```
//Eduardo Martinez
//CS211 Lab 6 Assignment 4
//Queue class - header file
// ========================================================
#ifndef QUEUE_H
#define QUEUE_H
#include <string>
using namespace std;



//create an enumuration type
enum op {ADD, SUB, MULT, DIVI};

//create a struct that will have an operand, a operator, another operand
struct expr
{
  int oprd1;
  op oprt;
  int oprd2;
};

typedef expr el_t; // el_t is an alias for char
const int QUEUE_SIZE = 10; // this is the max number of elements the queue can
have

class Queue
{
 private:
  // Data members are:
  el_t el[QUEUE_SIZE]; // a character array called el
  int count; // how many elements do we have right now?
  int front; // where the front element of the queue is.
  int rear; // where the rear element of the queue is.
  // a private utility function for fatal error cases
  // This displays an error messages passed to it and does exit(1);
  void queueError(string msg);
 public:
  // constructor
  Queue();
  // PURPOSE: if empty returns true, if not empty returns false
  bool isEmpty();
  // PURPOSE: if full returns true, if not full returns false
  bool isFull();
  // HOW TO CALL: pass an element to be added to the queue
  // PURPOSE: if full, calls an emergency exit routine
  // if not full, changes rear to the next slot and enters an element at rear
  void add(el_t);
  // PURPOSE: if empty, calls an emergency exit routine
  // if not empty, remove(return) the front element and change front to the next slot
  el_t remove();
  // PURPOSE: if empty, calls an emergency exit routine
  // if not empty, return the front element (but does not remove it)
  el_t getFront();
  //PURPOSE: if empty, calls an emergency exit routine
  //if queue has just 1 element, does nothing
  //if queue has more than 1 element, moves the front one to the rear
  void goToBack();
  // PURPOSE: returns the current size
  int getSize();
  //PURPOSE: display everything in the queue from front to rear enclosed in [].
E.g. [a][b][c]
  //Do not call the emergency exit routine when the queue is empty.
  void displayAll();
};

#endif
```

```
//Eduardo Martinez
//CS211 Lab 6 Assignment 4
//Queue class - implementation file
// =========================================================
#include "queue.h"
#include <iostream>
using namespace std;

// PURPOSE: constructor which initializes top
Queue::Queue()
{
  count = 0;
  front = 0;
  rear = -1;
}
// PURPOSE: (private) to handle unexpected errors encountered by other methods
// PARAMS: a string message to be displayed
// ALGORITHM: simply cout the message and exit from the program
void Queue::queueError(string msg)
{
  cout << msg << endl;
  exit(1);
}
// PURPOSE: Checks if queue is empty
// ALGORITHM: if count equals 0 returns true, else returns false
bool Queue::isEmpty()
{
  if(count == 0)
    return true;
  else
    return false;

}
// PURPOSE: checks if queue is Full
// ALGORITHM: if count is greater than queue size, then returns tru
// else returns false
bool Queue::isFull()
{
  if(count > QUEUE_SIZE)
    return true;
  else
    return false;
}
// PURPOSE: to add a passed element to the queue
// PARAMS: new element n of type el_t
// ALGORITHM: if not full, increment count and change rear
// else queueError is called
void Queue::add(el_t e)
{
  if(isFull())
```

```
    queueError("queue is Full");
  else
    {
      rear = (rear + 1) % QUEUE_SIZE;
      el[rear]= e;
      count++;
    }
}
// PURPOSE: to remove element from queue
// ALGORITHM: if not empty, decrement count, change front, and return
removed
// else stackError is called
el_t Queue::remove()
{
  if(isEmpty())
    queueError("queue is empty");
  else
    {
    count--;
    el_t e= el[front];
    front = (front + 1) % QUEUE_SIZE;
    return e;
    }
}
// PURPOSE:get front of queue without removing
// ALGORITHM: if not empty, returns front of queue
el_t Queue::getFront()
{
  if(isEmpty())
    queueError("queue is empty");
  else
    {
      return el[front];
    }
}
// PURPOSE: takes front and adds to the rear of queue
// AlGORITHM: if not empty, adds removed element to queue
// if size is 1 then does nothing
void Queue::goToBack()
{
  if(isEmpty())
    queueError("Queue is empty");
  else if(count == 1)
    {}
  else
    {
      el_t e = remove();
      add(e);
    }
}
```

```
// PURPOSE: returns the size of the queue
int Queue::getSize()
{
  return count;
}
// PURPOSE: Displays all elements in queue
/*
void Queue::displayAll()
{
  while(!isEmpty())
    {
      for(int i = front; i <= rear; i++)
            {
              cout << "[" <<  el[i] << "]";
            }
    }
}
*/
```

```
//Eduardo Martinez
//CS211 Lab 6 assignment 4
//queueClient
//========================================
#include <iostream>
#include "queue.h"
#include "inputCheck.h"
using namespace std;

int main()
{
  Queue myLine; // myLine is a new queue object
  char userans,eltoadd;

  cout << "Enter your choice Y to add a new element or N when you are done";
  //my getResponse() in inputCheck.h returns either Y or N (returns uppercase)
  userans = getResponse("Invalid choice. Enter Y or N only: ");
  while (userans == 'Y' && !myLine.isFull() )
    {
      cout << "Give me an element to add: ";
      cin >> eltoadd;
      myLine.add(eltoadd);

      cout << "Enter your choice Y to add a new element or N when you are done.";
      userans = getResponse("Invalid choice. Enter Y or N only: ");
    }// end of while

  cout << "The line has " << myLine.getSize() << " elements." << endl;
  cout << "Now removing and displaying all elements…" << endl;

while ( !myLine.isEmpty())
    cout << myLine.remove() << endl;
}//end of mai
```

```
/******************************************************************
***********
Eduardo Martinez
CS211
Lab 8, Assignment 4
Queue application
Template written by Kazumi Slott
3/7/2016

To compile:
g++ game.C queue.C -pthread

This program will ask the user to answer math questions (add, sub, mult, div).
