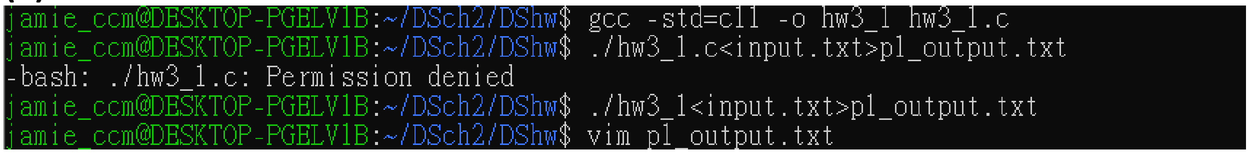
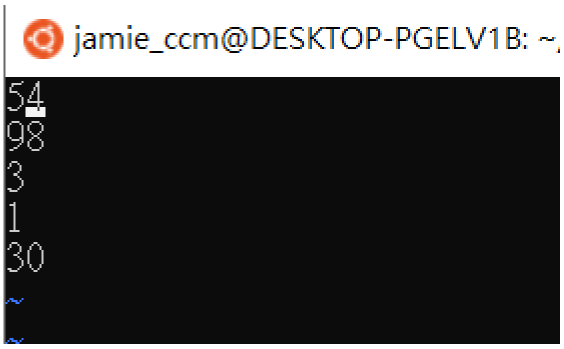
HW3_1

(1) result screenshot



```
jamie_ccm@DESKTOP-PGELV1B:~/DSch2/DShw$ gcc -std=c11 -o hw3_1 hw3_1.c
jamie_ccm@DESKTOP-PGELV1B:~/DSch2/DShw$ ./hw3_1.c<input.txt>p1_output.txt
-bash: ./hw3_1.c: Permission denied
jamie_ccm@DESKTOP-PGELV1B:~/DSch2/DShw$ ./hw3_1<input.txt>p1_output.txt
jamie_ccm@DESKTOP-PGELV1B:~/DSch2/DShw$ vim p1_output.txt
```

Screenshot of command line.

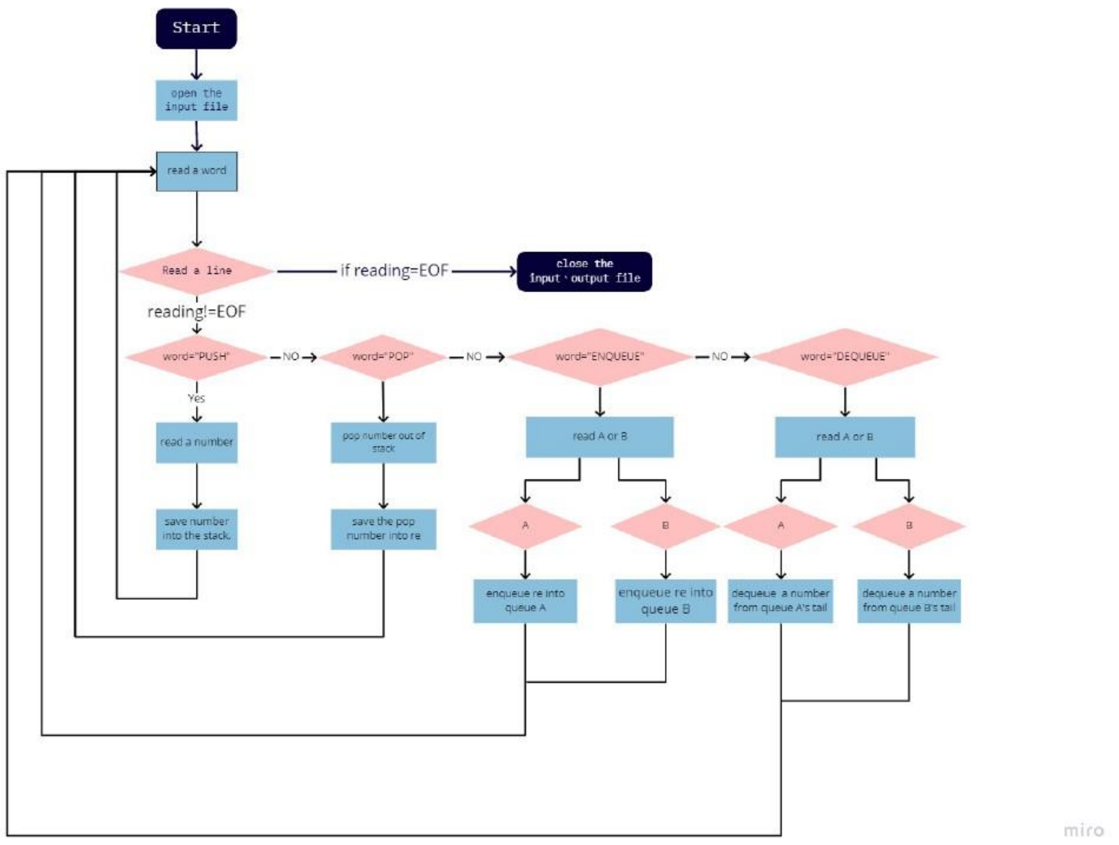


```
jamie_ccm@DESKTOP-PGELV1B: ~,
54
98
3
1
30
~
~
```

