CONDITIONS MODULE:

```
#include "Conditions.hpp"
#include <string>
#include <iterator>
#include <iostream>
#include <fstream>
#include <sstream>
#include <set>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

/*

  Constructor set's ChashTable size to ChashTableSize and ShashTable size to

*/

Conditions::Conditions(int ChashTableSize, int ShashTableSize){
  this->ChashTableSize=ChashTableSize;
  this->ShashTableSize=ShashTableSize;
  ChashTable= new condition* [ChashTableSize]; //condition type
  ShashTable= new symptom* [ShashTableSize]; //symtom type
  for (int i=0; i<ChashTableSize; i++) // set ChashTable indicies to nullptr;
  {
    ChashTable[i]=nullptr;
  }
  for (int j=0; j<ShashTableSize; j++) // set ShashTable indicies to nullptr;
  {
    ShashTable[j]=nullptr;
  }
}

/*
  Hastable delete so basically go down the array and than go down the linked
list for a given index in an array
*/
Conditions::~Conditions(){
  for (int i=0; i<ChashTableSize; i++) // deleting from the ChashTable
  {
    condition* temp=ChashTable[i];
    while (temp!=0)
    {
      condition* del = temp;//create del and repoint
      temp=temp->next;
      delete del;
    }
  }
  delete []ChashTable;

  for (int j=0; j<ShashTableSize; j++) // deleting from ShashTable
  {
    symptom* temp=ShashTable[j];
    while (temp!=0)
    {
      symptom* del = temp;//create del and repoint
      temp=temp->next;
      delete del;
    }
  }
  delete []ShashTable;
}

/*
  Pass in file name, opens file, first while loop reads in every line with getline and saves in "line",
  stringstream gets name and priority(use stoi to turn into integer), then use another while loop to
  read in the associated symptoms with stringstream, create symptoms, create pointers and put into a set.
  Then call add function and pass in all parameters
*/
void Conditions::readFile(string filename){
  ifstream file;
  file.open(filename);
  if (! file.is_open()){
    cout<<"File failed to open"<<endl;
    return;
  }
```

```
//cout<<"File opened"<<endl;
string line;
while(getline(file, line))//gets each line in file
{
  if (line=="") break;
  //cout<<"reading a line"<<endl;
  stringstream s1(line);
  string name;
  string prior;
  int priority;
  getline(s1, name, ',');//stringstream gets name of condition
  //cout<<"condition: "<<name<<endl;
  getline(s1,prior,',');//gets priority
  priority=stoi(prior);


  string sympt;
  set<symptom*>  symptoms;
  set<int> test;
  symptom* pointer=0;
  condition* conditionpointer;
  while(getline(s1, sympt,','))//gets each symptom
  {
    if (sympt=="") break;
    symptom* temp= new symptom;//create a symptom
    temp->name=sympt;
    //cout<<"reading a symptom"<<temp->name<<endl;
```

```
      //cout<<"Readfile: parse symptom "<<temp->name<<endl;
      //n++;
      //test.insert(n);
      //cout<<"Adding symptom number: "<<n<<endl;
      symptoms.insert(temp);//push pointer into set of symptoms
      //cout<<"first in set: "<<(*symptoms.begin())->name<<"; size of set is
"<<symptoms.size()<<endl;
      Sadd(temp);//add symptom to ShashTable
    }
      conditionpointer = Cadd(name, priority, symptoms);//add condition to
ChashTable
      //cout<<"iterator"<<endl;
      set<symptom*>::iterator i;
      //cout<<"before while loop"<<endl;
      //cout<<"size of symptoms is "<<symptoms.size()<<endl;
      for(i=symptoms.begin(); i!=symptoms.end(); ++i)//iterate through set of
symptoms
      {
        //cout<<"inserting condition pointer"<<endl;
        //cout<<"adding condition to: "<<(*i)->name<<endl;