The queue will have 3 questions before the game starts. After the game starts, a
new question
will be added every 1 second if the level is 5, 2 seconds if the level is 4, .. 5
seconds if the level is 1.
The user will be asked to choose a level from 1 to 5 before the game starts.
A question for the user to answer will be removed from the front of the queue.
The user will be asked to answer
the same question until he gives the correct answer. After he gives a correct
answer, the next question will be removed from the front of
the queue.

When the queue grows to have 10 questions, the game ends and the user loses (he
was too slow doing the math).
When the queue becomes empty, the game ends and the user wins (he was quick
doiing the math).
When the user answers 100 questions correctly, the game ends and the user wins
(the queue never became empty or grew to have 10 questions).

This program uses one thread to add new questions to the queue and another to let
the user
enter math questions.
*******************************************************************
**********/

#include <time.h>
#include <iostream>
#include <pthread.h>
#include "inputCheck.h"
#include "queue.h" //your queue class
using namespace std;

//prototypes
void *answerQuestion(void* data);
void *addQuestion(void* data);
int correctAnswer(int op1, char optr, int op2);
char getOperator(op o);
expr makeQuestion();
```

```
//global - easier to share them between threads
Queue q; //create a queu object. the queue will store math questions
bool win; //set to true if win, false if lose
int numCorrect = 0; //the number of correct questions the user answered
pthread_mutex_t lock; //used to lock a part of code where a shared resource (q)
                    //is updated by a thread
int level;//level of difficulty (1 for easy/slot, 5 for hard/fast)

int main()
{
  //get a different sequence of random nnumbers in each run
  srand(time(0));

  cout << "Which level do you want to try? 1 (easy) to 5 (hard): ";
  //level 1 will add a new question every 5 seconds. If level 2, every 4 seconds. If
level 5, every 1 second.
  level = getNumberInRange(1, 5, "Invalid level. Enter 1 to 5: "); //from
inputCheck.h

  //adds 3 questions into the rear of the queue
  q.add(makeQuestion());
  q.add(makeQuestion());
  q.add(makeQuestion());


  //initialize the mutex
  if (pthread_mutex_init(&lock, NULL) != 0)
    {
      cout << "Creating a mutext failed." << endl;
      return 1; //ending the program. 1 is an error code passed to the operating
system
    }

  //delcare 2 threads. first thread to add new questions and second for the user to
answer questions.
  pthread_t tAddQues, tAns;

  //thread to add new qustions to the rear of the queue
  pthread_create(&tAddQues, NULL, &addQuestion, NULL);

  //thread for the user to answer questions removed from the front of the queue
  pthread_create(&tAns, NULL, &answerQuestion, NULL);


  //wait for the thread to come back from addQuestion()
  pthread_join(tAddQues, NULL);

  //wait for the thread to come back from answerQuestion()
  pthread_join(tAns, NULL);
  //win is set to true in answeQuestion() - if the user answers quickly and the
queue gets empty or he answers 100 questions correctly, the user wins the game
  if(win == true)
    cout << "you win" << endl;
  else //if the user doesn't answer questions quick enough and the queue grows to
have 10 questions, he loses.
