Description

A queue is an abstract data type that serves as a collection of elements, where nodes are removed only from the *head* of the queue and are inserted only at the *tail* of the queue. Two principal operations can be used to manipulate a queue: enqueue, which inserts an element at the tail, and dequeue, which removes the element at the head of the collection.

Let’s see how the queue data structure can be realized in C++.We have an approach to implement queue: linked list. Thus, we define two classes as follows:

```
class Queue{
    friend std::ostream &operator<<(std::ostream &, Queue &);
    public:
        virtual ~Queue() {};
        virtual void enqueue(const int &) = 0;
        virtual int dequeue() = 0;
        virtual void print(std::ostream &output)=0;
};
class List_queue : public Queue{
    public:
        List_queue();
        virtual ~List_queue();
        void enqueue(const int &);
        int dequeue();
        void print(std::ostream &output);
    private:
        ListNode *head;
        ListNode *tail;
};
```

where

1. Class Queue serves as the abstract base class for realizing polymorphism
2. List_queue implements the queue data structure

Besides, we also overload the stream insertion operator (<<) to print the content of a queue object polymorphically.

REQUIREMENTS:

1. Implement the enqueue(), dequeue() and print() member functions of the List_queue class.
2. Implement the overloaded stream insertion operator (<<), which will call the correct member function print() polymorphically.

Note:

1.This problem involves three files.

- function.h: Class definitions.
- function.cpp: Member-function definitions.
- main.cpp: A driver program to test your class implementation.

You will be provided with main.cpp and function.h, and asked to implement function.cpp.

<code>function.h</code>

```
#ifndef FUNCTION_H

#define FUNCTION_H

#include <iostream>

class ListNode

{

    friend class List_queue; //make List_queue a friend

public:

    ListNode( const int &info ) //constructor

    : data( info ), nextPtr( NULL ), prevPtr( NULL )

    {
```

```
    } //end ListNode constructor
private:
    int data; //data
    ListNode *nextPtr; // next node in list
    ListNode *prevPtr;
}; //end class ListNode
class Queue{
    friend std::ostream &operator<<(std::ostream &, Queue &);
    public:
        virtual ~Queue() {};
        virtual void enqueue(const int &) = 0;
        virtual int dequeue() = 0;
        virtual void print(std::ostream &output)=0;
};
class List_queue : public Queue{
    public:
        List_queue();
        virtual ~List_queue();
        void enqueue(const int &);
        int dequeue();
        void print(std::ostream &output);
    private:
        ListNode *head;
        ListNode *tail;
};
#endif // FUNCTION_H
```

<code>main.cpp</code>

```
#include <iostream>

#include <string.h>

#include "function.h"

using namespace std;

int main(){

    List_queue L_queue;

    char command[10];

    int n;

    while(cin>>command){

        if(strcmp(command,"dequeue")==0){

            L_queue.dequeue();

        }else if(strcmp(command,"enqueue")==0){

            cin >> n;

            L_queue.enqueue(n);

        }else if(strcmp(command, "print") == 0){

            cout << L_queue << endl;

        }

    }

    return 0;

}
```

2.For OJ submission:

Step 1. Submit only your function.cpp into the submission block.

Step 2. Check the results and debug your program if necessary.

Input

There are three kinds of commands:

- *“enqueue integerA”* represents inserting an element with int value A at the tail of the queue.

- *“dequeue*” represents removing the element at the head of the queue.
- *“print”* represents showing the current content of the queue.

Each command is followed by a new line character.

Input terminated by EOF.

Output

The output should consist of the current state of the queue.

When the queue is empty, you don’t need to print anything except a new line character.

Sample Input

```
enqueue 2
enqueue 120
enqueue 15
enqueue 19
dequeue
enqueue 36
print
dequeue
print
```

EOF

Sample Output

```
120 15 19 36
15 19 36
```

EOF