Screenshot of “p1_output.txt”.

(2) program architecture

1. Read the file once line a word.
2. The first line of a word would be “PUSH”, “POP”, “ENQUEUE” or “DEQUEUE” without wrong, so check the word we read is which one.
3. If the word is not four of then, it will print out “Undefined command.” on the output file.
4. If the word is PUSH, read the word (number) behind it and convert it into integer, and push the integer into the stack.
5. If the word is POP, pop the number in the top of the stack, and record the number in a variable.
6. If the word is ENQUEUE, read the word (it would be either A or B if nothing wrong), and enqueue the record variable (the number you save in step 5) in to the head of queue you want.
7. if the word is DEQUEUE, read the word (A or B), and dequeue it from the tail of the queue you want. At the same time, print the number you dequeue on the output file.
8. Redo the step 1 to 7,until the end of the file (EOF).
9. End the program.



(3) program function

```
struct stackNode
{
    int index;
    struct stackNode* nextptr;
};
typedef struct stackNode stack;
typedef stack * stackPtr;
```

stackNode- Define the structure stack, each of them have a space to save data (the number print on the stack in question.), and a pointer to pointed out its last or next data.
type define- we called **stackNode** as **stack**, and **stackNode*** as **stackPtr**.
So does the structure queue.

```
int re,count;
FILE *fpi;
```

Parameter:
re- Used to record the number popped out of the stack, so that it could be enqueue later.
count- to record the line we already print. Since we won't change line before we print anything, it will use in **dequeue**.

```
bool IsEmptyS(stackPtr top)
{
        return top==NULL;
}
```

IsEmptyS- a function used to check if the stack is empty or not, the **bool IsEmptyQ(queuePtr head)** represents the same meaning.

Parameter **top** usually means the **stacktop**. (It could be the queue's head in **IsEmptyQ**.)

```
stackPtr getNewStack(int value)
{
        stackPtr newPtr = (stackPtr) malloc(sizeof(stack));
        if ( newPtr != NULL )
        {
                newPtr->index = value;
                newPtr->nextptr = NULL;
        }
        if (newPtr == NULL)
                printf("Didn;t have enough space to allocate.\n")
        return newPtr;
}
```

getNewStack- a function use to allocate a **stackNode** space and set a new data in that **stackNode**. So does the function **getNewQueue(int value)**.
Parameter:
value- the variable we want to store. This value is original from function **pushStack**.
newPtr- A new **stackNode** we create. The **stackNode**'s value is the number we read after "PUSH", and it hasn't pointed out its next data.
If **newPtr==NULL**, this means it didn't allocate a space. This will be warning at the function **pushStack**.
Return value: The function will return the **newPtr**, so we can assign its nextptr later.

```
void pushStack(int val,stackPtr *sptr)
{
            stackPtr newPtr = getNewStack(val);
            if (newPtr==NULL)
              printf("WRONG");
            else
        {
                    newPtr->nextptr=*sptr;
                    *sptr=newPtr;
        }
}
```

