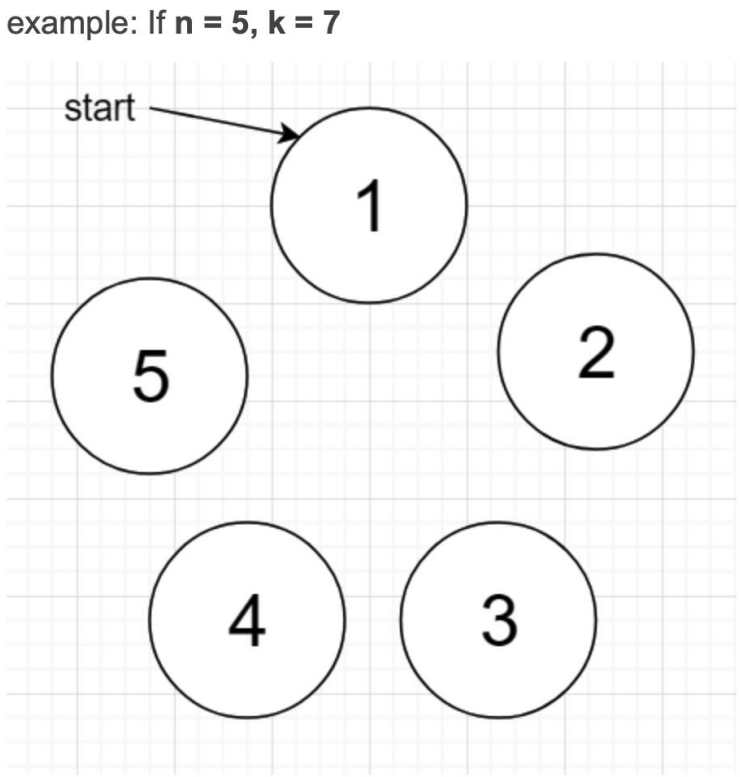
# Reverse

```
void reverse(Node** head, int id1, int id2){
    Node *ptr1 = *head, *ptr2 = *head, *prev1, *temp = NULL, *end;
    for(int i = 0 ; i < id1 && ptr1->next != NULL ; i++){
        prev1 = ptr1;
        ptr1 = ptr1->next;
    }
    for(int i = 0 ; i < id2 && ptr2->next != NULL ; i++){
        ptr2 = ptr2->next;
    }
    end = ptr2->next;
    while(1){
        temp = ptr1->next;
        ptr1->next = end;
        end = ptr1;
        ptr1 = temp;
        if(end == ptr2) break;
    }
    prev1->next = end;
```

# 12301 - Uncle Huang choose Tutor (Easy version)

# Joseph problem

- Given number of people and the steps you go to kill people



# Joseph problem

- Using link list

```
Node* createList(int n){
    Node *head, *now;
    for(int i = 1; i <= n; i++) {
        Node *newN = (Node*)malloc(sizeof(Node));
        if( i == 1 ) {
            head = newN;
            now = newN;
            newN->number = i;
            newN->next = head;
        } else {
            newN->number = i;
            newN->next = head;
            now->next = newN;
            now = newN;
        }
    }
    return head;
}
```

# Joseph problem

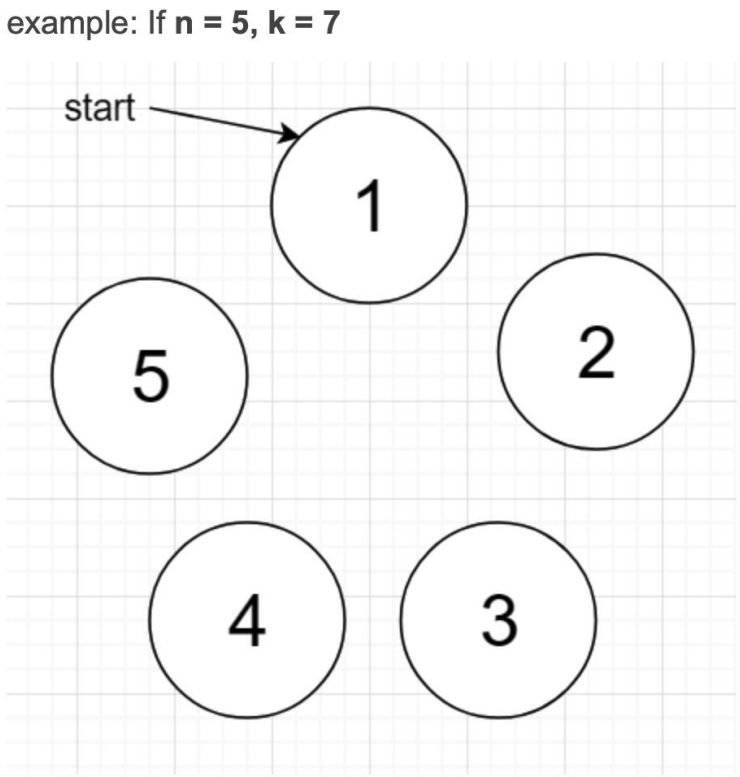
- Using link list

```
int solveJosephus(Node **head, int step){
    int len = 0;
    Node *cal = (*head)->next, *prevN =(*head);
    while( cal != (*head) ){
        len++;
        cal = cal->next;
    }
    len++;
    while( len > 1 ){
        int move = (step-1) % len + 1;
        move--;
        while(move--) (*head) = (*head)->next;
        while(prevN->next != (*head)) prevN = prevN->next;
        Node *kill = (*head);
        Node *nextN = (*head)->next;
        prevN->next = nextN;
        (*head) = nextN;
        free(kill);
        len--;
    }
    return (*head)->number;
```

