#include <iostream>

#include <conio.h>

#include <fstream>

#include <sstream>

#include <string>

#include <queue>

using namespace std;

class stringdata

{

public:

    string name;

    int age;

    string dateofbirth;

    void print()

    {

        cout << name << " " << age << " " << dateofbirth;

    }

};

class treeNode

{

public:

    stringdata data;

    treeNode \*leftchild = NULL;

    treeNode \*rightchild = NULL;

};

class node

{

public:

    stringdata data;

    node \*next = NULL;

};

class linkedlist

{

private:

    node \*head, \*last;

    int siz;

public:

    linkedlist()

    {

        head = NULL;

        last = NULL;

        siz = 0;

    }

    linkedlist(stringdata arr[], int n)

    {

        head = NULL;

        last = NULL;

        siz = 0;

        for (int i = 0; i < n; i++)

        {

            queue\_adding\_data(arr[i]);

        }

    }

    void printlinkedlist()

    {

        node \*last = head;

        while (last != NULL)

        {

            last->data.print();

            cout << endl;

            last = last->next;

        }

        cout << endl;

    }

    stringdata delete\_head()

    {

        if (head != NULL)

        {

            siz = (siz == 0) ? siz : siz - 1;

            node \*curr = head;

            head = head->next;

            stringdata a = curr->data;

            delete curr;

            return a;

        }

    }

    int size()

    {

        return siz;

    }

    node \*divide1(node \*h, node \*e, node \*\*nh, node \*\*ne)

    {

        node \*pivot = e;

        node \*prev = NULL, \*cur = h, \*tail = pivot;

        while (cur != pivot)

        {

            if (cur->data.age < pivot->data.age)

            {

                if ((\*nh) == NULL)

                    (\*nh) = cur;

                prev = cur;

                cur = cur->next;

            }

            else

            {

                if (prev)

                    prev->next = cur->next;

                node \*tmp = cur->next;

                cur->next = NULL;

                tail->next = cur;

                tail = cur;

                cur = tmp;

            }

        }

        if ((\*nh) == NULL)

            (\*nh) = pivot;

        (\*ne) = tail;

        return pivot;

    }

    void quickSort()

    {

        head = rquicksort(head, last);

        last = lastnode(head);

    }

    node \*rquicksort(node \*head, node \*last)

    {

        if (!head || head == last)

            return head;

        node \*newHead = NULL, \*newEnd = NULL;

        node \*pivot = divide1(head, last, &newHead, &newEnd);

        if (newHead != pivot)

        {

            node \*tmp = newHead;

            while (tmp->next != pivot)

                tmp = tmp->next;

            tmp->next = NULL;

            newHead = rquicksort(newHead, tmp);

            tmp = lastnode(newHead);

            tmp->next = pivot;

        }

        pivot->next = rquicksort(pivot->next, newEnd);

        return newHead;

    }

    node \*lastnode(node \*last)

    {

        while (last->next != NULL)

            last = last->next;

        return last;

    }

    void stack\_adding\_in\_linkedlist(stringdata a)

    {

        node \*temp = new node;

        siz++;

        temp->data = a;

        temp->next = head;

        if (last == NULL)

        {

            last = temp;

        }

        head = temp;

    }

    void queue\_adding\_data(stringdata d)

    {

        node \*temp = new node;

        siz++;

        temp->data = d;

        temp->next = NULL;

        if (head == NULL)

        {

            head = temp;

            last = temp;

        }

        else

        {

            last->next = temp;

            last = last->next;

        }

    }

    void add\_to\_sorted\_linked\_list(stringdata d)

    {

        node \*temp = new node;

        temp->data = d;

        node \*previous;

        node \*current = head;

        if (current->data.age > d.age)

        {

            temp->next = current;

            head = temp;

            return;

        }

        while (current != NULL && current->data.age < d.age)

        {

            previous = current;

            current = current->next;

        }

        previous->next = temp;

        temp->next = current;

        if (current == NULL)

        {

            last = temp;

        }

        siz++;

    }

};

class queuec

{

private:

    linkedlist ll;

public:

    void push(stringdata a)

    {

        ll.queue\_adding\_data(a);

    }

    void show()

    {

        ll.printlinkedlist();

    }

    stringdata get()

    {

        stringdata ele;

        ele = ll.delete\_head();

        return ele;

    }

    int size()

    {

        return ll.size();

    }

    bool empty()

    {

        return size() == 0;

    }

};

class stack

{

private:

    linkedlist ll;

public:

    void show()

    {

        ll.printlinkedlist();

    }

    void push(stringdata a)

    {

        ll.stack\_adding\_in\_linkedlist(a);

    }

    stringdata get()

    {

        stringdata ele;

        ele = ll.delete\_head();

        return ele;

    }

    int size()

    {

        return ll.size();