        symptom** dpointer=accessSymptom((*i)->name);
        (*dpointer)->conditions.insert(conditionpointer);
        //(*i)->conditions.insert(conditionpointer);//at each symptom insert
condition pointer into set of condition pointers
      }
  }
  //cout<<"Done"<<endl;
```

```
}

/*
  we pass in the string name, it's priority, a set of symptom pointers; add the
condition to the hastable at the correct index
*/

condition* Conditions::Cadd(string name, int priority, set<symptom*>
symptoms ){
  //cout<<"Entered Cadd"<<endl;
  int index = ChashFunction(name);
  //cout<<"hashed"<<endl
  if (searchCondition(name) != 0) return 0;
  condition* temp = new condition;
  temp->name = name;
  temp->priority = priority;
  temp->symptoms = symptoms;
  if(ChashTable[index] == 0){
    ChashTable[index] = temp;
    //cout<<"added "<<ChashTable[index]->name<<endl;
    return ChashTable[index];
  }
  else{
    condition *trav = ChashTable[index];
    while(trav->next != 0){
      trav = trav->next;
    }
    trav->next = temp;
    return trav->next;
  }
}

/*
  we pass in a symtom and we just add to the ShashTable
*/

void Conditions::Sadd(symptom* temp1){
  int index = ShashFunction(temp1->name);
  if(searchSymptom(temp1->name)!=0) return;
  //cout<<"Sadd is adding (Sadd) "<<temp1->name<<endl;
  if(ShashTable[index] == 0){
    ShashTable[index] = temp1;
    //cout<<"added (Sadd)"<<ShashTable[index]->name<<endl;
  }
  else{
    symptom* trav = ShashTable[index];
    while(trav->next != 0){
      trav = trav->next;
    }
    trav->next = temp1;
    //cout<<"added (Sadd)"<<trav->next->name<<endl;
  }
}
```

```
/*
  The ChashFunction
*/

int Conditions::ChashFunction(string word){
  unsigned int hashValue = 5381;
  int length = word.length();
  for (int i=0;i<length;i++)
  {
  hashValue=((hashValue<<5)+hashValue) + word[i];
  }
  hashValue %= ChashTableSize;
  return hashValue;
}

/*
  The ShashFunction
*/

int Conditions::ShashFunction(string word){
  unsigned int hashValue = 5381;
  int length = word.length();
  for (int i=0;i<length;i++)
  {
  hashValue=((hashValue<<5)+hashValue) + word[i];
  }
  hashValue %= ShashTableSize;
  return hashValue;
}

/*
  used when the patient already knows their condition and we add them to the
queue
*/
condition* Conditions::searchCondition(string name){
  int index = ChashFunction(name);
  if( ChashTableSize == 0){
    return 0;
  }
  condition *trav = ChashTable[index];
  while(trav != 0){
    if(trav->name == name) return trav;
      trav = trav->next;
  }

  return 0;
}



void Conditions::menu(){
  string a1;
  int a;
  cout<<"1. If this is an medical emergency please enter 1"<<endl;
  cout<<"2. If you would like to make a appointment please enter 2"<<endl;
  getline(cin, a1);
  a = stoi(a1);

  if(a == 1){
    cout<<"We are contacting emergency services. Help will arrive as soon as
possible."<<endl;
    return;
  }
  else{
    //cout<<"Before createPatient"<<endl;
    createPatient();
    menu1();
    cin.ignore(0);
    getline(cin, a1);
    a=stoi(a1);
    if (a==1){
      cout<<"Please enter your medical condition: "<<endl;
      string a2;
      getline(cin, a2);
      thepatient->condition = searchCondition(a2);
      //cout<<"after searchCondition"<<endl;
      if (thepatient->condition == 0){
        a=2;
      }
    }
```

```
else if (a == 2){
  string done;
  int temp;
  set<symptom*> p;
  cout<<"Select from the following symptoms (type by name): "<<endl;
  while( done != "done"){
    printSymptoms();