pushStack- a function use to insert a stacknode into the big stack.
newPtr==NULL- which means function **getNewStack** didn't allocate a space. But it may have space after the pop command, so the program could still read next word.
If the allocate is successful, we will assign the **newPtr**'s **nextptr** to the ***sptr**(which usually means the **stacktop**.), then the **stacktop** would pointed to **newPtr**.
For example, the original top is pointed to 21, the new value is 34. Then 34 would pointed to 21, and stacktop would pointed to 34. It would be top->34->21->others.
Parameter:
newPtr- the parameter made by **getNewStack**.
val- the variable we get from outside, the data we want to push in. The number we read after "PUSH".
***sptr**- the pointer we want to pointed to the new data, in this case, it is **stacktop**.

Return value: the function didn’t have return value.

```
void popStack(stackPtr *Sptr)
{
	if(IsEmptyS(*Sptr))
		printf("Stack is empty.\n");
	else
	{
		stackPtr tempPtr=*Sptr;
		re=(*Sptr)->index;
		*Sptr=(*Sptr)->nextptr;
		free(tempPtr);
	}
}
```

popStack- a function use to popped out a **stackNode**.
We need to check if the stack is empty or not, if the stack is empty we can’t pop.
If the stack is not empty, we can pop a **stackNode**, we need to use a variable to save the popped number, and we change the **stacktop**, let it pointed to the **stacktop**’s **nextptr**.
For example, if the original is top->23->31, the variable would record 23, and top would point to 31. finally, the program would release the space of 23.
Parameter:
re- the global parameter used to record the pop number, we will use it in the next loop.
***Sptr**- the pointer pointed to the number we want to delete. In this case, it is **stacktop**.
Return value: the function didn’t have return value.

```
void enqueue(queuePtr *Qtail,queuePtr *Qhead)
{
	queuePtr newPtr=getNewQueue(re);
	if(newPtr==NULL)
		printf("WRONG");
	if(IsEmptyQ(*Qhead))
	{
		*Qhead=newPtr;
	}
	else
	{

		(*Qtail)->nextptr=newPtr;
	}
		*Qtail=newPtr;

}
```

enqueue- the function use to insert a **queueNode** into queue.
In the beginning, it is the same as **pushStack**, however, we need to check out if the queue’s head is empty or not. If it is empty, let it point out to the first **queueNode**.
No matter the queue’s head is empty or not, we need to let the queue tail’s **nextptr** pointed to **newPtr** (The new **queueNode** we build.), and then queue’s tail would pointed to **newPtr**.

For example, if the original is head->…43->32<-tail, if the number you want to insert is 56, you would let 32 pointed to 56, and let tail pointed to 56. It would be head->…43->32->56<-tail.
Parameter:
re- **record** is the global parameter we get from **popStack**, in the question, we will **enqueue** immediately after we pop something out.
***Qtail**- the tail of queue, it will be **Atail** or **Btail** since we have two queue.
***Qhead**- the head of queue, like the head of line. It will be **Ahead** or **Bhead**.
newPtr- the parameter made by **getNewQueue**.

```
void dequeue(queuePtr *Qhead)
{
        if(IsEmptyQ(*Qhead));
        else if(count==0)
        {
              queuePtr tempPtr=*Qhead;
              *Qhead=( *Qhead )->nextptr;
              printf("%d",tempPtr->plateIndex);
              free(tempPtr);
              count++;

        }
        else
        {
              printf("\r\n");
              queuePtr tempPtr=*Qhead;
              *Qhead=( *Qhead )->nextptr;
              printf("%d",tempPtr->plateIndex);
              free(tempPtr);
              count++;
        }}
```

dequeue- the function use to remove a **queuNode** from a queue.
It almost the same as **popStack**, but you need to print out the data you **dequeue** since this is the question request.
Parameter:
***Qhead**- the head of queue, since queue is FIFO, you should remove from front and insrt at tail.
count- to print the correct line change, we will change line before every output expect count==0.

```
      stackPtr stacktop = NULL;
      queuePtr Ahead = NULL;
      queuePtr Atail = NULL;
      queuePtr Bhead = NULL;
      queuePtr Btail = NULL;
      char str[20]={0};
      int num;
      char aORb;
```