    cout << "you lose" << endl;

    cout << "You answered " << numCorrect << " questions correctly." << endl;

  return 0;
}
void *addQuestion(void* data)
{
    expr newQ;//a new question to be added

    clock_t endWait;
    //a new question will be added to the queue every 1 second if the level is 5,
    //2 seconds if the level is 4, .. 5 seconds if the level is 1.
    int waitTime = CLOCKS_PER_SEC * (level% 5 +1);
    endWait = clock() + waitTime;

    //as soon as the queue grows to have 10 questions, gets empty or the user
answers 100 questions correctly, the game ends
    while(q.getSize() < 10 && numCorrect < 100  && !q.isEmpty())
              {
                //it is time to add a new question to the queue
                if(clock() == endWait)
                  {
                        //create a new question
                        newQ = makeQuestion();
                        //lock the code so this thread has exclusive access to the queue
while updating
                        pthread_mutex_lock(&lock);
                        //add the new question to the rear of the queue
                        q.add(newQ);
                        pthread_mutex_unlock(&lock); //unlock the exclusive access so
the other thread
                                                        //can access the queue now
                        //reset the end wait time
                        endWait= clock() + waitTime;
                  }
              }
}

void *answerQuestion(void* data)
{
  int answer,correct;
  int op1, op2;
  char opr;

  //as soon as the queue grows to have 10 questions, gets empty, or the user
answers 100 questions correctly, the game ends
  while(q.getSize() < 10 && numCorrect < 100 && !q.isEmpty())
    {
      //get the question from the front of the queue
      //lock the code so this thread has exclusive access to the queue while updating
      pthread_mutex_lock(&lock);

      expr ques = q.remove();

      pthread_mutex_unlock(&lock); //unlock the exclusive access so the other
thread
                                   //can access the queue now

      op1 = ques.oprd1;
      opr = getOperator(ques.oprt);
      op2 = ques.oprd2;

      //get the answer to the question
      correct  = correctAnswer(op1,opr,op2);

      //ask the usert to enter the user's answer
      cout << op1 << " " << opr << " " << op2 << " = ";
      answer = getNumberInRange(0, 400, "Invalid answer. Enter your answer
again: ");//from inputCheck.h

      //as long as the user's answer is wrong, she/he will have to retry answering the
same question
      while(answer!= correct && q.getSize() < 10 && !q.isEmpty())
            {
             cout << "WRONG. try again. " <<  op1 << " " << opr << " " << op2 << "
= ";
             answer = getNumberInRange(0, 400, "Invalid answer. Enter your answer
again: ");//from inputCheck.h
            }
      //the user's answer was correct. the number of correct increases
      if(answer == correct)
           numCorrect++;
    }

  //if the queue grows to have 10 questions, the user loses the game
  //if the queue gets empty or the suer answers 100 questions correctly, the user
wins the game
  if(q.getSize() < 10 && numCorrect => 100 || q.isEmpty())
    win = true;
   else
     win = false;
}
```

```
//Converts an enum value to char
char getOperator(op o)
{
  switch(o)
    {
    case ADD: return '+';
    case SUB: return '-';
    case MULT: return '*';
    case DIVI: return '/';
    }
}

//do the math
int correctAnswer(int op1, char optr, int op2)
{
  switch(optr)
    {
    case '+': return op1 + op2;
    case '-': return op1 - op2;
    case '*': return op1 * op2;
    case '/': return op1 / op2;
    }
}

//creates a question and returns a struct
expr makeQuestion()
{
  int temp;
  expr e;
  e.oprt = (op)(rand()%3); //0 for add, 1 for sub, 2 for mult, 3 for divi

  if(e.oprt == MULT) //if the operator is multiplication, make operands between 1
and 20 for the first operand and between 1 and 10 for the second operand.
                    // (large operands would make multiplication hard.)
    {
      e.oprd1 = rand()% 20 + 1; //create a random number between 1 and 20
      e.oprd2 = rand()% 10 + 1;//create a random number between 1 and 10
    }
  else //the operator is add, sub or divi. Make operands between 1 and 100
    {
      e.oprd1 = rand()% 100 + 1;//create a randowm number between 1 and 100
      e.oprd2 = rand()% 100 + 1; //create a randowm number between 1 and 100

      //if the operator is sub or division, the first operand should be greater than or
equal to the second operator (otherwise the calulation
      //would the too difficult for SUB and too easy for DIVI.
      if(e.oprt == SUB || e.oprt == DIVI)
            {
```

```
          if(e.oprd1 < e.oprd2) //if the second operand is larger, swap operand1 and
operand2
           {
            temp = e.oprd1;
            e.oprd1 = e.oprd2;
            e.oprd2 = temp;
           }
        }
   }

 return e;
}
```