    cout<<"If you are done selecting your symptoms type: done"<<endl;
    cin.ignore(0);
    getline(cin, done);
    if (done != "done"){
      symptom* temp1 = searchSymptom(done);
      p.insert(temp1);
    }
  }
  //cout<<"Out of while loop"<<endl;
  //set<symptom*> s;
  //s=p;
  thepatient->symptoms = p;
  analyzeMatchedConditions(getBestMatchConditions());
  //cout<<"Back to the menu option 2"<<endl;
}
else if (a == 3){
  printOrder();
  //cout<<"here"<<endl;
}
```

```
    else{
      //cout<<"returns 281"<<endl;
      return;
    }
  }
  //cout<<"before adding patient"<<endl;
  addPatienttoqueue();
}

void Conditions::menu1(){
  cout<<"Choose from the following options below: "<<endl;
  cout<<"1. I already know medical condition "<<endl;
  cout<<"2. I need help to diagnose my medical condition "<<endl;
  cout<<"3. Print the Queue "<<endl;
  cout<<"4. Exit "<<endl;
}

void Conditions::menu2(){
  treatPatient();
}

/*Aks for imput, creates new patient, points patient to new patient and
deletes old patient*/
void Conditions::createPatient(){
  //cout<<"Entered createPatient function"<<endl;
  patient* temp=thepatient;
  string name;
  string painstr;
  int pain=0;
  cout<<"Please enter your legal name: ";
  getline(cin, name);
  while(pain<=0 || pain>20){
    cout<<"Please enter your current pain level ranging from 1-20 :";
    getline(cin, painstr);
    pain=stoi(painstr);
  }
  patient* newpatient=new patient;
  newpatient->name=name;
  newpatient->pain=pain;
  thepatient=newpatient;
  //delete temp;
  cout<<thepatient->name<<endl;
  cout<<thepatient->pain<<endl;
}

/*prints symptoms from hashtable*/
void Conditions::printSymptoms(){
  int cnt=1;
  for (int i=0; i<ShashTableSize; i++)
  {
    if (ShashTable[i]!=0)
    {
      symptom* temp=ShashTable[i];
      while(temp!=0)
      {
        set<condition*>::iterator g;
        cout<<cnt<<".) "<<temp->name<<endl;
      /*  for (g=temp->conditions.begin(); g!=temp->conditions.end(); ++g){
          cout<<"%%"<<(*g)->name<<endl;
        }
        */

        temp=temp->next;
        cnt++;
      }
    }
  }
}

void Conditions::printConditions(){
  int cnt=1;
  for (int i=0; i<ChashTableSize; i++)
  {
    if (ChashTable[i]!=0)
    {
      condition* temp=ChashTable[i];
      while(temp!=0)
      {
        set<symptom*>::iterator a;
        cout<<cnt<<".) "<<temp->name<<endl;
        /*for (a=temp->symptoms.begin(); a!=temp->symptoms.end(); ++a){
          cout<<"%%"<<(*a)->name<<endl;
        }
        */

        temp=temp->next;
        cnt++;
      }
    }
  }
}

symptom* Conditions::searchSymptom(string name){
  int index = ShashFunction(name);
  if( ShashTableSize == 0){
    return 0;
  }
  symptom *trav = ShashTable[index];
  while(trav != 0){
    if(trav->name == name) return trav;
    trav = trav->next;
  }
  //cout<<"Sorry, the symptom to you have typed does not exist in our
database or it has not been spelled properly"<<endl;
  return 0;
}
```

```
/*gets intersection of two symptom sets and returns intersection as a set*/
set<symptom*> Conditions::getIntersection(set<symptom*> set1,
set<symptom*> set2){
  set<symptom*> intersect;
  //cout<<"in getIntersection"<<endl;
  set<symptom*>::iterator i;
  set<symptom*>::iterator j;
  for (i=set1.begin(); i!=set1.end(); ++i){
    for (j=set2.begin(); j!=set2.end(); ++j){
      if ((*i)->name==(*j)->name){
        intersect.insert(*j);
        //cout<<"intersection: "<<(*j)->name<<endl;
      }
    }
  }
  //set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(),