# 12301 - Uncle Huang choose Tutor (Hard version)

# Joseph problem

- Given number of people and the steps you go to kill people



# Joseph problem

- Using DP

```
while( scanf("%d%d", &n, &k)!=EOF ){
    int pos = 1;
    for(int i = 2 ; i<=n ; i++){
        pos = ( pos + k - 1 ) % i + 1;
    }
    printf("%d\n", pos);
}
```

# Joseph problem

- Using DP
  - We suppose that we have n people and m steps
  - Initially, we have people 0~n-1(f(n,m))
  - After the first round, the kth people be killed(k = (m - 1) % n)
  - When the second round begin, k+1 = 0(f(n-1,k))
  - k+1 = 0 , k+2 =1 …n-1= n-k-2, 0 = n-k-1 , …… k-1 = n-2
  - f(n,m) = (f(n-1 , m) + k + 1)%n
  - f(n,m) = (f(n-1 , m) + m )%n

# Joseph problem

- Using DP
  - f(n,m) = (f(n-1 , m) + m )%n

```
while( scanf("%d%d", &n, &k)!=EOF ){
    int pos = 1;
    for(int i = 2 ; i<=n ; i++){
        pos = ( pos + k - 1 ) % i + 1;
    }
    printf("%d\n", pos);
}
```

# 12303 - Operation on Sequence

- You have a sequence `a`. Initially, `a` has exactly one integer. You're at the place of the `1st` element.
- There are some operations below:
  - `insert <val1> <val2>`: insert `<val2>` number of elements, all with value `<val1>`. Insert them next to your position.
  - `erase <val>`: erase `<val>` number of elements next to you.
  - `move <value>`: Move `<value>` number of indexes forward. Note that `<value>` might be negative, which means you might move forward or backward.
  - `show`: Start from your position, output the sequence `a`. Each element separated by a space. Note that there should be no space at the end but a `'\n'`.

For example: `a` = `{2}`, and execute operations below:

- `insert 3 6` // `a` = `{2,3,3,3,3,3,3}`, you're at the 1st position.
- `insert 1 1` // `a` = `{2,1,3,3,3,3,3,3}`, you're at the 1st position.
- `erase 2` // `a` = `{2,3,3,3,3,3}`, you're at the 1st position. Erase `1` and `3`.
- `move 5` // `a` = `{3,2,3,3,3,3}`, you're at the 1st position. Originally was the 6th position.
- `erase 3` // `a` = `{3,3,3}`, you're at the 1st position. Erase the first `2` and two `3`.
- `show` // print `3 3 3`. Note that there should be a `'\n'` at the end.

# Using Linked List

```c
typedef struct Node{
    int num;
    struct Node *prev;
    struct Node *next;
}Node;
```

```c
Node *creatnode(int n){
    Node *newN = (Node*)malloc(sizeof(Node));
    newN->num = n;
    newN->next = newN;
    newN->prev = newN;
    return newN;
}
```

```c
void insert(int num, int times){
    Node *nowN = head;
    Node *fin = head->next;
    for(int i = 0 ; i < times ; i++){
        // insert "times" nodes
    }
    nowN->next = fin;
    fin->prev = nowN;
}
```

```c
void erase(int times){
    Node *nowN = head->next;
    for(int i = 0 ; i < times ; i++){
        // delete "times" nodes
        // remember to use
    }
    head->next = nowN;
    nowN->prev = head;
}
```

# Using Linked List

```
void move(int step){
    step = (step - 1)%n + 1;
    if( step > 0 ){
        while(step--)
            head = head->next;
    }else{
        step = -step;
        while(step--)
            head = head->prev;
    }
}
```

```
void show(){
    Node *nowN = head->next;
    printf("%d", head->num);
    while( nowN != head ){
        printf(" %d", nowN->num);
        nowN = nowN->next;
    }
    printf("\n");
}
```

# 12305 - Airplane Shooter

- Given a sequence `a`. The elements of `a` has the following infomations:
    - `index`: The index is followed by the input order, starts from 1. The input won't contain `index`, you have to record it yourself. **Smaller input `index` has higher priority**.
    - `admin level`: Level starts from 0 to 999. **level 0 has the highest priority, while level 999 has the lowest.**
    - `license number`: An integer. **Smaller number has higher priority.**
- You are going to sort `a`. Element with higher priority brings to the front. Compare `admin level` first, then `license number`, then `index`.
- Output `index` (the old one) of every element in the new `a`.

**Easy qsort problem without any other tricks ……**