```
#ifndef HEADER_H_
#define HEADER_H_

#include <iostream> // cin, cout.
#include <string>   // string datatype variables.
#include <iomanip>  // fixed, setw, setprecision.
#include <limits>
#include <ios>
using namespace std;

enum Menu {
    EXIT = 0,
    ENQUEUE,
    DEQUEUE,
    ISEMPTY,
    FRONT,
    SIZE,
    CLEAR
};

struct PersonNode {
    string name;
    char gender;
    int age;
    PersonNode* next;
};

const int INPUT_COL = 14; // CALC - setw size for display column.

/*****************************************************************************
* Enqueue
*   This function will receive a queue and enqueue a person to it.
*  ==> returns nothing.
*                                                                            ↵
  ***************************************************************************** ↵
  /
void Enqueue(PersonNode* &head,  // IN & CALC - Stack front.
             PersonNode* &tail); // IN & CALC - Stack tail.

/*****************************************************************************
* Dequeue
*   This function will receive a queue and remove the person in front.
*  ==> returns stack after removing person on top.
*                                                                            ↵
  ***************************************************************************** ↵
  /
PersonNode* Dequeue(PersonNode* head); // IN & CALC - Stack

void IsEmpty(PersonNode* head); // IN & CALC - Queue.
```

```
46
47  /*************************************************************************
48  * Front
49  *   This function will receive a queue and peek at the name in front of the
50  *   queue and display it.
51  *  ==> returns nothing.
52  *                                                                        ↵
      ************************************************************************ ↵
      /
53  void Front(PersonNode* head); // IN & CALC - Queue
54
55  /*************************************************************************
56  * IsEmpty
57  *   This function will receive a stack and check if it's empty or not.
58  *  ==> returns nothing.
59  *                                                                        ↵
      ************************************************************************ ↵
      /
60  void IsEmpty(PersonNode* head); // IN & CALC - Queue.
61
62  /*************************************************************************
63  * Size
64  *   This function will receive a stack and check it's size and display it.
65  *  ==> returns nothing.
66  *                                                                        ↵
      ************************************************************************ ↵
      /
67  void Size(PersonNode* head); // IN & CALC - Queue.
68
69  /*************************************************************************
70  * ClearQ(ueue)
71  *   This function will receive a queue and clear it.
72  *  ==> returns nothing.
73  *                                                                        ↵
      ************************************************************************ ↵
      /
74  void ClearQ(PersonNode*& head, // IN & CALC - Queue head.
75              PersonNode*& tail); // IN & CALC - Queue tail.
76
77  /*************************************************************************
78   * PrintHeaderFile
79   *   This function will output the header information
80   *
81                                                                          ↵
    *************************************************************************/
82  void PrintHeaderFile(ostream& output,      // IN  - output datatype.
83      string asName,        // IN  - assignment name
84      int asNum,             // IN  - assignment number
85      string studentName,   // IN  - student's name
```

```
86      string classInfo,     // IN  - class that is being taken
87      char asType,          // IN  - assignment type
88      long long studentID); // IN  - student ID
89
90  #endif
91
```

```cpp
#include "Header.h"

/*****************************************************************************
 * PrintHeaderFile
 *   This function will output the header information
 *
 * PRE-CONDITIONS
 *   The following parameters need to have a defined value prior to calling
 *   the function
 *         asName: The name of the assignment given in the course
 *         asNum: The number of the assignment given in the course
 *         studentName: The name of the student writing the code
 *         classInfo: The course name, date, and time of the class
 *         asType: Will either output as a lab or an assignment
 *         studentID: The Identification Number of the student
 *
 *****************************************************************************/

void PrintHeaderFile(ostream& output,       // IN - output datatype.