inserter(intersect, intersect.begin()));
  //cout<<"in getIntersection"<<endl;
  //cout<<"intersect size is "<<intersect.size()<<endl;
  return intersect;
}
/*gets union of two condition sets and returns union as a set*/
set<condition*> Conditions::getUnion(set<condition*> set1, set<condition*>
set2){
  set<condition*> union_;
  set<condition*>::iterator i;
  set<condition*>::iterator j;
  //cout<<"set1 size "<<set1.size();
  //cout<<"set2 size "<<set2.size();
  for (i=set1.begin(); i!=set1.end(); ++i){
    union_.insert(*i);
  }
  for (j=set2.begin(); j!=set2.end(); ++j){
    union_.insert(*j);
  }
  //set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(),
inserter(union_, union_.begin()));
  //cout<<"in getUnion"<<endl;
  //cout<<"union_ size is "<<union_.size();
  return union_;
}
/*returns size of intersect divided by patient set (matching percentage)*/
float Conditions::getPercentage(set<symptom*> intersect){
  //cout<<"in getPercentage 1 "<<endl;
  //cout<<"intersect size "<<intersect.size()<<endl;
  //cout<<"patient size "<<thepatient->symptoms.size()<<endl;;
  //cout<<"intersect elements: "<<endl;
/*  set<symptom*>::iterator i;
  for(i=intersect.begin(); i!=intersect.end(); ++i){
    cout<<(*i)->name<<endl;


  }
  */
  float percent=(float)intersect.size()/(float)thepatient->symptoms.size();
  //cout<<"in getPercentage"<<endl;
  //cout<<percent<<endl;
  return percent;
}
/*puts conditions in priority queue based on matching percentage, returns
priority queue*/
priority_queue<condition*, std::vector<condition*>, Compare1>
Conditions::getBestMatchConditions(){
  //cout<<"in getBestMatchConditions"<<endl;
  set<condition*> allconditions;
  set<symptom*>::iterator i;
  for(i=thepatient->symptoms.begin(); i!=thepatient->symptoms.end(); ++i){
    //cout<<"conditions size is "<<(*i)->conditions.size()<<endl;;
    //cout<<(*i)->name<<endl;;
    allconditions=getUnion(allconditions, (*i)->conditions);
    //cout<<"allconditions size is "<<allconditions.size()<<endl;
    //cout<<"Out of getUnion"<<endl;
  }
  //cout<<"before matchedlist"<<endl;
  priority_queue<condition*, vector<condition*>, Compare1 > matchedlist;
  //cout<<"after matched list"<<endl;
  set<condition*>::iterator j;
  //cout<<"after iterator"<<endl;
  //cout<<"size of all conditions "<<allconditions.size()<<endl;
  for(j=allconditions.begin(); j!=allconditions.end(); ++j){
    //cout<<"In for loop"<<endl;
    set<symptom*> temp=getIntersection((*j)->symptoms, thepatient-
>symptoms);
    (*j)->percentage=getPercentage(temp);
  //  cout<<(*j)->name<<endl;
    //(*j)->percentage=0.8;
    //cout<<"After percentage"<<endl;
    //getPercentage(temp);
    matchedlist.push(*j);
    //cout<<"size of matchedlist "<<matchedlist.size()<<endl;
  }
  return matchedlist;
}
/*if there is one perfect match, saves that as patient's condition; if multiple
perfect matches or close matches,
patient gets to choose their condition or describe to doctor. If no close
matches, patient describes their condition.*/
void Conditions::analyzeMatchedConditions(priority_queue<condition*,
std::vector<condition*>, Compare1> Q){
  cout<<"Entered analyzeMatchedConditions"<<endl;
  vector<condition*> C;
  //cout<<"error1"<<endl;
  if(Q.top()->percentage==1.0){
    //cout<<"error2"<<endl;
    int cnt=0;
    while((Q.top()->percentage==1.0) && (cnt<Q.size()) ){
      //cout<<"error3"<<endl;
```

```
     condition* temp=Q.top();
     //cout<<"error4"<<endl;
     Q.pop();
     //cout<<"error5"<<endl;
     C.push_back(temp);//unfinished
     //cout<<"error6"<<endl;