Parameter:

stacktop- stack only have a entry, so you just need top. And it the beginning it should pointed out to nothing.
Ahead,Atail- this two are a couple, queue has two entry, one for in and one for out. They didn't point to anything in the beginning.
Bhead,Btail- the same as **Ahead, Atail**.
str[20]-build a array used to save the command we read, than we can compared them.
num- the integer to save the data we read in.
aORb- the character use to save the word after **enqueue** or **dequeue**, to ensure which queue the system want to need.

```
        if (strncmp(str,"PUSH",4)==0)
        {
            fscanf("%s",str);
            num=atoi(str);
            pushStack(num, &stacktop);
        }
```

Used to check if the word we read is "PUSH". If it is, read the number after it and push it into stack. The same as other three if statement.

```
            switch (aORb)
            {
                case 'A':
                    dequeue(&Ahead);
                    break;
                case 'B':
                    dequeue(&Bhead);
                    break;
                default:
                    break;
            }
```

After we read a character after enqueuer or dequeuer, we than call queueA or queueB, if we read neither of them, it would print wrong message in the output file, but it could still read next command.

If the reading is not four of this, it would skip and read the nextline.

```
    return 0;
```

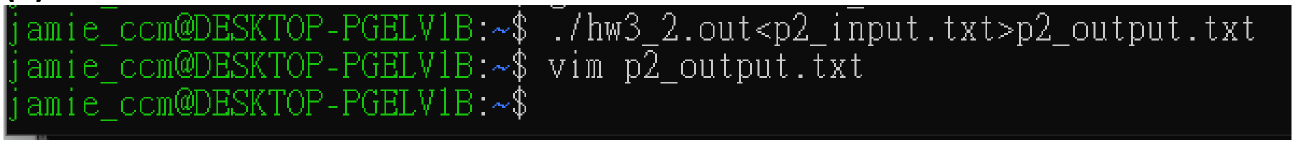
Return value- the return value of main function is 0, since it would end successfully if it encounters EOF.

(4) how you design your program

Since I use linked list in hw2, I didn't change my main program in hw3_1. However, I find some redundant in my program. First, I didn't need to send anything then I can use parameter re since it is a global parameter. While, in hw2 I pass it into function pop and enqueue, so this time I edit it. Next, I delete the parameter fpi since the input file will be restrict if I use it. I didn't know this before, I am so sorry if I let you confuse if you test my hw2 program.

