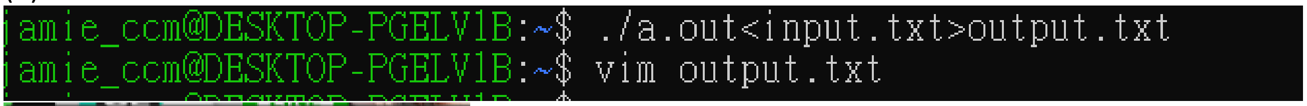
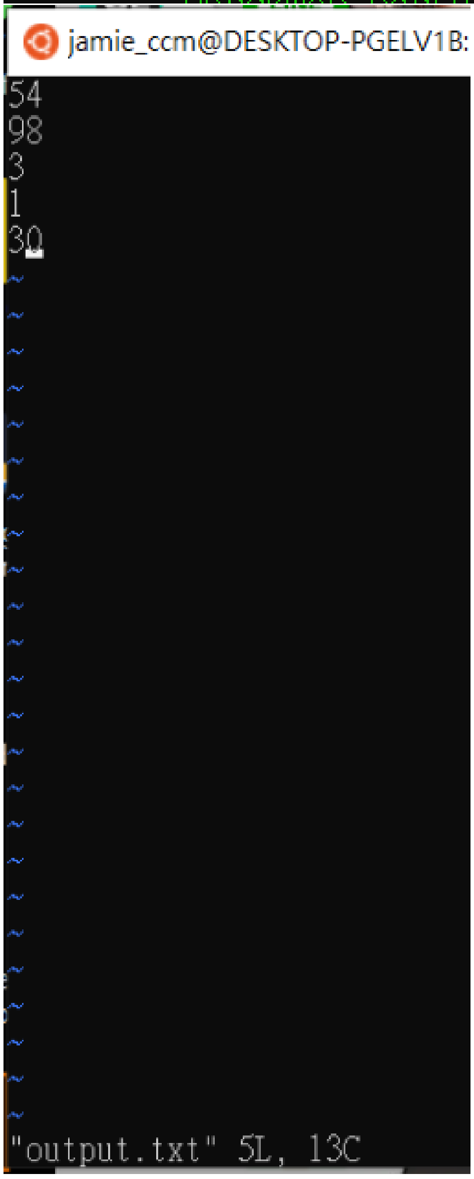
(1) result screenshot



```
jamie_ccm@DESKTOP-PGELV1B:~$ ./a.out<input.txt>output.txt
jamie_ccm@DESKTOP-PGELV1B:~$ vim output.txt
```