    string asName,      // IN - assignment name
    int asNum,          // IN - assignment number
    string studentName,  // IN - student's name
    string classInfo,    // IN - class that is being taken
    char asType,         // IN - assignment type
    long long studentID) // IN - student ID
{
    output << left;
    output << "*************************************************************   \n";
    output << "*   PROGRAMMED BY : " << studentName << endl;
    output << "*   " << setw(14) << "STUDENT ID " << ": " << studentID << endl;
    output << "*   " << setw(14) << "CLASS " << ": " << classInfo << endl;
    output << "*   ";

    // PROCESSING - This will adjust setws and format appropriately based
    //              on if this is a lab 'L' or assignment

    if (toupper(asType) == 'L')
    {
        output << "LAB #" << setw(9);
    }
    else
    {
        output << "ASSIGNMENT #" << setw(2);
    }
    output << asNum << ": " << asName << endl;
    output << "*************************************************************";
    output << right << endl;
```

```
47
48      return;
49  }
```

```
1  /*****************************************************************************
2  * AUTHOR     : Andrew Gharios
3  * STUDENT ID : 1449366
4  * LAB #9A    : Implementing a Queue.
5  * CLASS      : CS1B
6  * SECTION    : M-TH: 5-7:20p
7  * DUE DATE   : 7/13/21
8  *****************************************************************************/
9  #include "Header.h"
10
11 /*****************************************************************************
12 * Implementing a Stack
13 *----------------------------------------------------------------------------
14 * This program will provide a menu for the user to be able to manipulate a
15 * stack. The user has the option to Push, Pop, Peek, check the Size, and check
16 * if the stack is empty.
17 *----------------------------------------------------------------------------
18 * INPUT:
19 * input : user menu selection.
20 *****************************************************************************/
21 int main()
22 {
23     /*************************************************************************
24     * CONSTANTS
25     *
26     * OUTPUT - USED FOR CLASS HEADING
27     *
28     * PROGRAMMER : Programmer's Name
29     * CLASS      : Student's Course
30     * SECTION    : Class Days and Times
31     * LAB_NUM    : Lab Number (specific to this lab)
32     * LAB_NAME   : Title of the Lab
33     *************************************************************************/
34
```

```
35      const string AS_NAME = "Implementing a Queue";
36      const int AS_NUM = 9;
37      const string STUDENT_NAME = "Andrew Gharios";
38      const string CLASS_INFO = "M-Th 5-7:20p";
39      const char AS_TYPE = 'L';
40      const long long STUDENT_ID = 1449366;
41
42      PersonNode* head;  // IN & CALC - Stack front.
43      PersonNode* tail;  // IN & CALC - Stack tail.
44      int  input;        // IN & CALC - menu input.
45      Menu menu;         // CALC      - Menu option.
46      bool invalid;      // CALC      - Validation for input.
47
48      head = NULL;
49      tail = NULL;
50
51      PrintHeaderFile(cout, AS_NAME, AS_NUM, STUDENT_NAME, CLASS_INFO, AS_TYPE,
          STUDENT_ID);
52
53      cout << "STACK MENU:\n";
54      cout << "1 - ENQUEUE (Add a person)\n";
55      cout << "2 - DEQUEUE (Remove a person)\n";
56      cout << "3 - ISEMPTY (Is the queue empty?)\n";
57      cout << "4 - FRONT (Who is in front?)\n";
58      cout << "5 - SIZE (How many people are there?)\n";
59      cout << "6 - Clear the Queue\n";
60      cout << "0 - Exit\n";
61
62      do
63      {
64          do
65          {
66              invalid = false;
67              cout << "\nEnter a command? ";
68              if (!(cin >> input))
69              {
70                  cout << "**** Please enter a NUMBER between 0 and 6 ****\n";
71                  cin.clear();
72                  cin.ignore(numeric_limits<streamsize>::max(), '\n');
73                  invalid = true;
74              }
75              else if (input < 0 || input > 6)
76              {
77
78                  cout << "**** The number " << input << " is an invalid entry
                         *****\n";
79                  cout << "**** Please input a number between 0 and 6 *****\n";
80                  invalid = true;
81              }
```

```
82
83
84          } while (invalid);
85
86          cin.ignore(numeric_limits<streamsize>::max(), '\n');
87
88          menu = Menu(input);
89
90          switch (menu)
91          {
92          case 0:
93              break;
94          case 1:
95              Enqueue(head, tail);
96              break;
97          case 2:
98              head = Dequeue(head);
99              break;
100         case 3:
101             IsEmpty(head);
102             break;
103         case 4:
104             Front(head);
105             break;
106         case 5:
107             Size(head);
108             break;
109         case 6:
110             ClearQ(head, tail);
111             break;
112         }
113
114
115     } while (menu != EXIT);
116
117     return 0;
118
119 }
```

```cpp
#include "Header.h"

/*************************************************************************
* Enqueue
*   This function will receive a queue and add a person at the end of it.