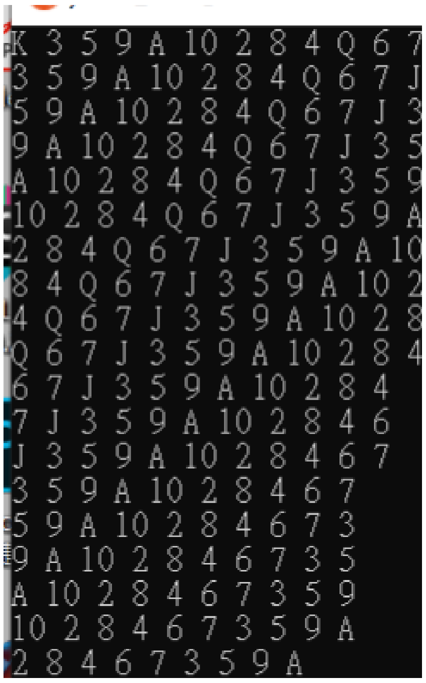
Hw3_2

(1) result screenshot

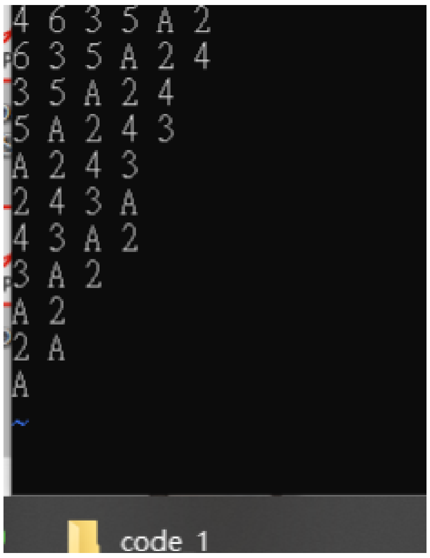


```
jamie_ccm@DESKTOP-PGELV1B:~$ ./hw3_2.out<p2_input.txt>p2_output.txt
jamie_ccm@DESKTOP-PGELV1B:~$ vim p2_output.txt
jamie_ccm@DESKTOP-PGELV1B:~$
```

Screenshot of command line.



```
K 3 5 9 A 10 2 8 4 Q 6 7
3 5 9 A 10 2 8 4 Q 6 7 J
5 9 A 10 2 8 4 Q 6 7 J 3
9 A 10 2 8 4 Q 6 7 J 3 5
A 10 2 8 4 Q 6 7 J 3 5 9
10 2 8 4 Q 6 7 J 3 5 9 A
2 8 4 Q 6 7 J 3 5 9 A 10
8 4 Q 6 7 J 3 5 9 A 10 2
4 Q 6 7 J 3 5 9 A 10 2 8
Q 6 7 J 3 5 9 A 10 2 8 4
6 7 J 3 5 9 A 10 2 8 4
7 J 3 5 9 A 10 2 8 4 6
J 3 5 9 A 10 2 8 4 6 7
3 5 9 A 10 2 8 4 6 7
5 9 A 10 2 8 4 6 7 3
9 A 10 2 8 4 6 7 3 5
A 10 2 8 4 6 7 3 5 9
10 2 8 4 6 7 3 5 9 A
2 8 4 6 7 3 5 9 A
```



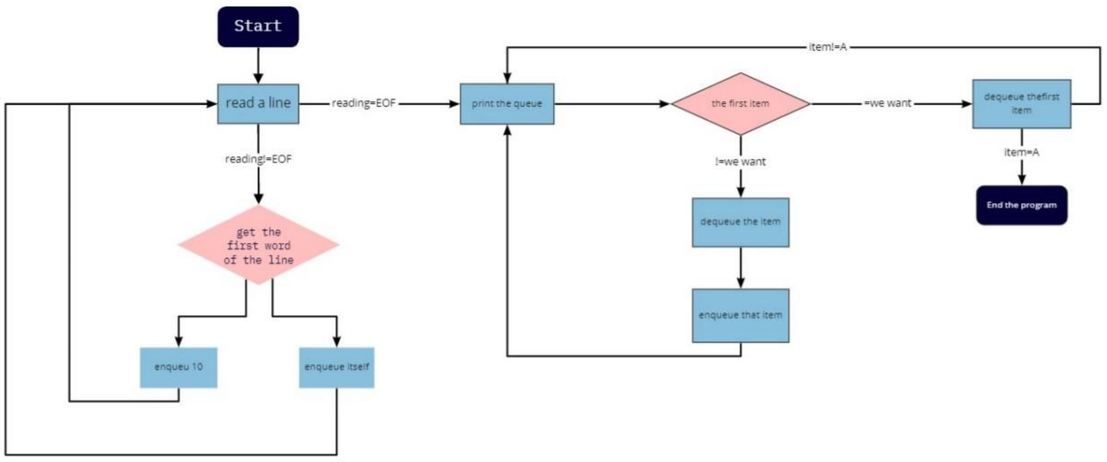
```
4 6 3 5 A 2
6 3 5 A 2 4
3 5 A 2 4
5 A 2 4 3
A 2 4 3
2 4 3 A
4 3 A 2
3 A 2
A 2
2 A
A
~
```

Screenshot of “p2_output.txt” (since the output is too long, I only put part of the output.)

