ĐỀ THI GIỮA KÌ 21CTT5

Thời gian làm bài: 60 phút

Dữ liệu sử dụng trong đề bài là dữ liệu tên trẻ sơ sinh được thống kê trong năm 1880 và 1881 có nội dung như sau:

```
Name Year Gender Count
Mary 1880 F 7065
Anna 1880 F 2604
Emma 1880 F 2003
Elizabeth 1880 F 1939
Minnie 1880 F 1746
Margaret 1880 F 1578
Ida 1880 F 1472
Alice 1880 F 1414
Bertha 1880 F 1320
Sarah 1880 F 1288
Annie 1880 F 1258
Clara 1880 F 1226
Ella 1880 F 1156
Florence 1880 F 1063
Cora 1880 F 1045
Martha 1880 F 1040
Laura 1880 F 1012
```

Cho các định nghĩa struct sau:

```
struct BabyName
{
    string name;
    string gender;
    int year;
    int count;
};
```

```
struct Node
{
    BabyName data;
    Node* pnext;
};
```

```
struct LinkedList
{
    Node* pHead;
    Node* pTail;
};
```

Thực hiện các yêu cầu sau:

Câu 1 (4 điểm)

1. (2 điểm) Đọc dữ liệu tên trẻ em lưu vào danh sách liên kết
   - `LinkedList* readBabyNames(string filename);`
   - Input: `filename` - tệp tin dữ liệu *"data.txt"*
   - Output: Danh sách liên kết `LinkedList`
2. (1 điểm) Từ DSLK trên tạo 2 dslk mới. DSLK thứ 1 chỉ chứa tên các trẻ em được sinh ra trong năm 1880 .DSLK thứ 2 chúa tên các trẻ em đươc sinh ra trong năm 1881.

- `LinkedList* splitBabyNames(LinkedList *list, int year)`
- Output: Danh sách tên trẻ em trong năm `year`

3. (1 điểm) Gộp 2 DSLK phía trên bằng cách: những tên trẻ em đều có trong cả 2 năm (và chung giới tính) thì gộp lại làm 1, lúc này `count` sẽ bằng tổng `count` của cả 2 năm.

   - `LinkedList* mergeLinkedList(LinkedList* list1, LinkedList* list2)`

Câu 2 (3 điểm)

Chương trình sau thực hiện tìm tất cả các hướng đi trong một ma trận 2 chiều MxN bắt đầu tại vị trí (i,j) cho trước và kết thúc tại vị trí cuối cùng của ma trận (M-1, N-1). Biết rằng đường đi chỉ có thể đi xuống, đi qua phải hoặc đi theo đường chéo (hướng xuống).

```
void printPaths(int** matrix, vector<int> &route, int len_route, int i, int j, int M, int N)
{
    // MxN matrix
    if(M == 0 || N == 0)
    {
        return;
    }

    // if the last cell is reached
    if(i == M-1 && j == N-1)
    {
        // print the route
        for (int k = 0; k < len_route; k++)
        {
            cout << route[k] << " ";
        }
        cout << matrix[i][j] << endl;
        return;
    }

    // add the current cell to route
    route.push_back(matrix[i][j]);
    len_route += 1;

    // move down
    if (i + 1 < M)
    {
        printPaths(matrix, route, len_route, i+1, j, M, N);
    }

    // move right
    if (j + 1 < N)
    {
        printPaths(matrix, route, len_route, i, j+1, M, N);
    }

    // move diagonally
    if (i + 1 < M && j + 1 < N)
    {
        printPaths(matrix, route, len_route, i+1, j+1, M, N);
    }

    // backtrack
```

```
    route.pop_back();
}

void Bai02()
{
    int M = 3, N=3;
    int** matrix = new int*[M];

    for(int i =0; i < M; i++)
    {
        matrix[i] = new int[N];
    }

    for (int i = 0; i < M*N; i++)
    {
        matrix[i/N][i%N] = i;
    }

    vector<int> route;
    int len_route = 0;
    int i = 0, j = 0;

    // Goi ham printPaths
    printPaths(matrix, route, len_route, i, j, M, N); // Ban co the thay doi dong nay neu co thay doi tham so
        cua ham

    // In ra so phep gan v so phep so sanh
    // CODE HERE
}
```

Sinh viên điều chỉnh mã nguồn của hàm `PrintPaths` để đếm số phép gán và số phép so sánh của hàm `PrintPaths`. Sau đó in ra số phép gán và số phép so sánh trong `Bai02()`
Lưu ý: Bỏ qua chi phí của các phương thức `pop_back()` và `push_back()`

Câu 3 (3 điểm)

(2 điểm) Cho trước một mảng số nguyên `arr` và một số k, trả vệ số lượng **cặp** số nguyên trong mảng (không trùng lặp, không hoán vị). Biết rằng các cặp số nguyên này phải thỏa mãn điều kiện: trị tuyệt đối khoảng cách giữa 2 số phải bằng k (nghĩa là: `(abs(arr[i] - arr[j]) == k)`

- **Ví dụ**: arr = [3,1,4,1,5], k = 2
- **Output**: 2
- **Giải thích**: Có 2 cặp đó là (1,3) và (3,5). Mặc dù có 2 số 1 do đó có thể tạo ra 2 cặp (1,3) tuy nhiên chúng trùng nhau nên ta không tính. Ngoài ra (1,3) và (3,1) được xem như 1 cặp vì ta không tính hoán vị của chúng.

```
int countPairs(int* arr, int n, int k);
```

(1 điểm) Hãy đảm bảo chi phí thuật toán của bạn về mặt thời gian có độ phức tạp nhỏ hơn O(n^2)