    }

    bool empty()

    {

        return size() == 0;

    }

};

class binaryTree

{

private:

    treeNode \*root;

public:

    binaryTree()

    {

        root = NULL;

    }

    void insert(stringdata d)

    {

        root = add(root, d);

    }

    treeNode \*add(treeNode \*root, stringdata d)

    {

        if (root == NULL)

        {

            root = new treeNode;

            root->data = d;

            root->leftchild = root->rightchild = NULL;

            return root;

        }

        queue<treeNode \*> q;

        q.push(root);

        while (!q.empty())

        {

            treeNode \*temp = q.front();

            q.pop();

            if (temp->leftchild != NULL)

            {

                q.push(temp->leftchild);

            }

            else

            {

                temp->leftchild = new treeNode;

                temp->leftchild->data = d;

                return root;

            }

            if (temp->rightchild != NULL)

            {

                q.push(temp->rightchild);

            }

            else

            {

                temp->rightchild = new treeNode;

                temp->rightchild->data = d;

                return root;

            }

        }

    }

    void preorder(treeNode \*root)

    {

        if (root == NULL)

        {

            return;

        }

        root->data.print();

        cout << endl;

        preorder(root->leftchild);

        preorder(root->rightchild);

    }

    void printPreorder()

    {

        preorder(root);

    }

    void postorder(treeNode \*root)

    {

        if (root == NULL)

        {

            return;

        }

        postorder(root->leftchild);

        postorder(root->rightchild);

        root->data.print();

        cout << endl;

    }

    void printPostorder()

    {

        postorder(root);

    }

    linkedlist inorder(treeNode \*root, linkedlist \*ll)

    {

        if (root == NULL)

        {

            return \*ll;

        }

        inorder(root->leftchild, ll);

        ll->queue\_adding\_data(root->data);

        inorder(root->rightchild, ll);

        return \*ll;

    }

    linkedlist printinorder()

    {

        linkedlist ll;

        ll = inorder(root, &ll);

        return ll;

    }

};

int main()

{

    ifstream file("data.txt");

    string line;

    binaryTree bt;

    stringdata dat;

    string age;

    queuec qr;

    queuec cc;

    stack s;

    queuec q;

    if (file.is\_open())

    {

        while (getline(file, line))

        {

            stringstream ss(line);

            getline(ss, dat.name, ',');

            getline(ss, age, ',');

            getline(ss, dat.dateofbirth, ',');

            dat.age = stoi(age);

            q.push(dat);

            qr.push(dat);

        }

        file.close();

    }

    cout << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl;

    cout << "Press any key(file to queue and print the queue)";

    getch();

    cout << endl

         << endl;

    while (!q.empty())

    {

        stringdata aa = q.get();

        aa.print();

        cc.push(aa);

        cout << endl;

    }

    cout << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl;

    cout << "Press any key (reverse the queue by stack ad print)";

    getch();

    cout << endl

         << endl;

    while (!cc.empty())

    {

        s.push(cc.get());

    }

    while (!s.empty())

    {

        q.push(s.get());

    }

    while (!q.empty())

    {

        stringdata aa = q.get();

        aa.print();

        cc.push(aa);

        cout << endl;

    }

    cout << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl;

    cout << "Press any key (print the original queue) ";

    getch();

    cout << endl

         << endl;

    qr.show();

    cout << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl;

    cout << "Press any key (queue to binary tree in levelwise and print preorder)";

    getch();

    cout << endl

         << endl;

    while (!qr.empty())

    {

        bt.insert(qr.get());

    }

    bt.printPreorder();

    cout << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl;

    cout << "Press any key (print postorder)";

    getch();

    cout << endl

         << endl;

    bt.printPostorder();

    cout << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl;

    cout << "Press any key (insert in linkedlist)PRINT THE LIST";

    getch();

    cout << endl

         << endl;

    linkedlist ll = bt.printinorder();

    ll.printlinkedlist();

    cout << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl;

    cout << "Press any key (sort the list and print)";

    getch();

    cout << endl

         << endl;

    ll.quickSort();

    ll.printlinkedlist();

    cout << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl

         << endl;

    cout << "Press any key (upadate the list)";

    getch();

    cout << endl

         << endl;

    cout << "Enter the Name:" << endl;

    getline(cin, dat.name);

    cout << "Enter the Age:" << endl;

    cin >> dat.age;

    cout << "Enter the Date Of Birth:" << endl;

    cin >> dat.dateofbirth;

    cout << endl

         << endl;

    ll.add\_to\_sorted\_linked\_list(dat);

    ll.printlinkedlist();

    cout << endl;

    cout << "WOW WE HAVE DONE SUCCESFULLY" << endl

         << "PRESS ANY KEY TO EXIT...";

    getch();

    cout << endl

         << endl;

    return 0;

}