     cnt++;
   }
   if(C.size()==1){
     //cout<<"error7"<<endl;
     cout<<"We have found one perfect match condition: "<<C.front()->name<<endl;
     //cout<<"error8"<<endl;
     thepatient->condition=C.front();
     //cout<<"error9"<<endl;
     return;
   }
   else{
     cout<<"Here are the perfect matches with their lists of symptoms. Enter the number of the one you identify with the most "<<endl;
     cout<<"If you don't identify with any of the matched conditions, type 0 to open a window to describe your symptoms and allow our doctors to judge your condition."<<endl;
     //print symptoms (choices)
     //cout<<"error10"<<endl;
     for (int i=0; i<C.size(); i++){
       cout<<i+1<<")["<<C[i]->name<<"] -- symptoms:";
```

```
      set<symptom*>::iterator j;
      for(j=C[i]->symptoms.begin(); j!=C[i]->symptoms.end(); ++j){
        cout<<"||"<<(*j)->name;
        //cout<<"error11"<<endl;
      }
      cout<<endl;
      //cout<<"error12"<<endl;
    }
    //patient's choice
    ////cout<<"error13"<<endl;
    int choice;
    string temp;
    getline(cin, temp);
    choice=stoi(temp);
    if (choice==0){
      //call description
      //cout<<"error14"<<endl;
      writeDescription();
    }
    else if(choice>0 && choice<=C.size()){
      thepatient->condition=C[choice-1];
      //cout<<"error15"<<endl;
    }
    return;
  }
}
```

```
 else if (Q.top()->percentage>=0.3){
  //cout<<"error16"<<endl;
  int cnt2=0;
  while(Q.top()->percentage>=0.3 && cnt2<Q.size() ){
   //cout<<"error17"<<endl;
   condition* temp=Q.top();
   //cout<<"error18"<<endl;
   Q.pop();
   //cout<<"error19"<<endl;
   C.push_back(temp);//unfinished
   //cout<<"error20"<<endl;
   cnt2++;
  }
  cout<<"These are close matches to your symptoms. Enter the number of the
one you identify with the most."<<endl;
  cout<<"If you don't identify with any of these conditions, enter 0 to open a
window to describe your symptoms to the doctor."<<endl;
  //cout<<"error21"<<endl;
  cout<<"size of C "<<C.size()<<endl;
  for (int i=0; i<C.size(); i++){
   //cout<<"error22"<<endl;
   cout<<i+1<<")["<<C[i]->name<<"] -- matching percentage:"<<C[i]-
>percentage<<" ; symptoms:";
   //cout<<"error23"<<endl;
   set<symptom*>::iterator j;
   //cout<<"error24"<<endl;
   for(j=C[i]->symptoms.begin(); j!=C[i]->symptoms.end(); ++j){
      //cout<<"error25"<<endl;
      cout<<"||"<<(*j)->name;
      //cout<<"error26"<<endl;
    }
    cout<<endl;
  }
  //patient's choice
  int choice;
  string temp;
  cin.ignore(0);
  getline(cin, temp);
  choice=stoi(temp);
  if (choice==0){
    //call description
    //cout<<"error27"<<endl;
    writeDescription();
  }
  else if(choice>0 && choice<=C.size()){
    thepatient->condition=C[choice-1];
    //cout<<"error29"<<endl;
  }
  return;

}
else{
  //cout<<"error30"<<endl;
```

```
    cout<<"Sorry we didn't not find a match for your symptoms in our
database.";
    //call description function
    //cout<<"error31"<<endl;
    writeDescription();
    return;
  }
}
/*patient describes symptoms, send to doctor*/
void Conditions::writeDescription(){
  cout<<"Please describe your symptoms as precisely as possible: "<<endl;
  string description;
  cin.ignore(0);
  getline(cin, description);
  ofstream file;
  file.open("patientdescription.txt");
  //cout<<"After file.open"<<endl;
  if(file.is_open()){
    //cout<<"In if statement"<<endl;
    file<<thepatient->name<<":"<<description<<"\n";
  }
  //cout<<"Out of if statement"<<endl;
  file.close();//FIGURE OUT HOW TO NOT OVERWRITE EACH TIME.