*  INPUTS:
*  head : Queue.
*
*  OUTPUTS:
*  head : Queue(with added person).
*
*************************************************************************
*/
void Enqueue(PersonNode* &head, // IN & CALC - Queue front.
                  PersonNode* &tail) // IN & CALC - Queue tail.
{
    PersonNode* perPtr; // CALC - Pointer for manipulatin of stack.

    perPtr = new PersonNode;
    perPtr->next = NULL;

    cout << left;
    cout << endl;
    cout << "Who would you like to add?" << endl;
    cout << setw(INPUT_COL) << "Enter Name:";
    getline(cin, (*perPtr).name);
    cout << setw(INPUT_COL) << "Enter Gender:";
    cin >> perPtr->gender;
    cin.ignore(10000, '\n');
    cout << setw(INPUT_COL) << "Enter Age:";
    cin >> perPtr->age;
    cin.ignore(10000, '\n');
    cout << right;

    if (head == NULL)
    {
        head = perPtr;
    }
    else
    {
        tail->next = perPtr;
    }
    tail = perPtr;
    perPtr = NULL;
    delete perPtr;
}
```

```cpp
#include "Header.h"

/*************************************************************************
* Dequeue
*   This function will receive a queue and remove the person in front.
*
*  INPUTS:
*  head : Queue head.
*
*  OUTPUTS:
*  head : Queue(with removed person in front)
*
*************************************************************************/
PersonNode* Dequeue(PersonNode* head) // IN & CALC - queue head.
{
    PersonNode* perPtr; // CALC - Pointer for manipulatin of stack.

    perPtr = head;

    if (perPtr != NULL)
    {
        cout << left;
        cout << setw(INPUT_COL) << "DEQUEUEING" << endl;
        cout << setw(INPUT_COL) << "Name: " << perPtr->name << endl;
        cout << setw(INPUT_COL) << "Gender: " << perPtr->gender << endl;
        cout << setw(INPUT_COL) << "Age: " << perPtr->age << endl;
        cout << endl;
        cout << right;

        head = perPtr->next;


        return head;
    }
    else
    {
        cout << "Can't DEQUEUE from an empty list!" << endl;
    }

    perPtr = NULL;

    return head;
}
```

```cpp
#include "Header.h"

/*****************************************************************************
* IsEmpty
*   This function will receive a queue and check if it's empty or not.
*
* INPUTS:
*   head : queue.
*
* No outputs.
*
*****************************************************************************/
void IsEmpty(PersonNode* head) // IN & CALC - Queue.
{
    if (head == NULL)
    {
        cout << "Yes, the QUEUE is empty.\n";
    }
    else
    {
        cout << "The QUEUE is NOT empty.\n";
    }
}
```

```cpp
#include "Header.h"

/*************************************************************************
* Size
*   This function will receive a queue and check it's size and display it.
*
*  INPUTS:
*   head  : queue front.
*
* No outputs.
*
*************************************************************************/
void Size(PersonNode* head) // IN & CALC - Stack.
{
    PersonNode* perPtr;
    int         count;

    perPtr = head;
    count = 0;

    while (perPtr != NULL)
    {
        count++;
        perPtr = perPtr->next;
    }

    if (count == 0)
    {
        cout << "Nobody is in the queue!\n";
    }
    else if (count == 1)
    {
        cout << "There is one person in the queue.\n";
    }
    else
    {
        cout << "There are " << count << " people in the queue.\n";
    }
}
```

```cpp
#include "Header.h"

/*****************************************************************************
* ClearQ(ueue)
*   This function will receive a queue and clear it.