(2) program architecture

1. Read the file once line a word.
2. Get the first word of the line.
3. If the word is “1”, enqueue 10 (in integer).
4. Else, enqueue the word.

5. Redo step 1~4, until the end of file.
6. print the queue.
7. if the first item in queue is the item we want. (We want K Q J 10 9 8 7 6 5 4 3 2 A in terms.), dequeue the item. you want.
8. If it is not the item we want, dequeue it first and re enqueue it into the queue's tail.
9. Redo step 6~8 until the item we dequeuer is A when we do step 7.
10. End the program.



(3) program function

```
void enqueue(int);
void dequeue();
queueptr getnewqueue(int);
bool IsEmpty(queueptr);
```

These function and structure queue is similar to the queue in HW3_1, I won't discuss them more. The only point need to notice is that it will **exit** if there didn't have enough space to enqueuer.

```
void printqueue()
{
        queueptr tempptr=Chead;
        do
        {
           if(tempptr->next==NULL)
              {
                    if(tempptr->data==10)
                          printf("%d",tempptr->data);
                    else
                          printf("%c",tempptr->data);
              }
```

```
            else
            {
                if(tempptr->data == 10)
                    printf("%d ", tempptr->data);
                else
                    printf("%c ", tempptr->data);
            }
            tempptr=tempptr->next;
        }while(tempptr!=NULL);
}
```

printqueue- The function help use print the whole queue. Use a **queueptr temptr** to record where we are. After we print out, move to next item by **queueptr next in the queue**. In the end we will pointed to **NULL**, since the last one in queue pointed to **NULL**. Notice that we store "10" in integer, and other in ASCII code (such as 'A'), so we need to use two different type to print it.
I also judge if it is the end of the line to incase it print one more space.

```
int card[14]={'A','2','3','4','5','6','7','8','9',10,'J','Q','K'};
queueptr Chead=NULL;
queueptr Ctail=NULL;
```

Parameter:
card[14]- the array used to show the character on the card. We save them in integer type, because since 10 is combine by 1 and 0, we can save it in one character.
Chead- the head of the queue, since the function only use one queue, we can declared it as global parameter. Thus, we don't need to pass it.
Ctail- the tail of the queue.

```
    char record[10];
    int value;
    while(scanf("%s",record)!=EOF)
    {
       value=record[0];
        if((int)value==49)
        {
            enqueue(10);
        }
       else if((int)value=='A'||(int)value=='2'…)
            enqueue(value);
   }
```

Almost every word save its ASCII code and enqueuer by value expect 10. We will find it by value=1, and enqueuer it in integer.
In the end, I also judge if the input is these 13 word or not, if it isn't, we will do nothing.
Parameter:
record[10]-when we use array, we can read a line in a time, and we can use the first word in the array to determine which data we want to save. It is in char type because we need to read A, J, Q, K.

value- we use this to get the first word in the line.

```
int count=0,c=0;
do {
    value=Chead->data;
    if(c!=0)
    printf("\n");
    if(value==card[12-count])
    {
        printqueue();
        dequeue();
        count=count+1;
    }
    else
    {
        printqueue();
        dequeue();
        enqueue(value);
    }
    c++;
}while(count<13);
```

This is used to do the output. The queue will be print before every loop Then, it will be compare, if it is the item we want dequeue it. Else, deueue and enqueuer so it will move to the last.

Parameter:

count- let us check the card more convenience, also another kind of counter. When we count to 13, we can end the loop.

c- the line we already print out. To print the correct line change, we will change line before every output expect count==0.

```
return 0;
```

Return value- the return value of main function is 0, since it would end successfully if it count=13, which means we dequeue all out.

(4) How you design your program

The biggest problem of this program is about how to save number 10, I want to use char at first, but I figure out 10 is not in ASCII code when I test my program. Then, I tried to save as integer, but I will confuse that I should read them in integer or character. One of the way I tried is that I read them in 1 char, when I meet number 1, I read one more char and do nothing. But it is not the best way since sometimes I need to input more than a word and I don’t know why. Finally, I scan a string and get the first word to judge.