  cout<<"Your description has been sent to the doctor for further
analysis."<<endl;
}
/*resets percentage of all conditions to 0 for next patient*/
```

```
void Conditions::resetPercentage(){
  for(int i=0; i<ChashTableSize; i++)
  {
    if (ChashTable[i]!=0)
    {
      condition* temp=ChashTable[i];
      while(temp!=0)
      {
        temp->percentage=0;
        temp=temp->next;
      }
    }
  }
}
/*pops patient from queue and updates queue*/
void Conditions::treatPatient(){
  patient* temp;
  temp=queue1.top();
  queue1.pop();
  cout<<"Now treating patient "<<temp->name<<endl;
  updateQueue();
  cout<<"Next patient (predicted): "<<queue1.top()->name<<endl;
}
/*calculates total priority of patient, then pushes patient into queue*/
void Conditions::addPatienttoqueue(){
  if (thepatient->condition==0) return;
  thepatient->totalP=thepatient->condition->priority + thepatient->pain;

  queue1.push(thepatient);

  cout<<"Entered "<<queue1.top()->name<<" to queue."<<endl;
  //cout<<"queue size is "<<queue.size()<<endl;
}
/*adds 10 to the priority to all patients already in queue*/
void Conditions::updateQueue(){
  priority_queue<patient*, vector<patient*>, Compare2> newqueue = queue1;
  for(int i=0; i<queue1.size(); i++){
    patient* temp=queue1.top();
    queue1.pop();
    temp->totalP+=10;
    if (temp->totalP>=100){
      temp->totalP=100;
    }
    newqueue.push(temp);//pushes updated patient into new queue
  }
  queue1=newqueue;
}
/*pops each patient from queue, prints patient, puts back in new queue*/
void Conditions::printOrder(){
  priority_queue<patient*, vector<patient*>, Compare2> newqueue = queue1;
  while(!newqueue.empty()){
    cout<<newqueue.top()->name<<endl;
    newqueue.pop();
  }
}

void Conditions::system(){
  string temp;
  int c=0;
  while(c!=4){
    cout<<"Enter 1 for patient interface, 2 for doctor interface, 3 to check the queue, 4 to exit"<<endl;
    cout<<"----------------------------------------------------------------"<<endl;
    getline(cin, temp);
    c=stoi(temp);
    if (c==1){
      menu();
    }
    else if (c==2){
      if (queue1.size()==0){
        cout<<"There are no patients queued up at the moment."<<endl;
      }
      else {
      menu2();
    }
    }
    else if (c==3){
      printOrder();
    }
    else if (c==4){
      return;
    }
    cout<<"-----------------------------------------------------------------"<<endl<<endl;;
  }
}

symptom** Conditions::accessSymptom(string name){
  int index = ShashFunction(name);
  if( ShashTableSize == 0){
    return 0;
  }
  symptom **trav = &ShashTable[index];
  while(trav != 0){
    if((*trav)->name == name) return trav;
    trav = &((*trav)->next);
  }
  //cout<<"Sorry, the symptom to you have typed does not exist in our
database or it has not been spelled properly"<<endl;
  return 0;
}
```

MAIN FILE:

```
#include "Conditions.hpp"
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <set>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

int main(){
  Conditions C(10, 10);
  C.readFile("conditions.txt");
  C.system();
  priority_queue<patient*> queue;
  patient a;
  C.printConditions();
  C.printSymptoms();
  C.menu();
  cout<<"TEST SEARCH FUNCTION"<<endl;
  C.searchSymptom("nausea");
  C.searchCondition("lung cancer");
  C.printSymptoms();
  C.writeDescription();
  set<int> test;
  test.insert(1);
  test.insert(2);
  test.insert(3);
  set<int>::iterator i;
  for (i=test.begin(); i!=test.end(); ++i){
    cout<<*i<<endl;
}
  return 0;
}
```