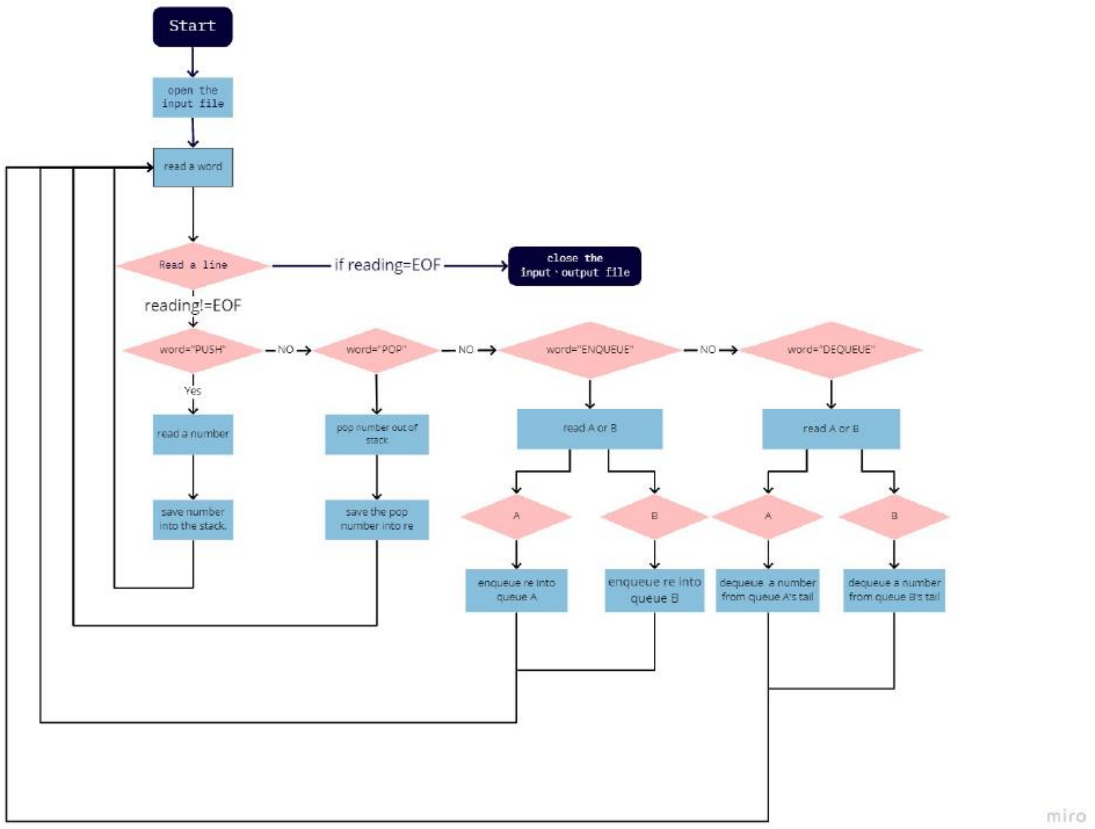


```
jamie_ccm@DESKTOP-PGELV1B:
54
98
3
1
30
~
~
~
"output.txt" 5L, 13C
```

(2) program architecture

1. Open and 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. Close the input file.
流程圖：



(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;
FILE *fpi;
```

Parameter:
re- Used to record the number popped out of the stack, so that it could be enqueue later.
fpi- the pointer pointed to input.txt.

```
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)
        fprintf(stderr,"Didn;t have enough space to allocate.")
    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)
        fprintf(stderr,"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(int *record,stackPtr *Sptr)
{
        if(IsEmptyS(*Sptr))
              fprintf(stderr,"Stack is empty.");
        else
      {
              stackPtr tempPtr=*Sptr;
              *record=(*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:
record- the parameter used to record the pop number (it is **re** in main **function**.) , we use the method of pass by address to let it maintain to 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(int record,queuePtr *Qtail,queuePtr *Qhead)
{
        queuePtr newPtr=getNewQueue(record);
        if(newPtr==NULL)
              fprintf(stderr,"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:
record- **record** is the number 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))
    {
        fprintf(stderr,"queue is empty.");
    }
    else
    {
        queuePtr tempPtr=*Qhead;
        *Qhead=( *Qhead )->nextptr;
        fprintf(stderr,"%d\n",tempPtr->plateIndex);
        free(tempPtr);
    }
}
```
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.

```
    fpi = freopen("input.txt", "r",stdin);
    stackPtr stacktop = NULL;
    queuePtr Ahead = NULL;
    queuePtr Atail = NULL;
    queuePtr Bhead = NULL;
    queuePtr Btail = NULL;
    char str[20]={0};
    int num;
    char aORb;
```
Parameter:

fpi- all standard input will come from input file.
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:
                    fprintf(stderr,"INCORECCT QUEUE.");
                    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.

```
        else fprintf(stderr,"Undefined command.");
```

if the reading is not four of this, it would print wrong message in the output file and it would 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

Rather than use an array like the textbook, I chose linked list to do it. At first, I forget to set up queue's head, this made the program running wrong. After I fixed the problem, I testing it.

However, the output file is all 0 instead of the correct answer. I used another compressor to test the same code and asked my friend to test it. I discover that, when I put variable **re** inside main function, the program can only run on clion and my friend’s computer, but not the gcc in unix system in my computer. If I move the variable **re** outside the main function, it could run at all of three compilers. I am not sure this is my computer’s problem or not. To ensure the program is executable at any computer, I put variable **re** outside the main function, but I think it can be both on outside or inside the main function.