*
*  INPUTS:
*  head : Queue head.
*  tail : Queue tail.
*
*  No outputs.
*                                                                            ↵
*****************************************************************************↵
 /
void ClearQ(PersonNode*& head, // IN & CALC - queue head.
            PersonNode* &tail) // IN & CALC - queue tail.
{
    PersonNode* perPtr;
    perPtr = head;

    if (head == NULL && tail == NULL)
    {
        cout << "The QUEUE is already clear!" << endl;
    }
    else
    {
        cout << "CLEARING..." << endl;
        while (perPtr != NULL)
        {
            cout << perPtr->name << endl;

            head = perPtr->next;
            perPtr = perPtr->next;
        }
        tail = NULL;
    }
    perPtr = NULL;
}
```

```
1  ************************************************************
2  *   PROGRAMMED BY : Andrew Gharios
3  *   STUDENT ID    : 1449366
4  *   CLASS         : M-Th 5-7:20p
5  *   LAB #9        : Implementing a Queue
6  ************************************************************
7  STACK MENU:
8  1 - ENQUEUE (Add a person)
9  2 - DEQUEUE (Remove a person)
10 3 - ISEMPTY (Is the queue empty?)
11 4 - FRONT (Who is in front?)
12 5 - SIZE (How many people are there?)
13 6 - Clear the Queue
14 0 - Exit
15
16 Enter a command? 1
17
18 Who would you like to add?
19 Enter Name:   George Boole
20 Enter Gender: M
21 Enter Age:    32
22
23 Enter a command? 1
24
25 Who would you like to add?
26 Enter Name:   Ada Lovelace
27 Enter Gender: F
28 Enter Age:    21
29
30 Enter a command? 1
31
32 Who would you like to add?
33 Enter Name:   Grace Hopper
34 Enter Gender: F
35 Enter Age:    44
36
37 Enter a command? 4
38
39 The first person in the queue is:
40 Name:         George Boole
41 Gender:       M
42 Age:          32
43
44 Enter a command? 5
45 There are 3 people in the queue.
46
47 Enter a command? 2
48 DEQUEUEING
49 Name:         George Boole
```

```
Gender:       M
Age:          32


Enter a command? 5
There are 2 people in the queue.

Enter a command? 3
The QUEUE is NOT empty.

Enter a command? 4

The first person in the queue is:
Name:         Ada Lovelace
Gender:       F
Age:          21

Enter a command? 2
DEQUEUEING
Name:         Ada Lovelace
Gender:       F
Age:          21


Enter a command? 5
There is one person in the queue.

Enter a command? 4

The first person in the queue is:
Name:         Grace Hopper
Gender:       F
Age:          44

Enter a command? 3
The QUEUE is NOT empty.

Enter a command? 2
DEQUEUEING
Name:         Grace Hopper
Gender:       F
Age:          44


Enter a command? 5
Nobody is in the queue!

Enter a command? 4
Nobody in FRONT, the queue is empty!
```

```
 99
100  Enter a command? 3
101  Yes, the QUEUE is empty.
102
103  Enter a command? 2
104  Can't DEQUEUE from an empty list!
105
106  Enter a command? 1
107
108  Who would you like to add?
109  Enter Name:   Alan Turing
110  Enter Gender: M
111  Enter Age:    25
112
113  Enter a command? 1
114
115  Who would you like to add?
116  Enter Name:   Blaise Pascal
117  Enter Gender: M
118  Enter Age:    19
119
120  Enter a command? 1
121
122  Who would you like to add?
123  Enter Name:   Dog Bert
124  Enter Gender: M
125  Enter Age:    3
126
127  Enter a command? 6
128  CLEARING...
129  Alan Turing
130  Blaise Pascal
131  Dog Bert
132
133  Enter a command? 6
134  The QUEUE is already clear!
135
136  Enter a command? 8
137  **** The number 8 is an invalid entry     *****
138  **** Please input a number between 0 and 6 *****
139
140  Enter a command? a
141  **** Please enter a NUMBER between 0 and 6 ****
142
143  Enter a command? 0
```